# An Analog VLSI Recurrent Neural Network Learning a Continuous-Time Trajectory

Gert Cauwenberghs, *Member, IEEE*

*Abstract*—**Real-time algorithms for gradient descent supervised learning in recurrent dynamical neural networks fail to support scalable VLSI (very large scale integration) implementation, due to their complexity which grows sharply with the network dimension. We present an alternative implementation in analog VLSI, which employs a stochastic perturbative algorithm to observe the gradient of the error index directly on the network in random directions of the parameter space, thereby avoiding the tedious task of deriving the gradient from an explicit model of the network dynamics. The network contains six fully recurrent neurons with continuous-time dynamics, providing 42 free parameters which comprise connection strengths and thresholds. The chip implementing the network includes local provisions supporting both the learning and storage of the parameters, integrated in a scalable architecture which can be readily expanded for applications of learning recurrent dynamical networks requiring larger dimensionality. We describe and characterize the functional elements comprising the implemented recurrent network and integrated learning system, and include experimental results obtained from training the network to produce two outputs following a circular trajectory, representing a quadrature-phase oscillator.**

## I. Introduction

Analog VLSI (very large scale integration) implementations of neural networks with learning capabilities have received much attention lately, and several working analog chips or systems employing analog chips incorporating learning have been reported [1]–[9], among others. The advantages of using analog VLSI as technology medium for special-purpose neural-network implementations include the inherent parallelism of the summing operations [10] and the compact size and low power consumption of the elements performing the local processing functions [11], [12]. Some disadvantages attributed to analog VLSI, in general, are the limited available dynamic range and the strong requirements on precise matching between components to achieve a reasonable degree of accuracy. The sensitivity to precision of implementation, however, depends strongly on the system-level specifications. One of the desired properties of artificial neural networks is exactly "graceful" degradation of performance under errors in the implementation, such as offsets in a typical analog VLSI process, through redundancy in distributed representation [13]. For learning neural networks, in particular, the effects of offsets and mismatches in analog hardware implementations can be significantly reduced by "learning" the set of parameters directly on the implemented network, rather than programming the network with the parameter values obtained from learning off-line using a model of the implemented network [14], [15]. Parallel architectures for fast and efficient learning, embedded with the implemented network, can be obtained for certain classes of learning algorithms, mostly those based on incremental outer-product rules [6]–[8] and other local update algorithms [1]–[5], [9]. On the other hand, the learning performance of such integrated learning networks may still be affected by the analog precision of the implemented learning functions themselves, depending on the nature of the algorithm used.

Virtually all of the learning hardware implementations of neural networks which have been developed so far exclude dynamical effects in the inputs and outputs, basically for applications of pattern recognition and association of input–output pairs. Such applications typically deal with feedforward networks, for which the learning is fairly standardized [13] and easy to implement in VLSI. VLSI learning architectures, capable of training recurrent networks as well, have been reported [1], [7], [16], for learning fixed point attractors discarding transient response. While analog recurrent networks learning time-varying features offer a wide range of attractive applications, e.g., for process control and identification of dynamical systems [17], their implementation in special-purpose hardware has currently not been demonstrated. One factor seriously inhibiting implementation is the complexity of the available learning algorithms in a dynamic setting. Several versions of gradient descent algorithms for supervised learning in dynamical recurrent neural networks exist [18]–[23]. None of those, however, support a scalable implementation for on-line (real-time) operation in a two-dimensional (2–D) arrangement, as required for a typical VLSI process technology.

In the following, we describe an analog VLSI system with simple and scalable architecture for learning in dynamical recurrent neural networks, and present experimental results obtained from a CMOS single chip implementation. The learning architecture employs a stochastic perturbative algorithm which probes the dependence of the network error on the parameters directly, rather than deriving an estimate of the gradient based on an explicit model of the network dynamics [25]. The direct approach of observing the error gradient on the physical

Manuscript received December 27, 1993; revised October 9, 1994. This work was supported by ARPA/ONR under Grant N0014-92-J-1891.

The author was with the Department of Electrical Engineering, California Institute of Technology, Pasadena, CA, 91125 USA. He is now with the Department of Electrical and Computer Engineering, Johns Hopkins University, Baltimore, MD, 21218-2686 USA.

Publisher Item Identifier S 1045-9227(96)00168-3.

network relates to techniques of stochastic approximation [26], [27] and finds parallels in other recent algorithmic developments and hardware learning architectures as well [28]–[33]. In addition to the significant reduction in implementation complexity, the direct observation of the gradient avoids the offsets which may arise in a derived gradient estimate, due to mismatches between the assumed network model and its physical implementation. In fact, the direct approach further extends directly toward optimization of arbitrary parameter-driven dynamical systems, of which the dependence of the error index on the parameters and a model of the network does not need to be known or specified [30]. As a demonstration of principle, we have implemented a small recurrent neural network with continuous-time dynamics, integrated with the learning circuitry on a CMOS chip, and have tested its learning performance with a simple trajectory learning task, representative of more general and useful applications for system identification and adaptive control. By virtue of the scalable and modular learning architecture, the chip presented here can be extended, with minor modifications to the internal structure of the cells, to accommodate applications of learning dynamical recurrent systems of larger dimensionality.

A brief specification of the implemented network model and learning algorithm is given in the next section. Section III describes the architecture of the integrated network and learning system, and the circuit implementation of its functional elements. Experimental results on learning a periodical continuous-time trajectory are analyzed in Section IV, and Section V concludes with the strengths and limits of the implemented system.

## II. System Architecture

The implemented network contains six fully interconnected recurrent neurons with continuous-time dynamics

$$\tau \frac{d}{dt} x_i = -x_i + \sum_{j=1}^{6} W_{ij}\, \sigma(x_j - \theta_j) + y_i \qquad (1)$$

with $x_i(t)$ the neuron state variables constituting the outputs of the network, $y_i(t)$ the external inputs to the network, and $\sigma(.)$ a sigmoidal activation function. The time constant $\tau$ governs the dynamics of the network, providing first order low-pass filtering in the evolution of the neuron state variables. A more elaborate model of neural dynamics would incorporate individual adjustable time constants at the level of the synaptic contributions [24]. The value for $\tau$ is kept fixed and uniform in the present implementation. The free parameters, to be optimally adjusted by the learning process, constitute the 36 connection strengths $W_{ij}$ and the 6 thresholds $\theta_j$. Since the implemented learning algorithm does not distinguish parameters based on a presumed model of the network structure, the parameters $W_{ij}$ and $\theta_j$ will be considered below as components of one single parameter vector $\mathbf{p}$.

The network output consists of the two neuron signals $x_1(t)$ and $x_2(t)$, which are trained in supervised mode with target output signals $x_1^T(t)$ and $x_2^T(t)$ presented to the network. For the specific trajectory learning example used in the experiments, no inputs $y_i(t)$ are specified along with the target outputs $x_i^T(t)$. The implemented architecture, however, allows for more general learning tasks typical in applications of identification and control, for which the target signals comprise the desired dynamical response of the network (or plant) under activation of inputs $y_i(t)$.

Supervised learning of the time-varying targets $x_i^T(t)$ consists of minimizing, in principle, the time average

$$\mathcal{E}(\mathbf{p}) = \lim_{T\to\infty} \frac{1}{2T} \int_{-T}^{T} e(\mathbf{x}^T(t), \mathbf{x}(t))dt \qquad (2)$$

of the network output error

$$e(\mathbf{x}^T(t), \mathbf{x}(t)) = \sum_{k=1}^{2} |x_k^T(t) - x_k(t)|^\nu \qquad (3)$$

with a distance metric of norm $\nu$. The infinite time window for integration of the network error (2) is not practical for real-time implementation, but a fair approximation to the error average can be obtained by replacing the integral (2) by the output of a low-pass filter operating on the instantaneous network error (3). The approximation is particularly valid in case the training sequence of input and target signals is periodic, with a repetition rate significantly higher than the cutoff frequency of the filter. The condition of periodicity is needed to ensure consistency in the outcome of the error measure taken at different instances in time. While alternative on-line versions of the error measure $\mathcal{E}(\mathbf{p})$ can be formulated, which allow for faster learning speed and better convergence behavior, the low-pass filtered version used here is easier to implement while still allowing to demonstrate the general principle. The choice implies the restriction of a periodic format for the training signals, which we adopt in the learning experiment.

A stochastic perturbative algorithm, which directly observes the dependence of the error measure $\mathcal{E}(\mathbf{p})$ on the parameters, is used for error descent learning [25]. The learning algorithm iteratively specifies incremental updates in the parameter vector $\mathbf{p}$ as

$$\mathbf{p}^{(k+1)} = \mathbf{p}^{(k)} - \mu\, \hat{\mathcal{E}}^{(k)}\, \pi^{(k)} \qquad (4)$$

with the differentially perturbed error

$$\hat{\mathcal{E}}^{(k)} = \frac{1}{2}\left(\mathcal{E}(\mathbf{p}^{(k)} + \pi^{(k)}) - \mathcal{E}(\mathbf{p}^{(k)} - \pi^{(k)})\right) \qquad (5)$$

obtained from a two-sided parallel activation of fixed-amplitude random perturbations $\pi_i^{(k)}$ onto the parameters $p_i^{(k)}$; $\pi_i^{(k)} = \pm\sigma$ with equal probabilities for both polarities. The algorithm basically performs random-direction descent of the error as a multidimensional extension to the Kiefer–Wolfowitz stochastic approximation method [27], and several related variants have recently been proposed for optimization [28], [29] and hardware learning [30]–[33]. When applied to on-line supervised learning in recurrent dynamical systems, the update rule (4) and (5) provides a net computational efficiency rivaling that of alternative techniques based on exact gradient descent, at a much reduced complexity of implementation [25].

To facilitate learning, a teacher forcing signal is injected into the external network input $\mathbf{y}$ according to

$$y_i(t) = \lambda\, \gamma(x_i^T(t) - x_i(t)), \quad i = 1, 2 \tag{6}$$

providing a feedback mechanism that forces the network outputs toward the targets [20]. A symmetrical and monotonically increasing function for $\gamma(.)$ serves this purpose. While not needed for general applications, the feedback signal provided by the teacher forcing (6) is quite essential in the case of trajectory learning, in absence of training input signals $\mathbf{y}(t)$, since otherwise no mechanism is available to synchronize the dynamics of the network $\mathbf{x}(t)$ with that of the target output signals $\mathbf{x}^T(t)$ during the process of learning. In principle, the teacher forcing signal vanishes near convergence, where the targets $x_i^T(t)$ and outputs $x_i(t)$ coincide. In practice, the teacher forcing amplitude $\lambda$ needs to be gradually attenuated along the learning process, as to suppress any bias in the network outputs that might result from residual errors at convergence [20].

## III. Analog VLSI Implementation

The network and learning circuitry are implemented on a single analog CMOS chip. While most learning functions, including generation of the random perturbation bits, are integrated on-chip along with the implemented network, some global and higher-level learning functions of low dimensionality, such as the evaluation of the error (2) and construction of the perturbed error (5), are performed outside the chip. The off-chip implementation of the higher-level global functions allows for greater flexibility in tailoring the learning process. Conversely, the lower-level learning functions involving the full dimensionality of the parameter vector are implemented locally on-chip, where they are performed in parallel for optimal efficiency. The implemented array structure of learning cells, providing locally for the parameter updates, additionally serves to refresh the volatile parameter values for long-term storage after learning is completed. The structure and functionality of the implemented network and learning circuitry are described below.

### A. Implementation Floor Plan

The floor plan of the VLSI network of synapses is organized in the usual 2-D array configuration for interconnecting two layers of neurons, with input and output lines running across the array of synapses in orthogonal directions, whereby two particular input and output lines intersect at the location of the synapse cell interconnecting the corresponding input and output neurons [10]. The 2-D array of synapses interfaces at the outside boundary with a linear array of output neuron cells, collecting synaptic contributions corresponding from the output lines for further processing to construct the neuron outputs. With synapse cells supporting analog current-mode outputs, the synaptic contributions are collected at the output simply by dumping the analog output currents supplied by the synapse cells directly onto the output lines. This current-mode output arrangement combined with the 2-D array configuration of synapse cells offer a simple, modular and scalable VLSI architecture for implementing a fully interconnected neural network, adopted here. Continuous-time recurrent dynamics in the neuron state variables, in accordance with (1), is obtained by feeding the first order low-pass filtered neuron outputs back into the neuron input layer [34].

To maintain the scalable and modular architecture of the implemented network while integrating local learning functions onto the chip, the added complexity of global on-chip interconnects necessitated by the implemented learning algorithm cannot exceed the $N^2$ limit imposed by the 2-D arrangement of cells. For feedforward steady-state neural networks, the typically used incremental outer-product learning rules, such as the delta rule and backpropagation for supervised learning and Hebb-type rules for unsupervised learning, support a scalable and integrated 2-D architecture intertwined with the implemented network [6]. No scalable on-line extensions to the outer-product rules are available, however, for on-chip learning of time-varying features in dynamical recurrent networks.

The stochastic perturbative algorithm, in contrast, supports a simple implementation structure integrated locally with the network, with basically no requirements for global interconnects other than those already supplied by the network itself, for observation of the network error $\mathcal{E}(\mathbf{p})$. The implementation structure comprises an array of identical local learning cells superimposed onto the array structure of network cells, each instance connecting locally to the analog storage node of one particular parameter $W_{ij}$ or $\theta_j$ in the network. Except for a few global learning signals, distributed in identical format to all local learning cells, no communication across different cells is needed to perform the learning functions, with the learning cells basically operating autonomously to supply the parameters updates from the locally generated parameter perturbations. Because of the model independence of the learning algorithm, the implementation structure providing the learning functions can be generalized directly to other parameter-driven networks, say with nonstandard or adjustable configuration of the processing cells and their interconnections [35]. The implemented learning functions essentially retain their relative complexity regardless of the structure of the implemented network in which they are embedded.

Fig. 1 shows the structural organization of the integrated network and learning functions, comprising an array of synapse and threshold parameter cells each including local learning and storage facilities, and a linear array of neuron output cells at the periphery. Auxiliary provisions to support local generation of the parameter perturbations during learning mode, and to refresh the volatile parameters during storage mode, are also indicated. Global connections for distributing certain scalar signals uniformly to all cells in the array are omitted from Fig. 1 for clarity. The functional specification and circuit implementation of most of the elements in Fig. 1 are further elaborated in the text below.

### B. Network Circuitry

A transconductance current-mode approach is adopted for the implementation of the network, allowing continuous-time

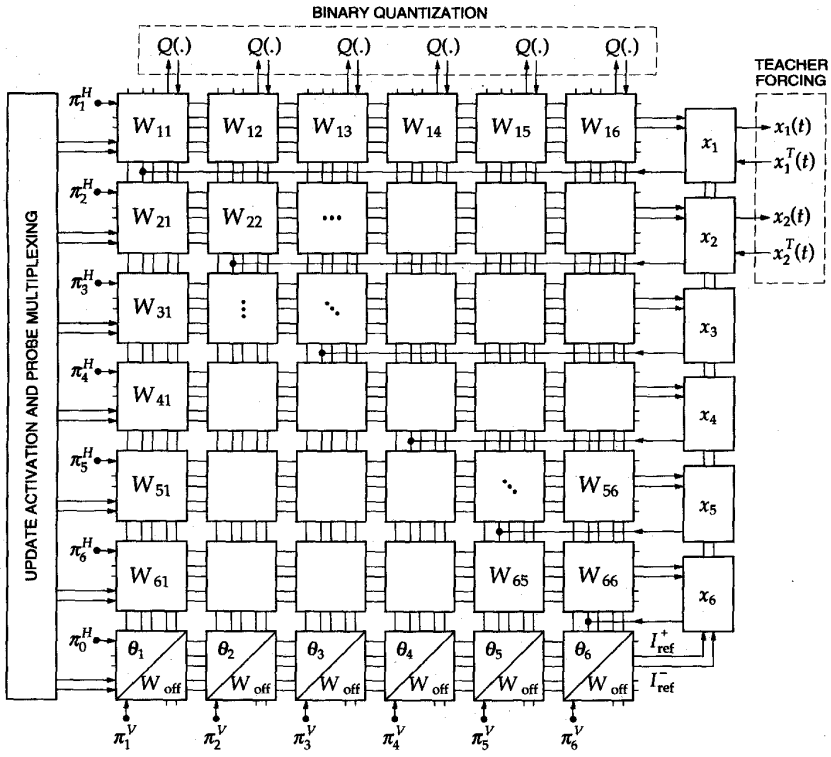



Fig. 1. Array structure of the network, containing parameter cells with integrated learning and storage functions.

evolution of the network state variables. Through simple transconductance circuitry, which includes regulated cascode triode structures and double-stack differential pairs described below, a wide dynamic range is achieved for the neuron voltages and parameter values at relatively low levels of power dissipation.

A high-output impedance transconductance element with wide voltage range, which is used in both the synapse and neuron cell circuitry, is shown in Fig. 2(a), together with its symbolic representation for further reference in subsequent figures. The device comprises a MOS (metal-oxide semiconductor) transistor MT biased in the linear triode region, connected to a cascode transistor MC which by means of a high-gain feedback circuit [36], [37] provides high output impedance while forcing the drain of the triode transistor MT to a constant voltage level. The triode drain voltage $V_d^{\mathrm{MT}}$ is primarily set by the control voltage $V_c$ of the regulated cascode circuit, and is largely independent of the triode gate voltage and cascode drain voltage. Therefore, the supplied output current $I_{\mathrm{out}}$ is proportional to the input voltage $V_{\mathrm{in}}$ while invariant to the output voltage $V_{\mathrm{out}}$, implementing a linear transconductance element operating over a fairly wide voltage range of $V_{\mathrm{in}}$. A bias circuit to generate the control voltage $V_c$ from a desired level for $V_d^{\mathrm{MT}}$, specified by an externally supplied voltage $V_d$, is shown in Fig. 2(b). The bias circuit allows for approximately linear control of the transconductance value of the element by means of the voltage $V_d$. Only one instance of the bias circuit needs to be provided for every family of elements with a common transconductance value.

An active resistive element with wide voltage range is effectively obtained by combining two instances of the triode transconductor with a current mirror, shown in Fig. 3(a). The variable resistive element, injecting a current into the $V_{\mathrm{out}}$ terminal proportional to the voltage difference $V_{\mathrm{off}} - V_{\mathrm{out}}$ and the control voltage $V_d$, is used particularly for conversion between voltage and current formats of the neuron state variables of the network. The measured I-V characteristics of the implemented variable resistor are given in Fig. 3(b), indicating fairly strong nonlinearities in the implemented resistance. Certainly, other CMOS (complimentary MOS) circuit implementations of variable transconductance and resistive elements, with better wide-range linear I-V characteristics, are available, e.g., [3] and [38]. Linearity, however, is not an absolute requirement in the implementation of an intrinsically nonlinear system. Nonlinearities and other errors in the implemented network furthermore provide an opportunity to experimentally verify the robustness of the learning performance in the presence of model mismatches. A primary motivation for using the transconductance element of Fig. 2(a), besides the high-impedance current output and the wide dynamic range of the voltage input, is the ohmic and capacitive decoupling between the voltage input and current output nodes, allowing for fairly insulated capacitive analog storage on the $V_{\mathrm{in}}$ terminal as needed to implement volatile adjustable parameters.

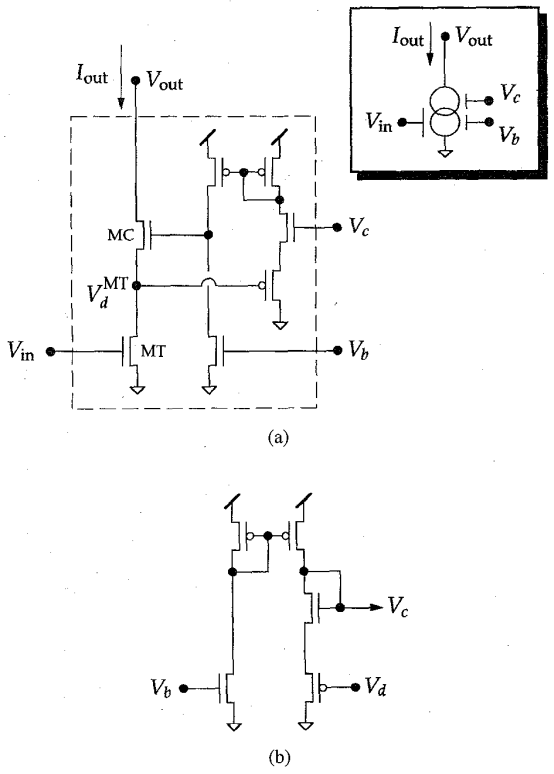



Fig. 2. Wide range CMOS triode transconductance element with regulated cascode high impedance output. (a) Circuit diagram. (b) Bias generating circuit.

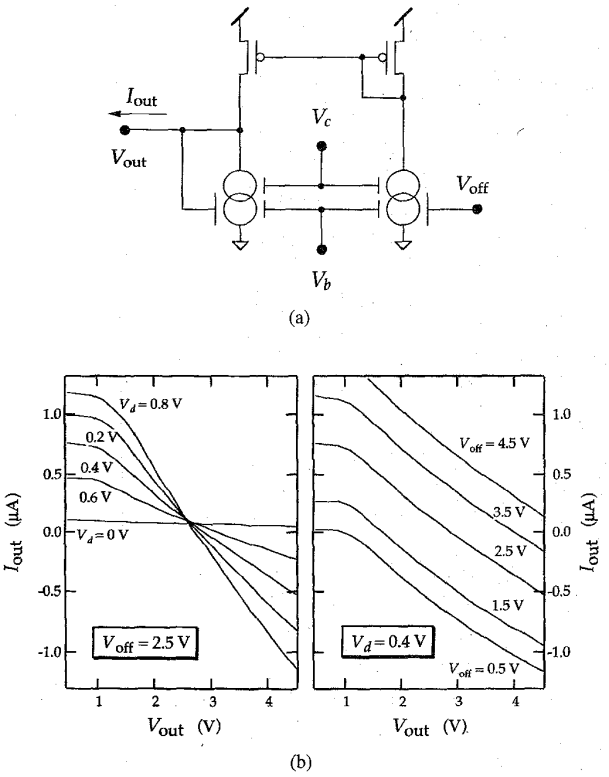



Fig. 3. Wide range active CMOS resistive element. (a) Circuit diagram. (b) Measured I-V characteristics.

Fig. 4 shows the schematic of the main synapse and neuron cell circuitry employed in the network array and its peripherals. A synapse cell of single polarity is shown in Fig. 4(a). A constant current $I_{ij}$, linear in the voltage $W_{ij}$ over a wide range, is provided by an instance of the triode multiplier of Fig. 2(a). The synaptic current $I_{ij}$ feeds into a differential pair, injecting the current $I_{ij}\,\sigma(x_j-\theta_j)$ differentially into the diode-connected $I_{\text{out}}^+$ and $I_{\text{out}}^-$ output lines. The double-stack transistor configuration of the differential pair offers an expanded linear sigmoid range at modest $I_{ij}$ current levels [39].

The summed output currents $I_{\text{out}}^+$ and $I_{\text{out}}^-$ of a row of synapses are collected in the output cell of the corresponding neuron $x_i$, Fig. 4(b). The diode connection of the load transistors on the output lines of Fig. 4(a) provides normalization of the collected output currents to an appropriate level, regardless of the number of participating synaptic cells feeding into the neuron output, thereby supporting scalable expansion of the network architecture [40]. The neuron output cells also receive two common reference currents $I_{\text{ref}}^+$ and $I_{\text{ref}}^-$ obtained from one separate row of reference synapses, identical in structure to the other synaptic cells. The reference synapses, represented in the bottom row of the array of Fig. 1, are supplied uniformly with a synaptic strength $W_{\text{off}}$, defining a common offset for the synaptic values $W_{ij}$ serving as an effective zero level reference for four-quadrant operation of the synapses [6]. By combining the currents $I_{\text{out}}^+$, $I_{\text{out}}^-$, $I_{\text{ref}}^+$ and $I_{\text{ref}}^-$ through mirror and summing operations, the neuron cell constructs the output current

$$\begin{aligned} I_{\text{out}} &= (I_{\text{out}}^+ - I_{\text{out}}^-) - (I_{\text{ref}}^+ - I_{\text{ref}}^-) \\ &= \sum_{j=1}^{6} g_c\, W_{ij}\, \sigma(x_j-\theta_j) - \sum_{j=1}^{6} g_c\, W_{\text{off}}\, \sigma(x_j-\theta_j) \\ &= g_c \sum_{j=1}^{6} (W_{ij} - W_{\text{off}})\, \sigma(x_j-\theta_j). \end{aligned} \tag{7}$$

Apart from a factor $g_c$, which accounts for the triode element transconductance and current scaling factors on the output line, the current (7) corresponds to the summed synaptic contributions in (1). Additionally, the contribution of the external input $y_i$ to neuron $x_i$ is included in (7) by injecting a current proportional to $y_i$ into the output node. With the particular network configuration of the present implementation, for demonstration of trajectory learning, the external inputs only contain the teacher forcing signals (6) applied to the first two neurons, $x_1$ and $x_2$. A differential transconductance element with inputs $x_i$ and $x_i^T$, for forced teacher action in accordance with (6), is shown connected to the neuron output in the dashed inset of Fig. 4(b), applicable to the forced neurons $x_1$ and $x_2$ only. The transconductance element implements the teacher forcing characteristics of (6), with the function $\gamma(.)$ performed by the sigmoidal transfer function of the input differential pair, and the amplitude $\lambda$ determined by the tail current of the device, equal to the product of the triode transconductance $g_{c\,\text{out}}$ and the control voltage $V_\lambda$.

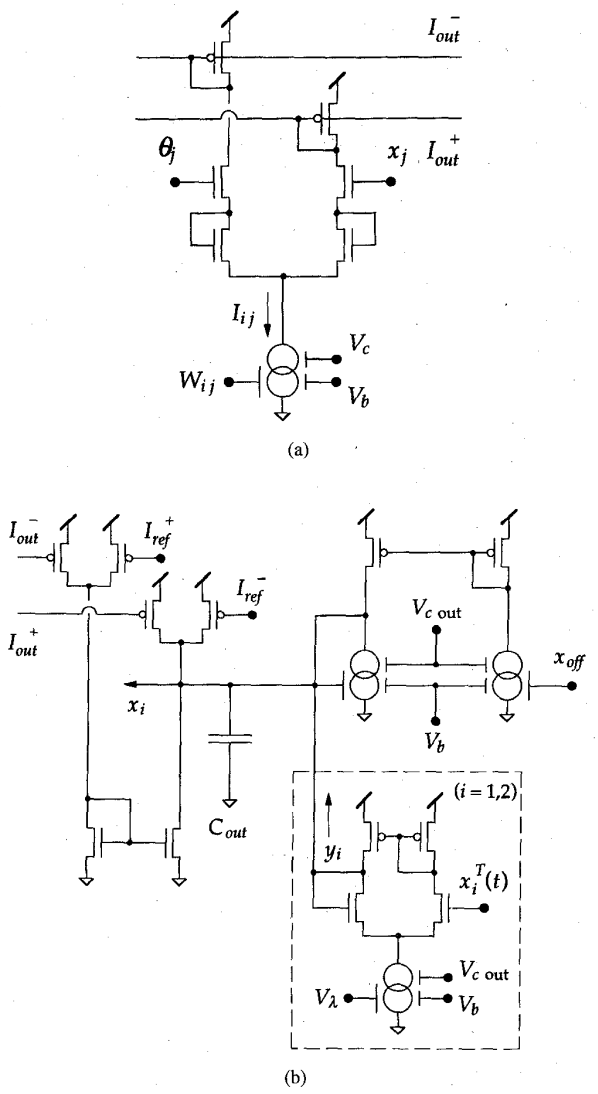



Fig. 4. Schematic of synapse and neuron network circuitry. (a) Synapse cell of single polarity. (b) Neuron output cell with current-to-voltage converter.

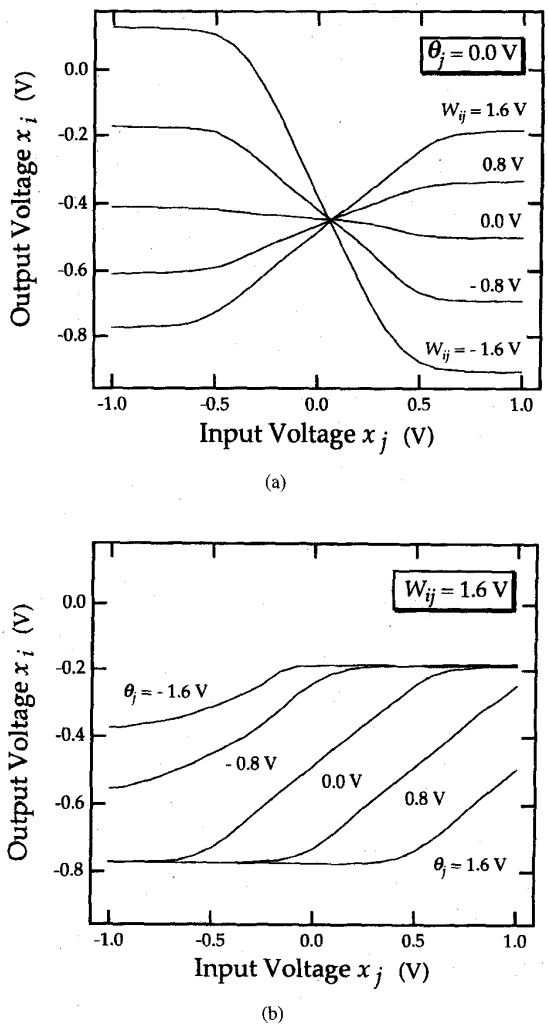



Fig. 5. Measured static synapse and neuron characteristics, for various values of (a) the connection strength $W_{ij}$ and (b) the threshold $\theta_j$.

The combined output current $I_{\text{out}}$ is converted to the neuron output voltage $x_i$ by means of an active resistive element, as described above with reference to Fig. 3(a). Besides serving to convert between current and voltage formats, the resistive element also implements the dynamics for the neuron state variables, with the time constant $\tau$ in (1) corresponding to the $RC$ product of the resistance value $1/g_{c\,\text{out}}$ and a parallel capacitance $C_{\text{out}}$. With $C_{\text{out}} = 5$ pF, the delay ranges between 20 and 200 $\mu$s, adjustable by the control voltage $V_{c\,\text{out}}$ ($V_{d\,\text{out}}$) of the regulated cascode defining the conductance $g_{c\,\text{out}}$ of the resistive element.

Fig. 5 shows the measured static characteristics of the synapse and neuron functions for different values of $W_{ij}$ and $\theta_j$ $(i = j = 1)$, obtained by disabling the neuron feedback and driving the neuron inputs of the synapse array externally. Effects of random and systematic errors in the implementation on the shape of the curves are clearly visible. In particular, the characteristics for different threshold values $\theta_j$ in Fig. 5(b) show a significant distortion due to saturation effects at the boundaries of the synapse dynamic range. Nevertheless, the discrepancy between expected and realized network characteristics is largely irrelevant, since the stochastic perturbative learning process does not assume a particular form of the network structure, or perfect model knowledge of the implemented structure. Significantly more important than the shape of the implemented network characteristics is the analog resolution it supports for the network parameters and state variables. As demonstrated in Fig. 5, the implemented network provides a dynamic range of 1 V for the neuron voltages, and 3 V of fairly linear voltage range for programming of the synapse and threshold parameters.

### C. Learning Circuitry

Fig. 6(a) shows the simplified schematic of the learning cell circuitry, replicated locally for every parameter $W_{ij}$ and $\theta_j$ in the network array of Fig. 1. As indicated above, most of the

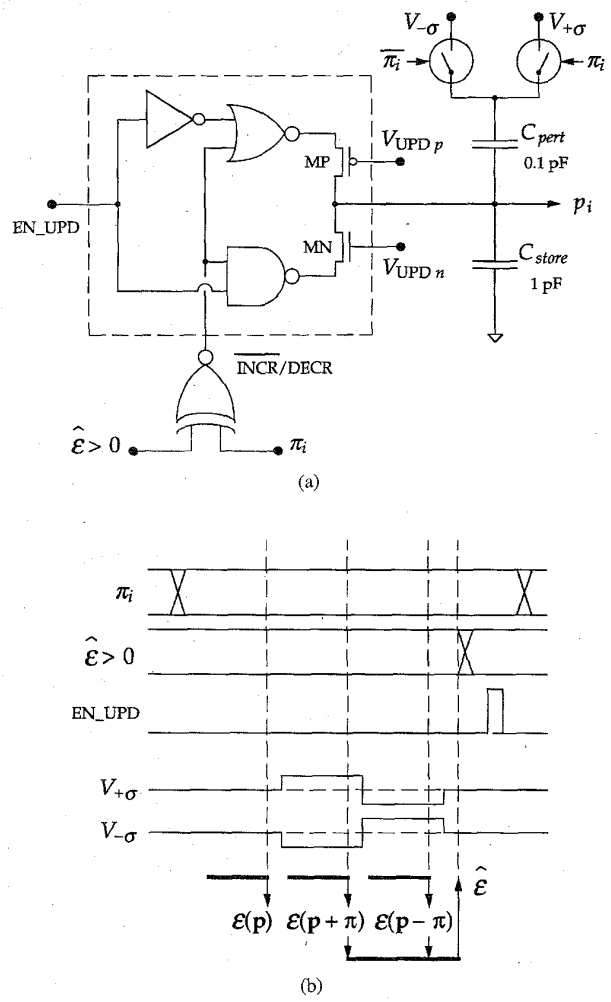



Fig. 6. Learning cell circuitry. (a) Simplified schematic. (b) Waveform and timing diagrams.

variables relating to the operation of the learning cell are local, with exception of a few global signals communicating to all cells. Global signals include the sign and the amplitude of the perturbed error $\hat{\mathcal{E}}$ and predefined control signals. The stored parameter $p_i$ and its binary perturbation $\pi_i$ are strictly local to the cell, in that they do not need to communicate explicitly to outside circuitry, except trivially through affecting the neural network it drives. This simplifies the structural organization and interconnection of the learning cells, which are integrated with the synapse and threshold cells in the network array of Fig. 1.

The parameter voltage $p_i$ is stored on the capacitor $C_{\mathrm{store}}$, and interfaces directly with the corresponding network circuitry providing the synapse or threshold function. The storage capacitor $C_{\mathrm{store}}$ furthermore couples to a second capacitor $C_{\mathrm{pert}}$ for activation of the perturbation. Since the perturbation can only take one of two values $\pm\sigma$, it is locally represented and processed in binary format. The binary perturbation is generated locally, once at the beginning of every update cycle, by means of a procedure described further below. The perturbation bit $\pi_i$ selects either of two complementary signals $V_{+\sigma}$ and $V_{-\sigma}$, supplied globally to all learning cells, for driving the coupling capacitor $C_{\mathrm{pert}}$. $V_{+\sigma}$ is selected for $\pi_i = 1$, and $V_{-\sigma}$ is selected otherwise. With the specific shape of the waveforms $V_{+\sigma}$ and $V_{-\sigma}$ depicted in Fig. 6(b), the proper sequence of perturbations activated onto the parameter is established, for successive observations of the complementary error terms in (5). In particular, both $V_{+\sigma}$ and $V_{-\sigma}$ represent the perturbation sequence $0, \pi_i, -\pi_i$, with $\pi_i = +\sigma$ for $V_{+\sigma}$ and $\pi_i = -\sigma$ for $V_{-\sigma}$. Successive observations of the network error, synchronized with the three phases of the perturbation sequence, yield estimates of the unperturbed error $\mathcal{E}(\mathbf{p})$ and the complementary perturbed errors $\mathcal{E}(\mathbf{p}+\pi)$ and $\mathcal{E}(\mathbf{p}-\pi)$, respectively. An estimate of the differential perturbed error $\hat{\mathcal{E}}$ is then constructed externally by combining the obtained values for $\mathcal{E}(\mathbf{p}+\pi)$ and $\mathcal{E}(\mathbf{p}-\pi)$, in accordance with (5). By virtue of the rigorous scheme used to activate the perturbations onto the parameters, involving mainly binary selection operations at the local implementation level, the perturbed error observations and the corresponding estimate of $\hat{\mathcal{E}}$ can be expected of reasonably high quality, provided the outcomes of the error observations are systematic and consistent in the respective parameter settings.

The obtained global value for $\hat{\mathcal{E}}$ is then used, in conjunction with the local perturbation bit $\pi_i$, to locally update the parameter value $p_i$ according to (4). For optimal learning performance, the update increment in (4) is decomposed into its amplitude and polarity components, each further being processed and activated independently. The motivation for separating both follows from accuracy considerations applying to incremental update learning rules in general. While the correct implementation of the polarity of the desired learning increments is crucial to warrant proper convergence of the parameter values, the amplitudes of the implemented increments or decrements are allowed relative errors within certain margins. Incidentally, formula (4) reveals that the increment polarity can be obtained from a binary exclusive-or operation on the polarities of the perturbed error $\hat{\mathcal{E}}$ and the perturbation bit $\pi_i$, with the parameter value being decremented if both polarities are equal, and incremented otherwise. Likewise, the increment amplitude is given by the amplitude of $\hat{\mathcal{E}}$. The binary operation to determine the increment polarity is implemented virtually error-free with simple digital CMOS circuitry. The obtained value for the polarity then activates either of a given analog increment or decrement, with amplitude proportional to $|\hat{\mathcal{E}}|$, onto the parameter value. The analog device serving the increments and decrements does not need to be excessively precise, other than required to satisfy the desired increment polarity. Requirements on relative accuracy of the increment size, scaling with the amplitude, are therefore more stringent than requirements on absolute, uniform accuracy.

A binary controlled charge pump, shown in the dashed-line inset of Fig. 6(a), is used for this purpose, offering a fine resolution of the charge increment amplitude ranging over several decades [45]. The charge pump dumps either a positive or a negative update current, of equal amplitude, onto the storage capacitor whenever it is activated by means of an EN_UPD high pulse, effecting either of a given increment or decrement on the parameter value $p_i$, respectively. The

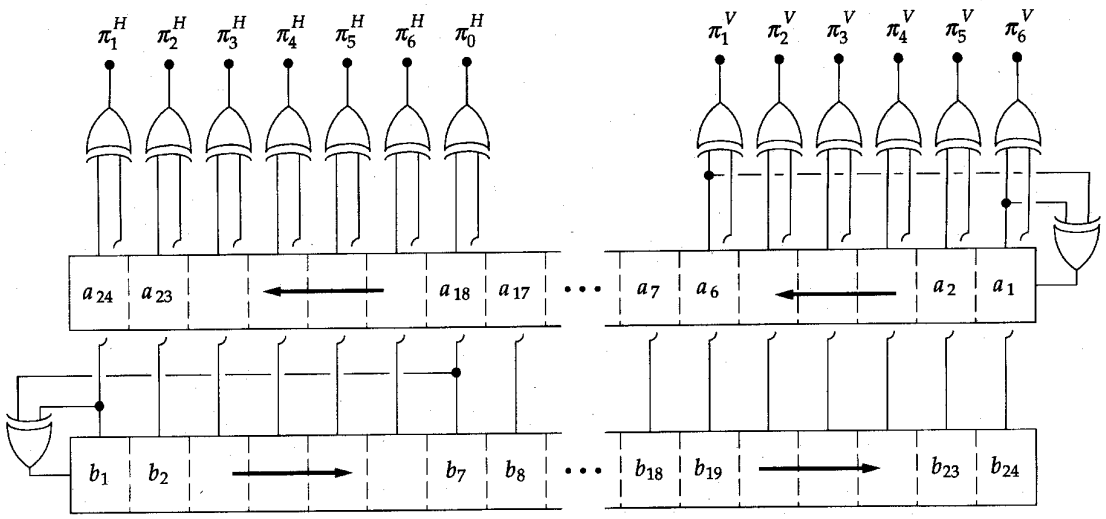



Fig. 7. Multichannel pseudorandom bit generation using linear feedback shift registers.

positive and negative currents of the device are supplied by complementary MOS transistors MP and MN. The particular transistor, supplying the current for either increment or decrement action, is activated by driving its source voltage while keeping its gate voltage fixed, thereby avoiding the typical effects of switch injection noise and clock feed through affecting switched capacitor implementations [41]. Both transistors are biased in the subthreshold region [42], [11], allowing exponential control of the current amplitude ranging down to pA levels. Consequently, the charge pump device is able to supply reliable increments and decrements of almost arbitrarily small size, as needed in the final stages of the learning near convergence.

The charge pump device interfaces with the storage capacitor $C_{\text{store}}$ to update the analog parameter $p_i$ according to the supplied analog and binary signals defining the increment amplitude and polarity. The amplitude of the incremental update, set proportionally to $|\hat{\mathcal{E}}|$, is controlled by the globally supplied $V_{\text{UPD }n}$ and $V_{\text{UPD }p}$ bias levels onto the gates of MN and MP, respectively. The polarity of the increment or decrement action is determined by the control signal DECR/$\overline{\text{INCR}}$, obtained from an exclusive-or gate operating on the polarities of the perturbed error $\hat{\mathcal{E}}$ and the perturbation bit $\pi_i$. The learning cycle is completed by activating the update with a high pulse on EN_UPD. The next learning cycle then starts with a new random bit value for the perturbation $\pi_i$.

### D. Local Generation of Random Perturbations

The random bit streams $\pi_i{}^{(k)}$ are generated on-chip using a variant on the well-known technique of obtaining pseudorandom bit sequences by means of linear feedback shift registers [43]. For optimal performance, the perturbations need to satisfy certain statistical orthogonality conditions, and a rigorous but elaborate method to generate a set of uncorrelated bit streams in VLSI has been derived [44]. To preserve the scalability of the learning architecture and the local nature of the perturbations, we have chosen a simplified scheme which drastically reduces the implementation complexity but which does not aversely affect the learning performance to first order, as verified experimentally. The array of perturbation bits, configured in a 2-D arrangement given by the location of the learning cells near the parameters in the network array, is constructed by an outer-product exclusive-or operation from two generating linear sets of uncorrelated row and column bits, $\pi_j{}^H$ $(j = 0 \cdots 6)$ and $\pi_i{}^V$ $(i = 1 \cdots 6)$, shown in Fig. 1. In particular, the values of the perturbation bits are obtained locally, inside the corresponding learning cells, by means of exclusive-or gates whose inputs connect to the locally intersecting horizontal and vertical bit lines. While deterministic relationships exist among the constructed bit values, statistically the obtained bits are mutually uncorrelated with equal probabilities for both binary outcomes, provided the generating row and column bits $\pi_j{}^H$ and $\pi_j{}^V$ are statistically uncorrelated and unbiased as well.

The row and column bits themselves are obtained from exclusive-or combinations of the parallel outputs of two counter-propagating linear feedback shift registers, with polynomial degrees 6 and 7, respectively, generating two independent pseudorandom bit sequences in time and space. The configuration of the two registers and the linear array of exclusive or-gates, combining the register outputs into the row and column bits, is shown in Fig. 7. In principle, the row and column bits could have been obtained from the parallel outputs of one single feedback register only, without the XOR operations, though such would imply correlations between the bit values over time, as any two given taps of the parallel output of a shift register represent the same bit sequence merely displaced in time. Since the parallel output bit sequences of the two registers generating the row and column bits are mutually counterpropagating and independent, propagation effects within the set of generated bits are avoided. With polynomial degrees of 6 and 7 for the two shift register bit sequences, the combination of the register outputs delivers a periodicity of $(2^6 - 1) \times (2^7 - 1) = 8,001$ [43], which is appropriately long for the perturbative learning

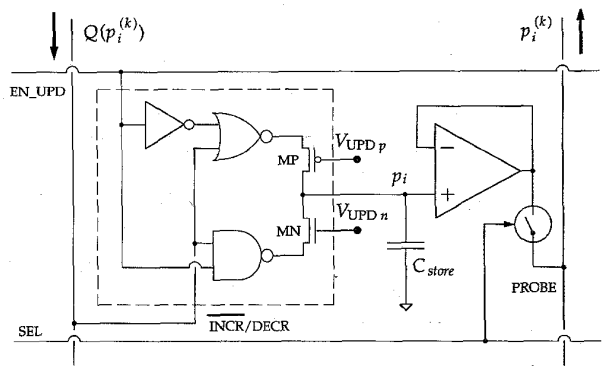



Fig. 8. Simplified schematic of the storage cell circuitry.

scheme. While longer random sequences could be obtained by choosing higher degrees for the generating polynomials of the registers, the choice was made in particular to allow for a simple implementation of the linear feedback circuitry.

### *E. Long-Term Volatile Storage*

After learning, it is desirable to retain the parameter settings established onto the network. The volatile storage of the parameter values on capacitors undergoes a spontaneous decay due to junction leakage and other drift phenomena, and needs to be refreshed periodically. The implemented chip includes local provisions for refreshing the parameters, using a local analog memory technique which does not require external storage of the parameters [45]. The parameter refresh is performed in the background, and does not interfere with the continuous-time network operation.

The implemented technique effectively quantizes a given stored analog parameter value to one of a discrete set of voltage levels, by repeatedly refreshing the analog value toward the identified discrete level to counteract the drift of the volatile storage medium. A typical scheme employed for refresh consists of identifying the discrete level closest to the stored analog parameter value, and then updating the analog value by loading the identified discrete level onto the storage device [4], [9]. The refresh scheme used here [45] is more robust to random errors, by specifying small fixed-size increments in the parameter values in the direction of the nearest discrete level, rather than completely substituting the stored analog value with the identified level. In particular, the incremental updates for partial refresh are constructed as

$$p_i{}^{(k+1)} = p_i{}^{(k)} - \delta\, Q(p_i{}^{(k)}) \tag{8}$$

with $\delta$ the increment amplitude, small compared with the separation between adjacent discrete levels, and where the binary quantization value $Q(p_i{}^{(k)})$ is obtained from the analog value $p_i{}^{(k)}$ in fairly consistent manner [45]. The binary quantization function $Q(.)$, indicated in Fig. 1, serves to determine the polarity of the direction toward the nearest discrete level, and can be implemented in practice by extracting the least significant bit (LSB) obtained from analog-to-digital conversion of the analog value [45].

For practical reasons, the refresh circuitry implementing the above analog memory technique is integrated with the learning circuitry driving the parameter storage capacitor. Incidentally, the charge pump used for the learning updates can serve directly to supply the incremental updates in the parameter values for partial refresh as well. To that purpose, probing and multiplexing circuitry are added to the learning cell of Fig. 6(a), activated in the refresh mode of operation while the learning circuitry is disabled, to access the stored parameter value for binary quantization and drive the charge pump correspondingly. The simplified schematic of the refresh circuitry, discarding the learning portion of the cell sharing the same charge pump, is shown in Fig. 8. The actual binary quantization of the stored analog values is performed sequentially outside the array of cells, in a multiplexed arrangement along each of the columns in the array, illustrated in Fig. 1, to reduce the complexity of implementation. Since the refresh rates required for the volatile storage medium are rather modest, typically in the 100 Hz range, a small set of binary quantizers, one per column in the array, provides sufficient bandwidth to accommodate multiplexing of a fairly large number of storage cells. A binary SEL signal selects the particular row of cells in the array to be refreshed at a given time, connecting the buffered stored value of a selected cell to the binary quantizer on the corresponding column. A logic high pulse on the EN_UPD line of the selected row finally establishes the partial increments (8), activating the charge pumps of the selected storage cells with polarity DECR/$\overline{\text{INCR}}$ given by the binary quantization values of the corresponding columns.

The complete circuit-level schematic of the synaptic cell $W_{ij}$ used in the array of Fig. 1, including the local functions of synaptic contribution, construction of the perturbations, stochastic error-descent learning, and long-term storage of the volatile parameters, is given in Fig. 9. The layout of the synapse cell, measuring 170 $\mu$m $\times$ 180 $\mu$m in 2 $\mu$m CMOS technology, is depicted in Fig. 10. The reference synaptic cells $\theta_j/W_{\text{off}}$ shown in the bottom row of Fig. 1, which supply the value of the locally stored threshold parameter $\theta_j$ to all synapses on the same column while providing the $W_{\text{off}}$ synaptic contributions on the reference row, adapt a circuit and layout structure almost identical to that of Figs. 9 and 10.

### *F. Global Supervision of Learning and Storage Functions*

Besides the local learning and storage functions, of which the implementation was described in detail above, their coordination and supervision requires some global functions, which operate at a higher level surpassing the individual structure of parameter cells in the array. The global functions involve a dimensionality much smaller than that of the local functions, such that a dedicated hardware implementation tuned toward optimal efficiency is not needed. On the contrary, the low-dimensional global functions can be implemented on a general-purpose platform such as a micro-controller, offering a wide spectrum of functionality which can be tuned toward optimal performance at virtually no cost in hardware efficiency. For the stochastic perturbative learning algorithm in particular, the global functions operate on scalar variables

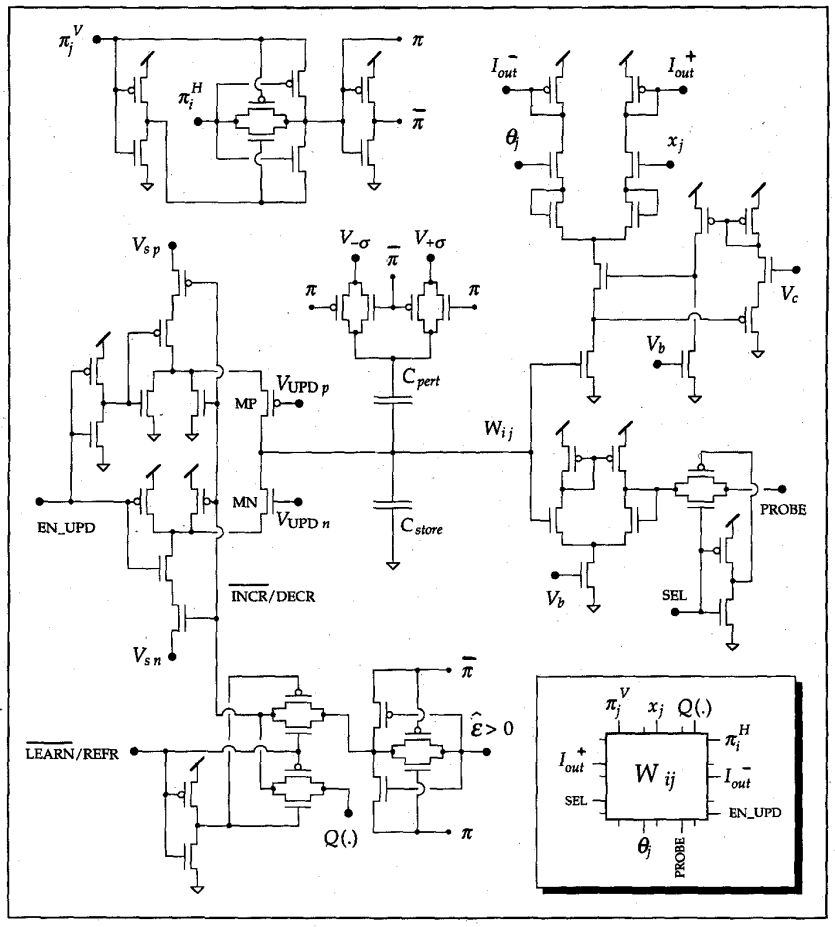



Fig. 9. Complete schematic of the synapse cell including learning and storage functions.

relating to error observations on the network, evaluating the network error under complementary perturbations, and combining the two results to construct an estimate of the differential perturbed error (5). Possible functional extensions for increased performance, which can easily be incorporated in a general-purpose implementation, include adaptive biasing techniques for dynamically optimizing the learning rate, and conditional tests on the range of unperturbed and perturbed error observations [or various combinations of all three observed values $\mathcal{E}(\mathbf{p})$, $\mathcal{E}(\mathbf{p}+\pi)$, and $\mathcal{E}(\mathbf{p}-\pi)$] to guard and remedy against potential instabilities in the updates.

In the present implementation, all global learning functions are performed off-chip, by means of analog and digital hardware interfacing with a general-purpose computer. The evaluation of the error functional (2) on the network is performed with discrete analog components, leaving some flexibility to experiment with different formulations of error functionals that otherwise would have been hardwired. A mean absolute difference ($\nu = 1$) norm is used for the metric distance, and the time-averaging of the error (3) is achieved by a fourth-order Butterworth low-pass filter. The cutoff frequency is adjusted to accommodate an AC ripple smaller than 0.1%, giving rise to a filter settling time extending 20 periods of the training signal. Since three observations of the error (2) on the network are needed to obtain the unperturbed and complementary perturbed error estimates, a single learning iteration spans at least 60 periods of the training signal.

In principle, the refresh of the parameters does not require higher-level global supervision at all, since the timing and multiplexing of local update and binary quantization functions can be coordinated by means of predefined cyclic control sequences, which can be generated on-chip driven by a periodic clock signal. Such allows the refresh operations to run autonomously in the background, transparent to the operation of the network. In the present realization, the binary quantization functions are not included on the chip circuitry and need to be performed externally. The technology for integrating *A/D/A* converters, implementing on-chip quantizers, has been developed and demonstrated [46], and the main focus in the experiment conducted here is to characterize the learning performance of the network chip, to which the supporting storage method is secondary. For simplicity of demonstration, the parameters $p_i$ are stored externally once the learning is completed, and thereafter refreshed sequentially by supplying values for the quantization bits $Q(p_i^{(k)})$ as defined by the

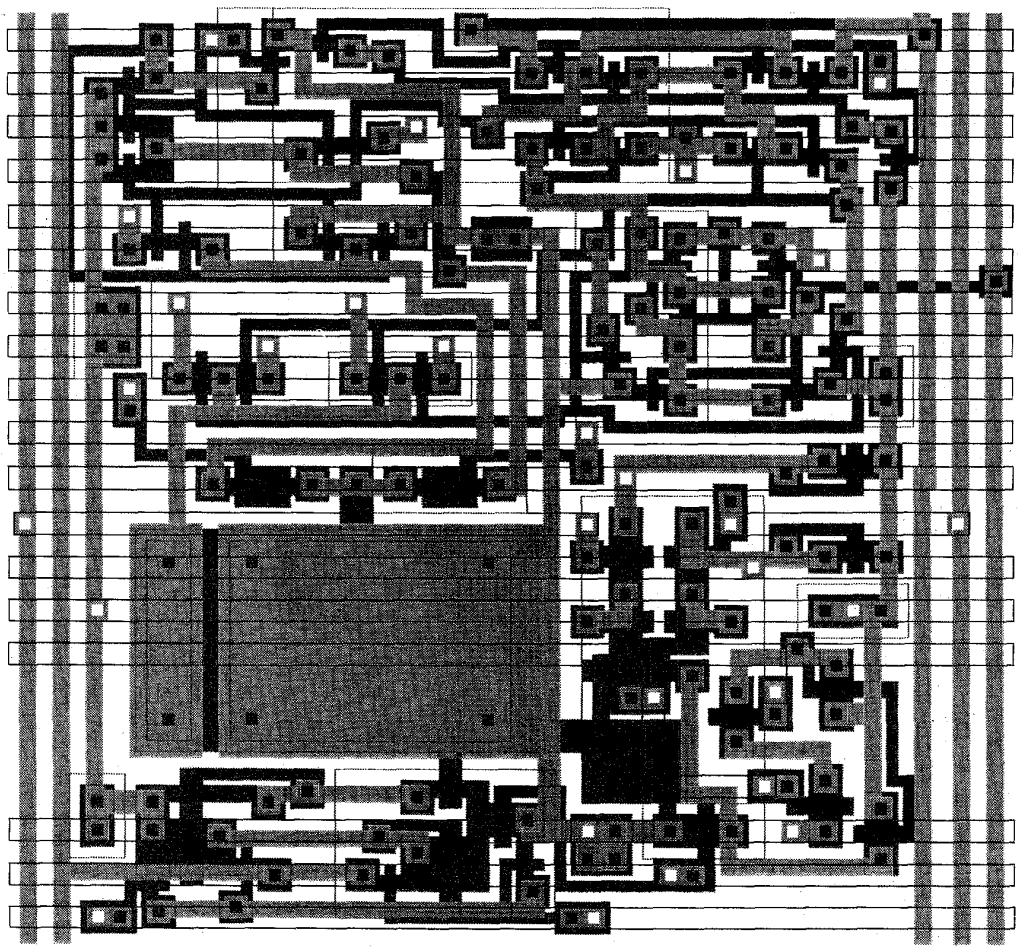


Fig. 10. Physical layout of the synapse cell with integrated learning and storage functions.

polarity of the observed deviation between internally probed $p_i^{(k)}$ and externally stored $p_i$ parameters values. External storage and the generation of binary quantization values are performed and controlled by the computer supervising the experiment. The simplified refresh scheme with external storage does not involve internal quantization feedback loops, and therefore allows error-free operation at slower refresh rates. The parameter refresh is performed in the background with a 100 ms cycle, whenever the learning process is disabled or interrupted.

## IV. Experimental Learning Results

As a proof of principle, the network is trained to generate outputs following a circular target trajectory, in absence of externally supplied inputs. The target output signals are defined by the quadrature-phase oscillator

$$\begin{cases} x_1^T(t) &= A\cos(2\pi ft) \\ x_2^T(t) &= A\sin(2\pi ft) \end{cases} \tag{9}$$

with amplitude $A = 0.8$ V and frequency $f = 1$ kHz. In principle, a recurrent network of two neurons suffices to generate quadrature-phase oscillations, and the extra neurons in the network serve to accommodate the particular amplitude and frequency requirements and assist in reducing the nonlinear distortion.

Training a recurrent neural network to exhibit prescribed oscillatory behavior by means of error descent in the parameter space is a harder problem than might be anticipated at first thought. The shape of the error surface in parameter space, defined by the time-averaged format of (2) or its equivalent as implemented by low-pass filtering of (3), contains local minima and abrupt discontinuities, which lead to unpredictable learning results when the network is initialized arbitrarily. This is because the asymptotic dynamics of recurrent neural networks depend strongly on the parameter values, especially in the neighborhood of the boundary regions between slightly damped and slightly amplified small-signal dynamics of the state variables, where infinitesimal small parameter changes may trigger drastic transitions between a fixed point attractor behavior, limit cycle oscillations, instability and multistability in the dynamics [47]. Incidentally, we found that randomly initialized learning sessions usually fail to generate oscillatory behavior at convergence, the network being trapped in a

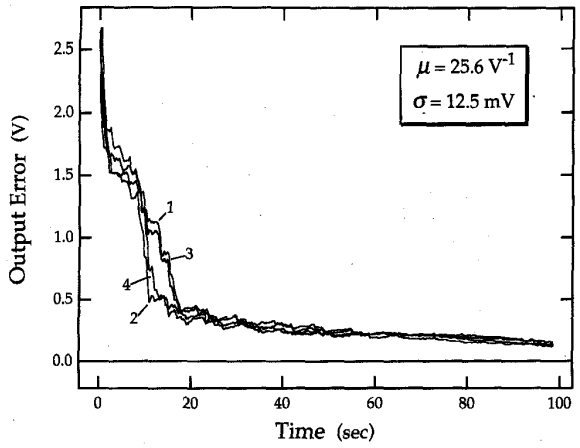
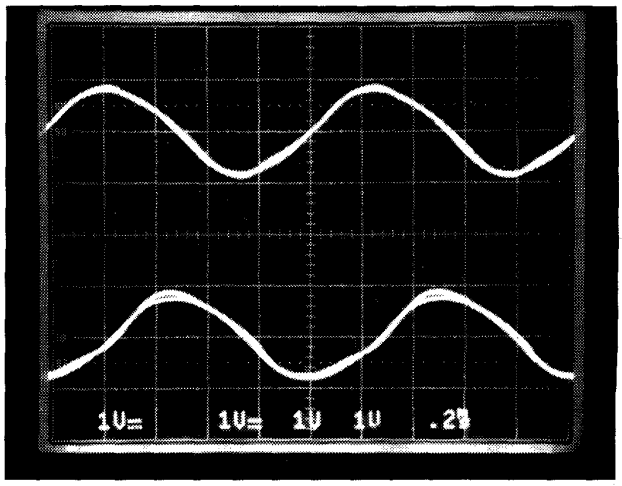
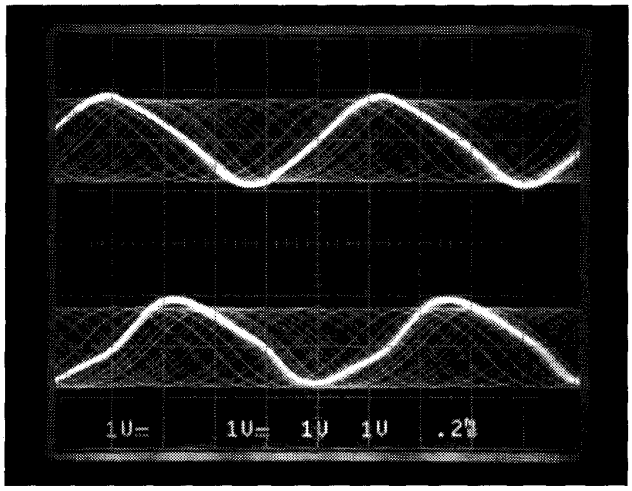



Fig. 11. Recorded evolution of the error during learning, for four different sessions on the network.



(a)



(b)

Fig. 12. Oscillograms of the target signals and network outputs with learning suspended, (a) under weak teacher forcing and (b) without teacher forcing. Top traces of each: $x_1(t)$ and $x_1^T(t)$. Bottom traces: $x_2(t)$ and $x_2^T(t)$.

local minimum defined by a strong point attractor. Even with strong teacher forcing, these local minima persist. The sharp parameter dependence of the error functional, fueling the convergence problems, can be mostly avoided by using shorter integration time intervals for averaging of the error (2), matching the characteristic time scale of the network dynamics and training signal. Such requires a modified scheme of coordinating the perturbations and error observations to allow practical on-line implementation, which could not be incorporated in the present experimental setup.

The convergence problems due to the ill-shaped structure of the error functional can be circumvented through a rational choice of the initial conditions. Clearly a particular choice of initial conditions for the parameter values may artificially convert a hard learning problem to a trivial one, and to warrant meaningful results we have derived initial conditions based on physical considerations generally valid for other trajectory learning tasks on recurrent neural networks as well. We obtained consistent and satisfactory results with the following initialization of network parameters: strong positive diagonal connection strengths $W_{ii} = 1$, zero off-diagonal terms $W_{ij} = 0$; $i \neq j$ and zero thresholds $\theta_i = 0$. The positive diagonal connections $W_{ii}$ are needed to free the neuron state variables from the point attractor at the origin, corresponding to the spontaneous decay term $-x_i$ in (1). This allows the network outputs to better follow the target signals under strong initial teacher forcing, for fast and robust learning. The zero initial values for the cross connections $W_{ij}$, $i \neq j$ are required to avoid any bias in the dynamics, due to coupling between neurons, during the initial phase of the learning. Gradual relaxation of the teacher forcing strength afterwards allows to establish the desired coupling strengths needed to maintain the neuron output waveforms closely near those of the target signals.

Fig. 11 shows recorded error sequences under training of the network with the target oscillator (9), for four different sessions of 1500 learning iterations each starting from the above initial conditions. The learning iterations span about 60 ms each, for a total of 100 s per session. The teacher forcing amplitude $\lambda$ is set initially to $V_\lambda = 3$ V, and thereafter decays logarithmically over one order of magnitude toward the end of the sessions. Fixed values of the learning rate and the perturbation amplitude are used throughout the sessions, with $\mu = 25.6\ V^{-1}$ and $\sigma = 12.5$ mV. All four sessions show a rapid initial decrease in the error under stimulus of the strong teacher forcing, and thereafter undergo a region of persistent flat error slowly tapering off toward convergence as the teacher forcing is gradually released. Notice that this flat region does not imply slow learning; instead the learning constantly removes error as additional error is adiabatically injected by the relaxation of the teacher forcing.

Near convergence, the bias in the network error due to the residual teacher forcing becomes small. Fig. 12 shows the network outputs and target signals at convergence, with the learning suspended and the parameter refresh activated, illustrating the minor effect of the residual teacher forcing

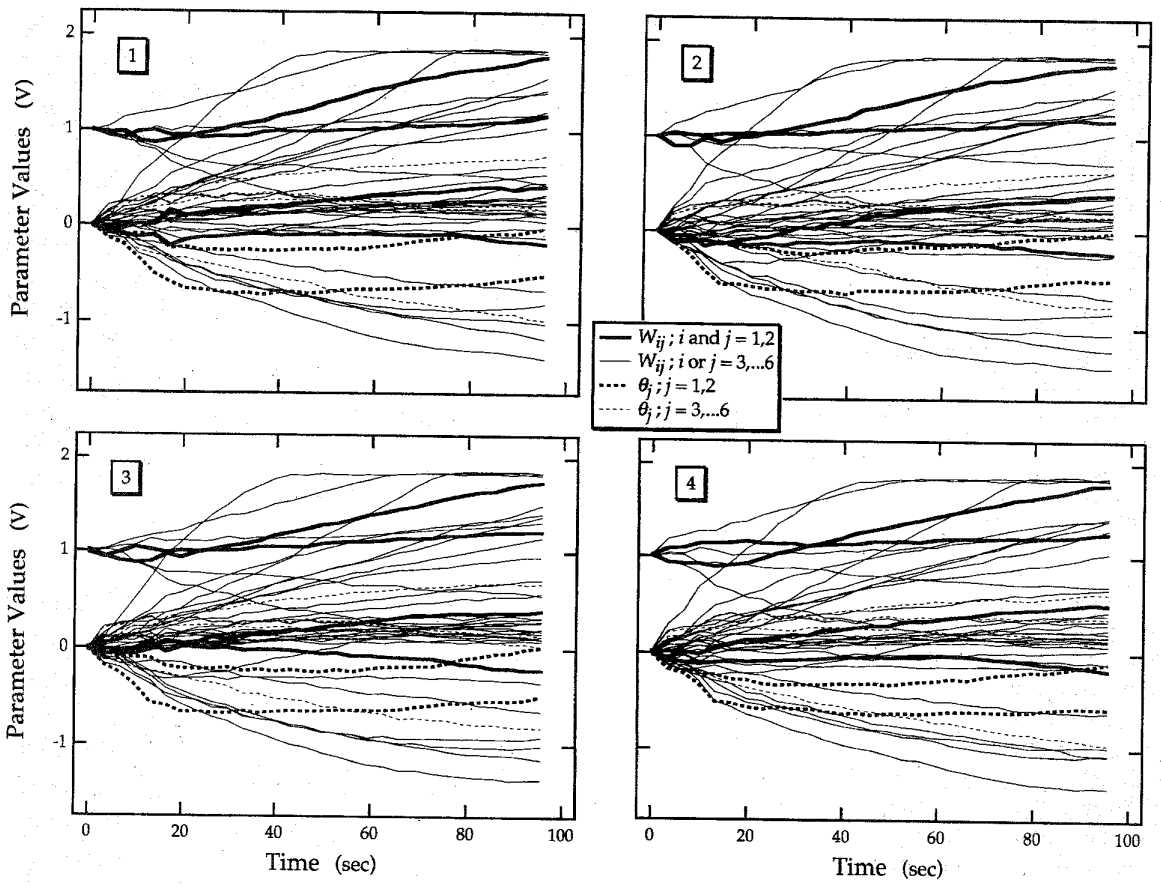



Fig. 13. Recorded evolution of the network parameters during learning, for four different sessions on the network.

signal on the network dynamics. The oscillogram of Fig. 12(a) is obtained under a weak teacher forcing signal, and that of Fig. 12(b) is obtained with the same network parameters but with teacher forcing disabled. In both cases the oscilloscope is triggered on the network output signals. Obviously, in absence of teacher forcing, the network does no longer run synchronously with the target signal. The discrepancy in frequency, amplitude, and shape between either of the free-running and forced oscillatory output waveforms and the target signal waveforms, however, is evidently small.

It would have been instructive to observe the internal dynamics of the network other than that of the two output neurons $x_1$ and $x_2$. Unfortunately, the output voltages of internal neurons $x_3$ through $x_6$ are not accessible off-chip in the present chip implementation, due to pin-out limitations. Instead, we recorded the evolution of the complete set of network parameters, which are accessible through the probing and multiplexing circuitry supporting the binary quantization, during each of the above four learning sessions. The recorded curves for the synapse and threshold parameters, sampled every 50 learning update cycles, are shown separately for all four sessions in Fig. 13. To ease interpretation, the parameter curves relating to any of the four internal neurons are visually distinguished from those pertaining exclusively to the two output neurons. Both families of curves clearly display significant changes in the parameter values from the initial settings during the learning sessions, which testifies that all neurons, including the internal neurons, are actively engaged in training the dynamics of the output neurons. The strong similarity between sets of parameter curves across different learning sessions, shown in the four separate plots of Fig. 13, further indicates that the involvement of the internal neurons during learning is a systematic rather than a purely diffusive process, whereby the complete spectrum of dynamics achievable by the fully configured network is explored to produce the desired output neuron waveforms.

Closer examination of the parameter values at convergence, given in Table I, reveals that all four sessions actually result into approximately the same network, with the neurons configured in a particular order. In principle, the outputs of the network are invariant to topological interchanging of internal neurons, with corresponding permutations of rows and columns in the parameter matrix. Therefore, the learning task contains multiple degenerate solutions, all of which should equally likely be obtained under learning from the unbiased initial values for the parameters. In practice, analog offsets and mismatches in the implementation of the network break the symmetry existing among the neurons, and introduce a bias in the network dynamics at the initial parameter settings, favoring a particular solution. The fact that the implemented system is

TABLE I
PARAMETER VALUES OBSERVED AT CONVERGENCE, FOR FOUR DIFFERENT SESSIONS ON THE NETWORK

Session 1

| $j$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $W1j$ | 1.18 | -0.21 | 0.61 | -0.68 | -1.17 | 1.06 |
| $W2j$ | 0.44 | 1.80 | 0.47 | -0.79 | 1.86 | -1.40 |
| $W3j$ | 0.16 | 0.29 | 1.18 | 0.15 | 0.26 | 0.06 |
| $W4j$ | 1.61 | 0.16 | -0.04 | 0.45 | -0.12 | 0.19 |
| $W5j$ | 0.68 | 0.33 | 1.46 | 1.83 | 1.84 | 0.30 |
| $W6j$ | 0.27 | 1.42 | 0.25 | 1.23 | -1.02 | 0.15 |
| $\theta j$ | -0.00 | -0.51 | -0.99 | 0.78 | 0.18 | 0.16 |

Session 2

| $j$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $W1j$ | 1.17 | -0.24 | 0.48 | -0.69 | -1.24 | 1.05 |
| $W2j$ | 0.40 | 1.80 | 0.34 | -0.84 | 1.86 | -1.45 |
| $W3j$ | -0.05 | 0.37 | 1.26 | 0.13 | 0.17 | 0.04 |
| $W4j$ | 1.40 | 0.11 | -0.26 | 0.73 | 0.00 | 0.06 |
| $W5j$ | 0.69 | 0.19 | 1.56 | 1.82 | 1.85 | 0.40 |
| $W6j$ | 0.21 | 1.43 | 0.28 | 1.31 | -1.08 | 0.24 |
| $\theta j$ | 0.01 | -0.47 | -0.78 | 0.66 | 0.16 | 0.09 |

Session 3

| $j$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $W1j$ | 1.21 | -0.24 | 0.62 | -0.66 | -1.16 | 0.99 |
| $W2j$ | 0.39 | 1.76 | 0.45 | -0.90 | 1.86 | -1.38 |
| $W3j$ | 0.14 | 0.29 | 1.32 | 0.09 | 0.19 | 0.14 |
| $W4j$ | 1.54 | 0.31 | 0.11 | 0.54 | -0.37 | 0.08 |
| $W5j$ | 0.73 | 0.26 | 1.43 | 1.82 | 1.83 | 0.35 |
| $W6j$ | 0.21 | 1.41 | 0.29 | 1.19 | -1.02 | 0.20 |
| $\theta j$ | 0.02 | -0.46 | -0.81 | 0.68 | 0.12 | 0.03 |

Session 4

| $j$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $W1j$ | 1.25 | -0.20 | 0.54 | -0.61 | -1.06 | 1.06 |
| $W2j$ | 0.49 | 1.79 | 0.47 | -0.98 | 1.86 | -1.42 |
| $W3j$ | 0.05 | 0.21 | 1.24 | 0.32 | 0.15 | 0.17 |
| $W4j$ | 1.46 | 0.25 | -0.16 | 0.70 | -0.10 | 0.08 |
| $W5j$ | 0.67 | 0.32 | 1.43 | 1.83 | 1.85 | 0.41 |
| $W6j$ | 0.20 | 1.44 | 0.21 | 1.31 | -1.07 | 0.13 |
| $\theta j$ | -0.08 | -0.57 | -0.96 | 0.64 | 0.35 | 0.06 |

able to consistently regenerate the same network solution from different learning sessions, following more or less the same trajectory in parameter space despite different instances for the random perturbation values, illustrates the global deterministic error descent property of the stochastic scheme, resembling the characteristics of gradient descent on a macroscopic scale. More importantly, learning sessions on different fabricated instances of the same chip design all consistently converged to network solutions of comparable quality, despite fairly large differences in the parameter values obtained at convergence due to the randomness of the implementation errors. The robustness of the implemented learning system to random offsets and nonlinearities induced at fabrication is a major advantage of the model-independent approach followed here, using statistical techniques based on direct observations of the error to obtain the gradient information needed for learning.

## V. CONCLUSION

We implemented and characterized an analog recurrent neural network chip incorporating local provisions for on-line learning of continuous-time dynamics. By employing a simple stochastic perturbative algorithm for error descent learning, we avoided the usual sensitivity of learning performance to model errors in the network implementation, and were able to obtain a modular and scalable VLSI architecture for the learning functions, locally integrated with the array of network cells in a 2-D. Through simple extensions on the circuit structure of the learning cells, the local provisions for updating the parameters during learning are furthermore used to refresh the volatile parameter storage in the background, allowing for long-term retention of the network information. The implemented network and embedded learning and storage systems can be directly expanded, without changing the internal structure of the cells, to accommodate applications of recurrent dynamical networks requiring larger dimensionality. Table II summarizes system level features of the tested chips. The power dissipation measures 1.2 mW under the conditions of the experiments, both in learning and refresh mode. The chip micrograph is shown in Fig. 14.

TABLE II
CHIP FEATURES

| | |
|---|---|
| Technology | 2 μm p-well double-poly CMOS |
| Supply voltage | +5 V |
| Power dissipation | |
| total | 1.2 mW |
| Area | |
| active die | 2.2 mm X 2.2 mm |
| synapse cell | 170 μm X 182 μm |
| Transistor count | |
| total | 3885 |
| synapse cell | 56 |
| Pin count | 40 |

Experimental tests on the chip, trained with a circular dynamic trajectory, confirmed the robustness of learning per-

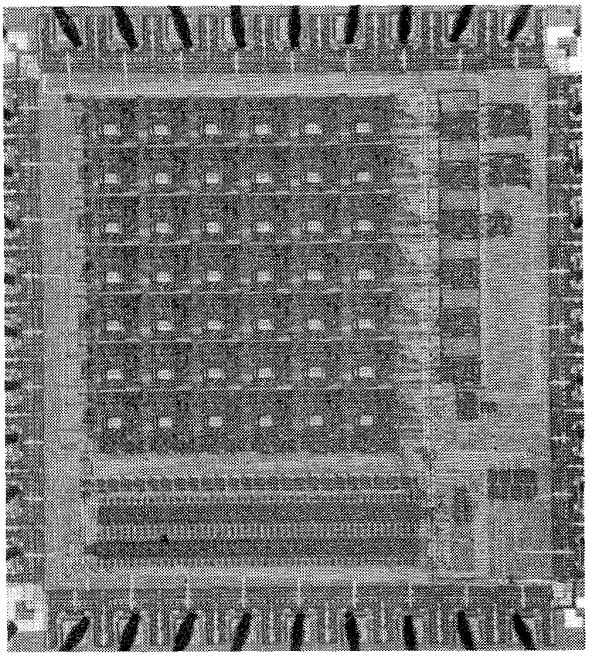


Fig. 14. Chip micrograph. Center: network array of synapse and neuron cells. Bottom: multichannel random bit generator.

formance in the presence of analog offsets in the implementation due to imprecision in the fabrication. While the time-averaged formulation of the functional used for error-descent learning is known to cause convergence problems due to local minima and discontinuities in the error surface, consistently successful results were obtained using strong initial teacher forcing and unbiased initial settings for the parameters. The network repeatedly succeeded to learn 1 kHz quadrature-phase oscillatory waveforms in 1500 update cycles, spanning 100 s total. A present limitation of the implemented learning model is the requirement of periodicity on the input and target signals during the learning process, which is needed to allow a repetitive and consistent evaluation of the network error for the parameter updates.

## ACKNOWLEDGMENT

The author wishes to thank V. Pedroni, A. Yariv, and others for helpful discussions and/or critical reading of the manuscript. Fabrication of the CMOS chip was provided through the DARPA/NSF MOSIS service. Tanner Research assisted with microscope photographs of the chip.

## REFERENCES

[1] J. Alspector, B. Gupta, and R. B. Allen, "Performance of a stochastic learning microchip," in *Advances in Neural Information Processing Systems*, vol. 1. San Mateo, CA: Morgan Kaufman, 1989, pp. 748–760.

[2] E. Vittoz and X. Arreguit, "CMOS integration of Herault–Jutten cells for separation of sources," in *Analog VLSI Implementation of Neural Systems*. Norwell, MA: Kluwer, 1989, pp. 57–83.

[3] S. Bibyk and M. Ismail, "Issues in analog VLSI and MOS techniques for neural computing," in *Analog VLSI Implementation of Neural Systems*. Norwell, MA: Kluwer, 1989, pp. 103–133.

[4] B. Hochet, V. Peiris, S. Abdot, and M. J. Declercq, "Implementation of a learning Kohonen neuron based on a new multilevel storage technique," *IEEE J. Solid-State Circuits*, vol. 26, no. 3, pp. 262–267, 1991.

[5] M. H. Cohen and A. G. Andreou, "Current-mode subthreshold MOS implementation of the Jutten–Herault autoadaptive network," *IEEE J. Solid-State Circuits*, vol. 27, no. 5, pp. 714–727, 1992.

[6] G. Cauwenberghs, C. F. Neugebauer, and A. Yariv, "Analysis and verification of an analog VLSI outer-product incremental learning system," *IEEE Trans. Neural Networks*, vol. 3, no. 3, pp. 488–497, 1992.

[7] R. G. Benson and D. A. Kerns, "UV-activated conductances allow for multiple time scale learning," *IEEE Trans. Neural Networks*, vol. 4, no. 3, pp. 434–440, 1993.

[8] J. Donald and L. Akers, "An adaptive neural processor node," *IEEE Trans. Neural Networks*, vol. 4, no. 3, pp. 413–426, 1993.

[9] B. Linares-Barranco, E. Sanchez-Sinencio, A. Rodriguez-Vazquez, and J. L. Huertas, "A CMOS analog adaptive BAM with on-chip learning and weight refreshing," *IEEE Trans. Neural Networks*, vol. 4, no. 3, pp. 445–455, 1993.

[10] H. P. Graf and L. D. Jackel, "Analog electronic neural network circuits," *IEEE Circuits Devices Mag.*, vol. 5, no. 4, pp. 44–49, 1989.

[11] C. A. Mead, *Analog VLSI and Neural Systems*. Reading, MA: Addison-Wesley, 1989.

[12] A. G. Andreou, K. A. Boahen, P. O. Pouliquen, A. Pavasovic, R. E. Jenkins, and K. Strohbehn, "Current-mode subthreshold MOS circuits for analog VLSI neural systems," *IEEE Trans. Neural Networks*, vol. 2, no. 2, pp. 205–213, 1991.

[13] D. E. Rumelhart and J. L. McClelland, Eds., *Parallel Distributed Processing, Explorations in the Microstructure of Cognition*, vol. 1. Cambridge, MA: MIT Press, 1986.

[14] M. J. S. Smith, "An analog integrated neural network capable of learning the Feigenbaum logistic map," *IEEE Trans Circuits Syst.*, vol. 37, no. 6, pp. 841–844, 1990.

[15] R. C. Frye, E. A. Rietman, and C. C. Wong, "Backpropagation learning and nonidealities in analog neural network hardware," *IEEE Trans. Neural Networks*, vol. 2, no. 1, pp. 110–117, 1991.

[16] M. Jabri and B. Flower, "Weight perturbation: An optimal architecture and learning technique for analog VLSI feedforward and recurrent multilayered networks," *IEEE Trans. Neural Networks*, vol. 3, no. 1, pp. 154–157, 1992.

[17] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Networks*, vol. 1, no. 1, pp. 4–27, 1990.

[18] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Computa.*, vol. 1, no. 2, pp. 270–280, 1989.

[19] B. A. Pearlmutter, "Learning state-space trajectories in recurrent neural networks," *Neural Computa.*, vol. 1, no. 2, pp. 263–269, 1989.

[20] N. B. Toomarian and J. Barhen, "Learning a trajectory using adjoint functions and teacher forcing," *Neural Networks*, vol. 5, no. 3, pp. 473–484, 1992.

[21] J. Schmidhuber, "A fixed size storage $O(n^3)$ time complexity learning algorithm for fully recurrent continually running networks," *Neural Computa.*, vol. 4, no. 2, pp. 243–248, 1992.

[22] G.-Z. Sun, H.-H. Chen, and Y.-C. Lee, "Green's function method for fast on-line learning algorithm of recurrent neural networks," in *Advances in Neural Information Processing Systems*, vol. 4. San Mateo, CA: Morgan Kaufman, 1992, pp. 333–340.

[23] P. Baldi, "Gradient descent learning algorithms: A general dynamical systems perspective," *IEEE Trans. Neural Networks*, vol. 6, no. 1, pp. 182–195, Jan. 1995.

[24] P. Mueller, J. Van Der Spiegel, D. Blackman, T. Chiu, T. Clare, C. Dunham, T. P. Hsieh, and M. Loinez, "Design and fabrication of VLSI components for a general purpose analog neural computer," in *Analog VLSI Implementation of Neural Systems*. Norwell, MA: Kluwer, 1989, pp. 135–169.

[25] G. Cauwenberghs, "A fast stochastic error-descent algorithm for supervised learning and optimization," in *Advances in Neural Information Processing Systems*, vol. 5. San Mateo, CA: Morgan Kaufman, 1993, pp. 244–251.

[26] H. Robins and S. Monro, "A stochastic approximation method," *Annals Math Statist.*, vol. 22, pp. 400–407, 1951.

[27] H. J. Kushner and D. S. Clark, *Stochastic Approximation Methods for Cconstrained and Unconstrained Systems*. New York: Springer-Verlag, 1978.

[28] M. A. Styblinski and T.-S. Tang, "Experiments in nonconvex optimization: Stochastic approximation with function smoothing and simulated annealing," *Neural Networks*, vol. 3, no. 4, pp. 467–483, 1990.

[29] J. C. Spall, "Multivariate stochastic approximation using a simultaneous

perturbation gradient approximation," *IEEE Trans. Automat. Contr.*, vol. 37, no. 3, pp. 332–341, 1992.
[30] A. Dembo and T. Kailath, "Model-free distributed learning," *IEEE Trans. Neural Networks*, vol. 1, no. 1, pp. 58–70, 1990.
[31] D. Kirk, D. Kerns, K. Fleischer, and A. Barr, "Analog VLSI implementation of gradient descent," in *Advances in Neural Information Processing Systems*, vol. 5. San Mateo, CA: Morgan Kaufman, 1993, pp. 789–796.
[32] J. Alspector, R. Meir, B. Yuhas, and A. Jayakumar, "A parallel gradient descent method for learning in analog VLSI neural networks," in *Advances in Neural Information Processing Systems*, vol. 5. San Mateo, CA: Morgan Kaufman, 1993, pp. 836–844.
[33] B. Flower and M. Jabri, "Summed weight neuron perturbation: An $\mathcal{O}(n)$ improvement over weight perturbation," in *Advances in Neural Information Processing Systems*, vol. 5. San Mateo, CA: Morgan Kaufman, 1993, pp. 212–219.
[34] G. Cauwenberghs, C. F. Neugebauer, A. Agranat, and A. Yariv, "Large scale optoelectronic integration of asynchronous analog neural networks," in *Proc. Dig. Int. Neural Network Conf. (INNC-90 Paris)*, vol. 2, pp. 551–554, 1990.
[35] S. Satyanarayana, Y. P. Tsividis, and H. P. Graf, "A reconfigurable VLSI neural network," *IEEE J. Solid-State Circuits*, vol. 27, no 1, pp. 67–81, 1992.
[36] J. W. Fattaruso, S. Kiriaki, G. Warwar, and M. de Wit, "Self-calibration techniques for a second-order multibit sigma-delta modulator," in *ISSCC Tech. Dig.*, vol. 36, pp. 228–229, 1993.
[37] E. Säckinger and W. Guggenbühl, "A high-swing, high-impedance MOS cascode circuit," *IEEE J. Solid-State Circuits*, vol. 25, no. 1, pp. 289–298, 1990.
[38] Z. Czarnul, "Modification of the Banu–Tsividis continuous-time integrator structure," *IEEE Trans. Circuits Syst.*, vol. CAS-33, no. 7, pp. 714–716, 1986.
[39] L. Watts, D. A. Kerns, and R. F. Lyon, "Improved implementation of the silicon cochlea," *IEEE J. Solid-State Circuits*, vol. 27, no. 5, pp. 692–700, 1992.
[40] S. Satyanarayana, Y. P. Tsividis, and H. P. Graf, "Analogue neural networks with distributed neurons," *Electron. Lett.*, vol. 25, no. 5, pp. 302–303, 1989.
[41] B. Sheu and C. Hu, "Switched-induced error voltage on a switched capacitor," *IEEE J. Solid-State Circuits*, vol. SSC-19, no. 4, pp. 519–525, 1984.
[42] E. Vittoz and J. Fellrath, "CMOS analog integrated circuits based on weak inversion operation," *IEEE J. Solid-State Circuits*, vol. SSC-12, no. 3, pp. 224–231, 1977.
[43] S. W. Golomb, *Shift Register Sequences*. San Francisco, CA: Holden-Day, 1967.
[44] J. Alspector, J. W. Gannett, S. Haber, M. B. Parker, and R. Chu, "A VLSI-efficient technique for generating multiple uncorrelated noise sources and its application to stochastic neural networks," *IEEE Trans. Circuits Syst.*, vol. 38, no. 1, pp. 109–123, 1991.
[45] G. Cauwenberghs and A. Yariv, "Fault-tolerant dynamic multilevel storage in analog VLSI," *IEEE Trans. Circuits Syst. II* vol. 41, no. 12, pp. 827–829, 1994.
[46] ——, "Method and apparatus for monotonic algorithmic digital-to-analog and analog-to-digital conversion," U.S. Patent 5 258 759, 1993.
[47] P. Baldi, "Learning in dynamical systems: Gradient descent, random descent, and modular approaches," California Institute Technol., JPL Tech. Rep., 1992.

**Gert Cauwenberghs** (S'89–M'92) received the engineer's degree in applied physics from the Vrije Universiteit Brussel, Belgium, in 1988, and the M.S. and Ph.D. degrees in electrical engineering from the California Institute of Technology, Pasadena, in 1989 and 1994, respectively.

He joined Johns Hopkins University, Baltimore, MD, in 1994 as an Assistant Professor in Electrical and Computer Engineering. His research includes VLSI systems and algorithms for parallel signal processing and adaptive neural computation.