An Experimentation Platform for On-Chip Integration of Analog Neural Networks: A Pathway to Trusted and Robust Analog/RF ICs

Dzmitry Maliuk, *Member, IEEE*, and Yiorgos Makris, *Senior Member, IEEE*

***Abstract*— We discuss the design of an experimentation platform intended for prototyping low-cost analog neural networks for on-chip integration with analog/RF circuits. The objective of such integration is to support various tasks, such as self-test, self-tuning, and trust/aging monitoring, which require classification of analog measurements obtained from on-chip sensors. Particular emphasis is given to cost-efficient implementation reflected in: 1) low energy and area budgets of circuits dedicated to neural networks; 2) robust learning in presence of analog inaccuracies; and 3) long-term retention of learned functionality. Our chip consists of a reconfigurable array of synapses and neurons operating below threshold and featuring sub-μW power consumption. The synapse circuits employ dual-mode weight storage: 1) a dynamic mode, for fast bidirectional weight updates during training and 2) a nonvolatile mode, for permanent storage of learned functionality. We discuss a robust learning strategy, and we evaluate the system performance on several benchmark problems, such as the XOR2-6 and two-spirals classification tasks.**

***Index Terms*— Analog neural network, chip-in-the-loop training, multilayer perceptron, nonvolatile weight storage, ontogenic neural network, reconfigurable array, translinear circuits.**

I. Introduction

In RECENT years, machine learning-based solutions have found use in a number of applications related to examining robustness and trustworthiness of analog/RF integrated circuits (ICs). The general architecture of these solutions is shown in the top portion of Fig. 1 and consists of circuitry which can generate predefined on-chip stimuli and simple sensors which can obtain parametric measurements from an analog/RF IC in response to these stimuli. These measurements can then be processed by an on-chip classifier, to assess various operational aspects of the IC. For example, a trained classifier can be periodically used to assess, on the basis of simple sensor measurements, whether the performances of an analog/RF IC continue to meet the design specifications, thereby providing a *built-in self-test* (BIST) capability [1], which is particularly desirable in circuits deployed in safety-critical applications. Similarly, a trained classifier can periodically use simple sensor measurements to choose appropriate knob positions in tunable analog/RF ICs, to select the optimal setting for a given set of operating conditions, thereby increasing robustness and optimizing power consumption through *self-tuning* [2]. Another application of this general architecture is in *detecting hardware Trojans* [3], a contemporary concern brought about by the globalization of the IC supply chain. Specifically, a trained on-chip classifier can be distinguished on the basis of simple parametric sensor measurements (also known as side-channel fingerprints), between hardware Trojan-free and hardware Trojan-infested ICs, thereby enhancing trustworthiness. Yet another trust-related application of on-chip classifiers can be found in using sensor measurements to distinguish between new and aged chips, thereby helping in combating the growing concern of *detecting recycled and counterfeit ICs* [4].

All these applications share a common characteristic, i.e., the use of a trained machine learning entity to make a crucial decision regarding robustness or trustworthiness of

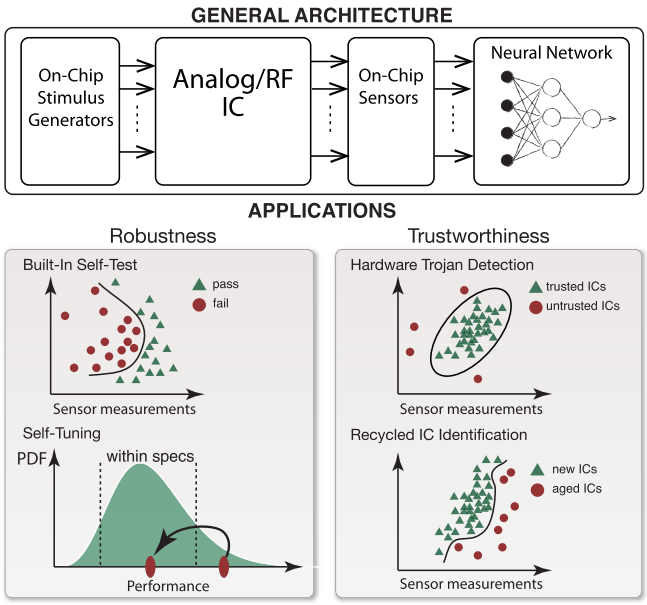

Fig. 1. Applications of on-chip classifiers in trusted and robust analog/RF ICs.

Manuscript received November 27, 2013; accepted August 27, 2014. Date of publication September 17, 2014; date of current version July 15, 2015.

D. Maliuk was with the Department of Electrical Engineering, Yale University, New Haven, CT 06520 USA. He is now with Quantlab Financial, Houston, TX 77006 USA (e-mail: dmaliuk@quantlab.com).

Y. Makris is with the Department of Electrical Engineering, University of Texas at Dallas, Richardson, TX 75080 USA (e-mail: yiorgos.makris@utdallas.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2014.2354406

an analog/RF IC. While it has been shown that even relatively simple learning models posses the power to extract the decision information from low-cost measurements [5], [6], prior research in this application domain has only explored their software implementation running on an external computer [7] or a built-in DSP block [8]. However, such processing resources may be prohibitively expensive and are not always available for performing such classification tasks on a stand-alone analog/RF IC. Accordingly, in this paper we explore the potential of developing a low-cost hardware neural network implementation which can be integrated onto an IC in order to provide the aforementioned capabilities. To this end, we describe an experimentation platform which we developed in order to prototype appropriate neural topologies and circuit parameters for solving classification problems such as the ones described above.

While numerous neural platforms employing various implementation technologies and addressing a range of requirements dictated primarily by large-scale biological systems exist in the literature, the specific set of requirements imposed by our target applications call for a custom solution. In particular, the resulting on-chip integrated neural network must consume only a small fraction of available IC resources, especially with regards to area and power. For battery powered ICs, it is not uncommon that the power budget of such circuits lies in the micro- or even nano-Watt range. To this end, it is imperative that all processing is performed in the analog domain, since analog-to-digital conversion of the sensor measurements and processing in the digital domain is bound to be far more demanding in area and power. Additionally, the neural network must remember its learned regression function or classification boundary throughout the lifetime of the analog/RF IC. In a typical scenario, weight values are trained once at a vendor's facility and stored permanently using nonvolatile memory, such that the deployed ICs do not require any off-chip support to examine their robustness or trustworthiness. Furthermore, training time is another important resource, which calls for a rapidly programmable weight memory and a good training strategy capable of overcoming implementation nonidealities such as mismatch and process variation. Lastly, although it is necessary for the platform to be programmable with respect to the supported learning models, their topologies, circuit parameters, power consumption, etc., scalability is less important than in other state-of-the-art platforms. After all, if the resources of a single platform are not sufficient to address a problem, it is very unlikely that such a trained network would be a cost-efficient solution for the targeted applications.

The key contribution of this paper is the development of a custom neural platform targeting the specific design requirements outlined above. The design represents a major revision of an earlier version [9] with improved circuit- and system-level characteristics. The architecture supports two popular learning models, namely the multilayer perceptron (MLP) and the ontogenic neural network (ONN). Sacrificing the generality and scalability of spiking neural networks, we resort to a fully analog design operating in the space of sensor voltages and providing low overhead in terms of power consumption and transistor count. Equally important, our design allows direct connection of voltage-encoded sensor outputs, thereby eliminating the need for costly analog-to-digital or voltage-to-spiking-frequency converters. The basic computational primitives, such as multiplication and nonlinear activation functions, are realized by translinear circuits biased in the subthreshold region. A hybrid approach is used for weight storage: for fast bidirectional updates during training, weights are stored on dynamic capacitors; after convergence, they are copied onto floating gate transistors (FGTs) for permanent storage. In training, we employ a chip-in-the-loop paradigm for robustness against circuit nonlinearities and for modest use of extra resources dedicated to learning.

The remainder of this paper is structured as follows. In Section II we outline the current state-of-the-art in hardware implementation of neural networks. Section III introduces the MLP and ONN learning models. The overall chip architecture is described in Section IV. In Section V, we explain the dual-mode weight storage mechanism. Circuit implementation of the core building blocks—synapses and neurons—is given in Section VI. In Section VII, we provide the description of peripheral blocks that support characterization, training and operation of the neural network. Section VIII discusses training strategies for the MLP and ONN models. Measurement results and performance evaluation of the hardware learning models on two benchmark problems are given in Section IX. Finally, Section X concludes this paper.

II. Prior Art in Hardware Neural Platforms

Circuit implementation of neural networks is an active research area with several neural platforms successfully deployed and tested on real world problems [10]. One direction focuses on building neuromorphic systems of spiking neurons for hardware emulation of large-scale biological neural networks. These systems adopt various implementation technologies, offer great configurability and scalability (up to 1M neurons [11] per system), consume low power and implement complex neuron dynamics [12]. The SpiNNaker platform is an example of a fully digital implementation supporting a wide range of neuron models (MLP, IZH, LIF, etc.), providing high scalability (up to 1000 neurons per core), and offering low-power consumption of 12 to 45 nJ/ms per neuron [13]. Another example leverages digital technology to provide an accelerated simulation platform with up to 512 neurons and 100 000 synapses per core [14]. Mixed-signal implementations aim at further reducing transistor count and improving power efficiency, as evidenced by the Neurogrid project (941 pJ per synaptic activation). Stability considerations of large scale neural systems have also been investigated in [15].

A second direction focuses on direct implementation of neural networks represented by abstract mathematical models, such as feed-forward networks of multiplying synapses and summing neurons with sigmoid-like activation function. One of the first neural platforms for prototyping feed-forward topologies was Intel's ETANN chip [16]. The weight values were found off-line using a hardware emulation model and stored on-chip using floating gate (FG) memory. The design in [17] uses synapses with hybrid dynamic and nonvolatile

weight storage. Most of the training is completed on-chip utilizing dynamic storage and dedicated learning circuitry tailored for a chain perturbation algorithm. The authors in [18] address an ultralow power implementation of a neural classifier, which is composed of a vector-matrix multiplication (VMM) and a winner-take-all (WTA) circuit. Weight values are determined in an off-line training procedure and programmed onto the analog floating gate memory for permanent storage. The advantages of off-chip training, however, vanish rapidly for circuits which have small geometries and which are biased in the deep subthreshold region.

The variety of existing solutions suggests that design choices are largely dictated by application requirements. The spiking network techniques of the first direction, for example, are not directly applicable to our problem due to their focus on large scale systems (while our applications require very small footprint), complex neuron dynamics (which is unnecessary for learning the relatively simple classification boundaries of our applications), and digital or mixed-signal technology (while in our applications crossing the analog/digital domain is prohibitively expensive). Neural networks along the second direction provide a better alternative for our requirements by emphasizing a fully analog implementation (which offers energy efficiency of up to 1000× over the digital domain [19]), simple computational models (i.e., a multiplier for a synapse and a sigmoid function for a neuron), and compact nonvolatile weight storage solution (i.e., analog FG memory). Accordingly, we only compare the performance characteristics of our platform to implementations from this second group.

III. Mathematical Models

The first model supported by our platform is the widely used MLP, which is capable of learning complex nonlinear classification problems due to the presence of hidden layers in its topology [20]. The block diagram of an MLP is illustrated in Fig. 2(a). The network is feed-forward; it does not contain feedback loops and each layer receives connections only from inputs or previous layers. The first layer (also known as hidden) has M neurons which receive the inputs $X_1, X_2, \ldots, X_P$ and a constant $X_0 = 1$. The second layer (also known as output) contains a single neuron (for binary classification) which receives the outputs $Y_1^H, Y_2^H, \ldots, Y_M^H$ of the first layer and produces the network output Y^O. The strength of connections is controlled by synapses which act as multipliers of input signals and their local weight values. The sum of synaptic products is passed through a nonlinear activation function of a neuron. The number of inputs and output neurons is usually defined by the classification problem itself, while the number of hidden neurons reflects the learning capacity of the model.

The second model supported by our platform is the ONN. Unlike MLP, the ONN learns the boundary by both adjusting the weights and expanding its topology [7]. It has a similar structure, including a layer of inputs, several hidden neurons and an output neuron [Fig. 2(b)]. However, a single neuron receives connections from both the inputs and the outputs of preceding neurons. A decision boundary is constructed by

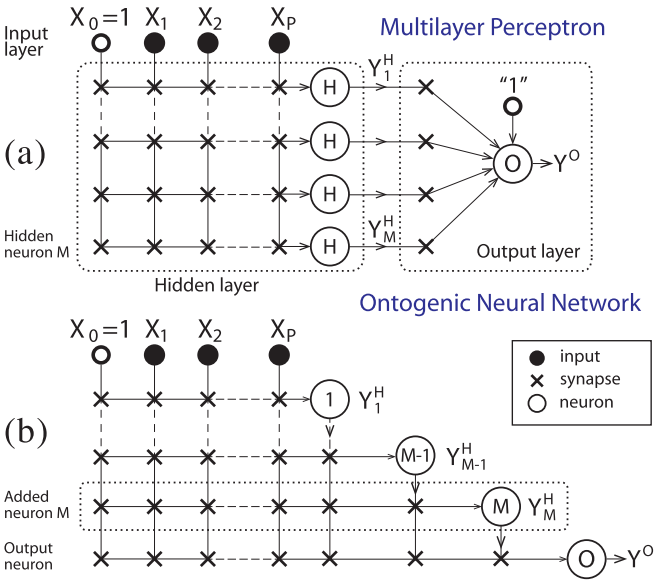

Fig. 2. Two learning models supported by the neural platform. (a) Multilayer perceptron. (b) Ontogenic neural network. The synapse acts as an ideal multiplier of an input signal and a local weight value. The neuron performs a tanh-like activation function on the sum of postsynaptic signals.

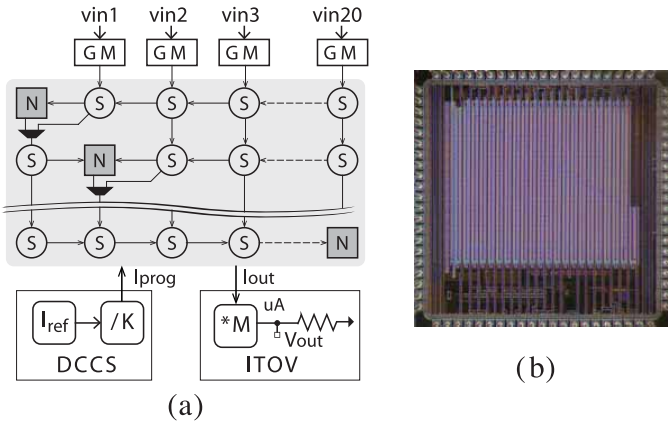

Fig. 3. (a) System architecture of the neural network platform. The reconfigurable array of synapses (S) and neurons (N) is shown in the shaded box. (b) Die photograph of the chip implemented in 0.35-μm complimentary metal–oxide–semiconductor and measuring 3×3 mm^2.

successively adding hidden neurons: each hidden neuron augments the feature space of the original inputs with the intention of making the derived space linearly separable. Trained by cascade-correlation (CC), the ONN avoids the moving target problem inherent in back propagation [21], thereby resulting in more compact network sizes for a given classification problem. The experiments in Section IX-D confirm this conjecture. In the remainder of this paper, the topologies of both neural models have an X–Y–Z designation, where X, Y, and Z are the number of inputs, hidden neurons, and output neurons, respectively.

IV. Chip Architecture

The block diagram of the neural platform is shown in Fig. 3. A 30×20 array of synapses (S) and neurons (N) is arranged so that the neurons are aligned along the main

diagonal of the upper matrix and along the right edge for the bottom part. Each row of synapses is locally connected to a corresponding neuron (N), forming a single unit. Global connectivity is programmable by means of multiplexors inserted between rows. The basic neural network operations, such as multiplication and nonlinear activation functions, are implemented via the translinear principle [22]. The signals and weights are represented by balanced differential currents for increased robustness and four-quadrant multiplication. As a result, a single weight value requires two current sources for differential current storage. For this purpose, we designed a current storage cell (CSC) featuring two modes of weight storage: dynamic, for rapid bidirectional programming, and nonvolatile, for long-term storage of learned weights. The dynamic mode is engaged during training, which requires thousands of weight update operations to be completed in a short period of time. Upon completion of training, the learned weights are stored permanently using FGTs.

The peripheral circuits provide support for fast programming and interfacing with the external world. The differential transconductors GM convert voltage-encoded input signals into balanced differential currents, as required by the core. The digitally controlled current source (DCCS) generates target currents from an on-chip reference for dynamic programming of the CSCs. Finally, the current-to-voltage converter (ITOV) facilitates the reading of internal currents by converting them to voltages that can be sampled using an external analog to digital converter. Each of these blocks requires characterization prior to their first use, to achieve a desired accuracy.

In subsequent sections, we provide a detailed description of neural circuit operation by using the following signal notation. The positive and negative components of a balanced differential current are denoted by I^+ and I^-, respectively, with an appropriate subscript. The differential (D) and common-mode (CM) currents are defined as

$$I^D = I^+ - I^- \quad \text{and} \quad I^{\text{CM}} = I^+ + I^-. \tag{1}$$

We will also use a notion of a normalized current (X) defined by

$$I^X = \frac{I^+ - I^-}{I^+ + I^-} = \frac{I^D}{I^{\text{CM}}} \tag{2}$$

and spanning the range from -1 to 1. Also, unless stated otherwise, we consider the default range for operating currents to be limited by 10 nA, so that the CM signal of a balanced current is 10 nA.

V. Weight Storage Mechanism

The principle of the CSC, which is the building block of the weight memory, is shown in Fig. 4. At the core of the circuit is a dual-gate FGT P1 (in the shaded box), whose drain current $I_w^{\pm}$ is the entity being stored. The drain current is modulated by the voltage on the FG (V_{fg}), which is itself determined by the FG node charge Q_{fg} and the control gate voltages V_{g1} and V_{g2} as

$$V_{\text{fg}} = \frac{Q_{\text{fg}} + C_{g1}V_{g1} + C_{g2}V_{g2} + C_{\text{tun}}V_{\text{tun}} + \sum_i C_i V_i}{C_{\text{tot}}} \tag{3}$$

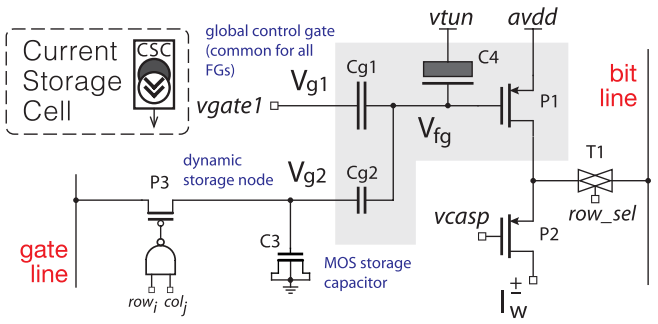

Fig. 4. Current storage cell. The CSC stores $I_w^{\pm}$, one of the weight current components, which is determined by both the FG node charge Q_{fg} (nonvolatile mode) and the two control gate voltages V_{g1} and V_{g2} (dynamic mode). The sizes of key devices are as follows: $P1 = 2 \times 2\ \mu\text{m}^2$, $P2 = 2 \times 1\ \mu\text{m}^2$, $P3 = 0.4 \times 0.35\ \mu\text{m}^2$; $Cg1 = 40$ fF, $Cg2 = 15$ fF, $C3 = 1.35$ pF, $C4 = 0.6$ fF.

where C_{g1}, C_{g2}, C_{tun}, and C_{tot} are, respectively, the first gate, second gate, tunneling, and total FG node capacitances, V_{tun} is the voltage of the *vtun* terminal, and C_i and V_i combine all parasitic coupling capacitors and voltages. The *vgate1*, *vtun*, and *vcasp* terminals are shared across all FGTs. In dynamic storage mode, V_{g1} is fixed at 2.5 V, whereas the drain current is controlled by V_{g2}, which is stored on a local sample-and-hold (S/H) circuit comprising an metal–oxide–semiconductor (MOS) capacitor C3 and a switch transistor P3. In nonvolatile storage mode, the FGT acts as a single gate transistor by connecting the second gate to the global *vgate1*. The programmed value of its drain current is defined at $V_{g1} = 2.5$ V. FGT is considered fully erased when its drain current is less than 1 pA. The cascode transistor P2 is inserted to minimize the drain coupling effect on the FG node and also to isolate the drain from the main circuit during programming. The purpose of using the dual gate with nonequal coupling ratios is explained in the subsection on dynamic programming.

A. Nonvolatile Programming

The FGT layout is similar to the one described in [23]. The source and body terminals are connected to *avdd*. The FG node is electrically isolated from the control gates by two poly-to-poly capacitors. The tunneling capacitor is a minimum size p-channel metal–oxide–semiconductor (pMOS) transistor with its gate connected to the FG node and its source, drain, and bulk terminals connected to *vtun*. We employ two mechanisms for nonvolatile programming of FGTs. Hot-electron injection is used to add electrons to the FG, thus, lowering its voltage and increasing the drain current. Conversely, Fowler–Nordheim (FN) tunneling is used to remove electrons from the FG. Although the two mechanisms allow for bidirectional charge transfer, owing to poor controllability of FN tunneling we use it for global erase only. Individual programming is, thus, performed only using injection.

To program a FGT of interest, it is first isolated from the containing circuitry by raising *vcasp* to *avdd* and connecting its drain to a bit line by *row_sel* via the transmission gate T1. The bit line is routed by a column multiplexer to several destinations, including an external pin, ITOV, and DCCS, each controlling a specific aspect of the programming sequence.

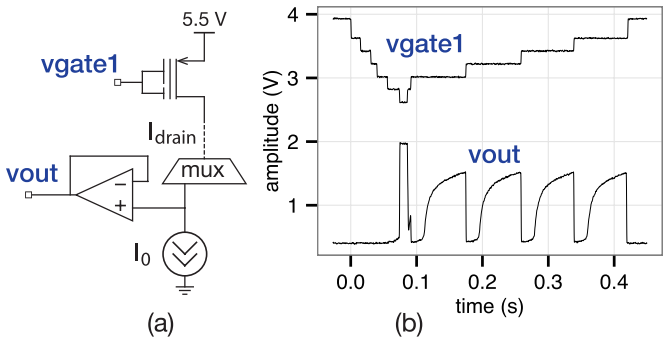

Fig. 5. *Coarse* injection stage. (a) Simplified circuit diagram. (b) Time waveform consisting of a sequence of self-terminating injection steps.

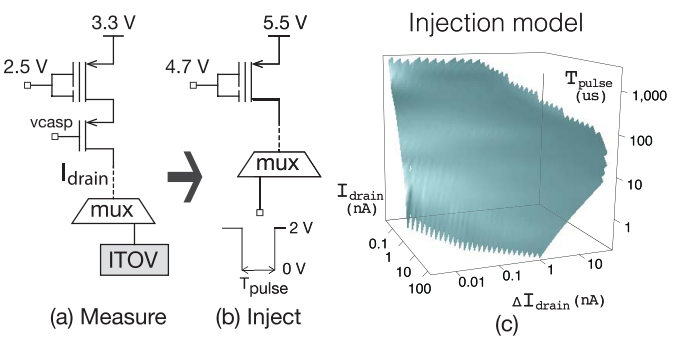

Fig. 6. *Fine* injection stage repeats the following steps. (a) Measures the initial drain current. (b) Applies a drain pulse to shorten the distance to the target current. (c) Injection model is fitted by a low-order polynomial. The cascode transistor in (a) is formed by applying *vcasp* to a pMOS from the transmission gate T1.

To prevent other FGTs from parasitic injection, *row_sel* is turned off for rows not containing the target FGT, and nonactive bit lines are tied to *avdd*. The injection occurs when a large source-to-drain (SD) potential is applied to the FGT biased at a nonzero drain current. This condition is created by raising *avdd* (to 5.5 V in our experiments) and pulling the drain to a low potential (near ground), while setting *vgate1* to produce a nonzero drain current. The programming is accomplished in two stages: 1) *coarse* stage, which employs a negative feedback to inject the FGT slightly below a target current and 2) *fine* stage, which achieves the desired accuracy using a fitted injection model.

The negative feedback injection is made possible by connecting the drain of the target FGT to DCCS, sourcing a fixed current I_0 (e.g., 20 nA) [Fig. 5(a)]. The voltage on *vgate1* is ramped down until the FGT starts conducting a nonzero current I_{drain}. At this point, the drain potential (represented by *vout*) is almost at the ground level (due to $I_{\text{drain}} < I_0$), creating a favorable condition for injection. The SD potential drops when I_{drain} exceeds I_0 and the FGT stops injecting [Fig. 5(b)]. In a series of subsequent steps, raising *vgate1* by a small amount resets the drain current such that injection can start again. The drain current by the end of the coarse injection is roughly 10 pA, which is below the range of programmable currents, yet sufficient to initiate injection in the fine stage.

During the second stage, *vgate1* is fixed and at its final value (i.e., 2.5 V). For accurate injection, we adopt the algorithm described in [24]; however, we use a pulse-width instead of a drain voltage modulation, that is, the injected charge is modulated by T_{pulse}, the amount of time the drain is kept low, while the SD voltage is fixed. Prior to any programming, we characterize a random set of FGTs and fit a polynomial of the form

$$\log_{10}(T_{\text{pulse}}) = P(\log_{10}(I_{\text{drain}}), \log_{10}(\Delta I_{\text{drain}})) \tag{4}$$

which predicts the duration of a pulse needed to increase an initial current I_{drain} by the amount ΔI_{drain}. Programming to a target current I_{target} then proceeds in a series of steps comprising: 1) measuring the initial current I_{drain} and 2) applying a drain pulse for the duration T_{pulse} predicted by the injection model (Fig. 6). In fact, we set ΔI_{drain} to $\alpha(I_{\text{target}} - I_{\text{drain}})$ with $\alpha < 1$ to prevent accidental overinjection owing to variations in injection characteristics of individual devices. As a result, several injection pulses are required to reach the target current within a given tolerance.

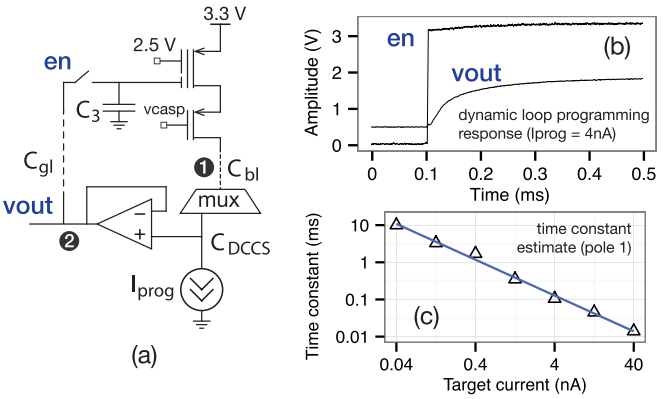

Fig. 7. Dynamic programming setup. (a) Simplified circuit diagram. (b) Time response of the feedback loop (*vout*) after closing the switch. (c) Estimated time constant of the loop response. Estimated capacitances: $C_{\text{bl}} = 2.5$ pF, $C_{\text{DCCS}} = 0.25$ pF.

Finally, for bulk erase of the entire array we apply high-voltage pulses to *vtun* while grounding the control gates of FGTs. The actual value of *vtun* varies from 8 to 9 V to achieve the necessary 10 MV/cm electric field across the tunneling oxide [25], which is 8 nm thick.

B. Dynamic Programming

Dynamic storage is employed during training for fast bidirectional weight updates. Fig. 7 shows a simplified circuit configuration of the dynamic programming. In this mode, a CSC of interest is isolated from the array in a way similar to the nonvolatile mode. However, the bit and gate lines are connected to the output of DCCS, which generates a target current I_{prog}, charging the dynamic capacitor C3 and forcing the diode-connected FGT to supply equivalent current. Once this self-biasing loop is stabilized, the switch transistor disconnects the gate which now stores a new voltage value on its dynamic capacitor. Although this programming scheme is conceptually simple, it is important to address several design considerations affecting its performance and, more broadly, the overall network learning ability.

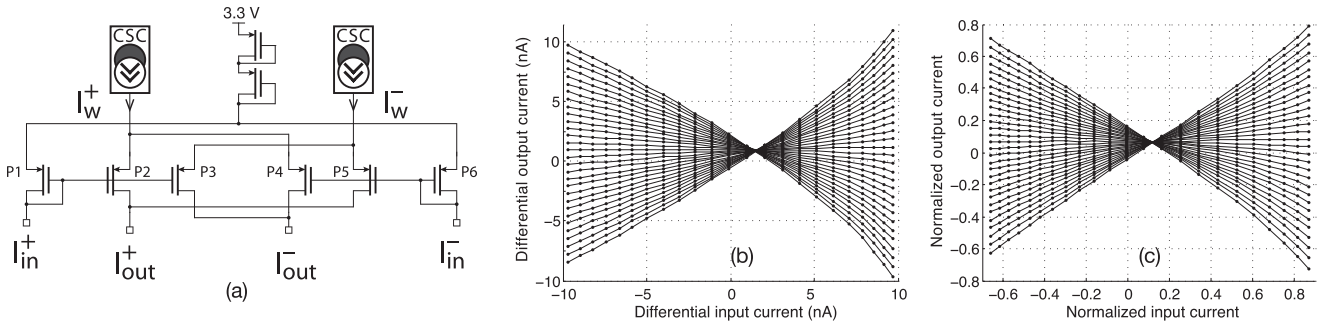

Fig. 8. (a) Synapse circuit schematic ($P1 = \cdots = P6 = 4 \times 2\ \mu\text{m}^2$). (b) Measured characteristic showing I_{out}^D versus I_{in}^D and parameterized by I_w^D sampled equidistantly from -10 to 10 nA. The input current is generated by GM. (c) Same characteristic in the normalized notation showing that $I_{\text{out}}^X \sim I_{\text{in}}^X \times I_w^X$.

The sequential nature of dynamic weight updates contributes to the overall training time. The time to program an individual CSC is dominated by the settling time of the negative feedback loop formed by the FGT, DCCS, the unity gain op-amp, and the S/H circuit [Fig. 7(a)]. The loop has two major poles: 1) at the output of DCCS driving the bit line capacitance C_{bl} and 2) at the output of the op-amp charging the gate line capacitance C_{gl} and the dynamic storage capacitance C_3. Fig. 7(b) shows a measured time response of the gate voltage (a buffered version of the bit line voltage), which resembles the settling behavior of a single-pole system. Indeed, the op-amp is designed and biased to move the second pole outside the unity gain frequency of the loop transfer function. As a result, the closed loop is stable and has a well-behaved time response which can be approximated by

$$v(t) = v(0)[1 - e^{-t/\tau}] \tag{5}$$

where $v(t)$ is the bit line voltage and τ is the time constant of an equivalent RC circuit. The estimated time constant varies over three orders of magnitude for target currents from 40 pA to 40 nA, as shown in Fig. 7(c). The actual length of the programming window is adjusted according to the target current and consists of several time constants to allow for the drain current to settle to within 1% of its target value. Note that the primary reason of using the unity gain op-amp was to decouple the large values of C_{gl} and C_3 from the capacitance seen by DCCS, which would otherwise contribute to a longer settling time.

Beyond the settling time, several design choices were made toward diminishing parasitic effects of the switching transistor P3 (such as clock feedthrough, charge injection, and reverse-bias leakage current). For example, the dynamic storage node is connected to a low coupling input of the dual-gate FGT with the intention of reducing transconductance and, thus, sensitivity to voltage disturbances on the dynamic capacitor. A minimum size pMOS as a switching transistor and a large 1.35 pF n-channel MOS (nMOS) as a dynamic capacitor further reduce parasitic effects. The pMOS capacitor is chosen for its large unit capacitance, while ignoring the C–V nonlinearity because it is used in static mode only. Equally important are the employed layout techniques, including metal shielding and guard rings around sensitive nodes. Lastly, the accuracy of dynamic programming is limited by the resolution of DCCS, which is considered in subsequent sections.

VI. Implementation of Neural Circuits

A. Synapse Circuit

The synapse circuit, shown in Fig. 8(a), implements a four-quadrant multiplication function [26]. The circuit features two CSC cells for the storage of differential weight components and a six-transistor core P1–P6. Applying the translinear principle to the core operating in the subthreshold region, the output differential and CM currents are obtained by

$$I_{\text{out}}^D = \frac{I_{\text{in}}^+ - I_{\text{in}}^-}{I_{\text{in}}^+ + I_{\text{in}}^-} \cdot (I_w^+ - I_w^-); \quad I_{\text{out}}^{\text{CM}} = I_w^+ + I_w^- \tag{6}$$

where I_{in}^+, I_{in}^- are the differential components of the input signal and I_w^+, I_w^- are the differential components of the weight value. In normalized current notation, the synapse equation becomes

$$I_{\text{out}}^X = I_{\text{in}}^X \cdot I_w^X \tag{7}$$

where I_{in}^X, I_w^X, and I_{out}^X are the input, weight, and output normalized currents, respectively. Note that the above equations assume that the core transistors are identical and there is no mismatch. Equation (6) reveals an important characteristic of the synapse circuit: the input signal enters the equation in the normalized form, which effectively limits the range of the first multiplier to $[-1, 1]$, regardless of the CM signal.

The actual measured characteristic of one of the synapses appears to be nonlinear and has a dc offset [Fig. 8(b) and (c)]. It is worth mentioning that the effect of mismatch on the entire network is more profound than individual synapses or neurons having deviations from their ideal characteristics. For example, current mirrors used in the design to deliver signals to multiple recipients (e.g., an input signal propagating to all synapses in a hidden layer) introduce a significant error because of their distributed nature and process variation across long distances. Small device geometries and subthreshold currents result in each recipient seeing as much as 30% variation of the original signal [27]. Learning around these nonidealities is achieved by a chip-in-the-loop training strategy described further below.

B. Neuron Circuit

The neuron circuit applies a nonlinear activation function to the sum of the outputs of the connected synapses. Current summation is realized by connecting positive and negative

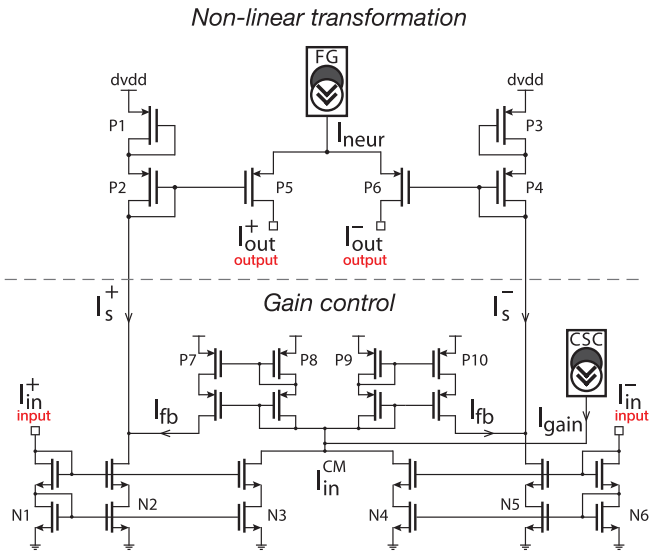

Fig. 9. Neuron circuit schematic. The bottom part controls the gain, and the top part implements nonlinear conversion. All pMOS and nMOS transistors have size $4 \times 1\ \mu\text{m}^2$.

components of the synaptic outputs. Denoting the output currents of the N connected synapses as $I_1, I_2, \ldots, I_N$, the input current of the neuron circuit has the following components: $I^D_{\text{in}} = \sum_i I^D_i$ and $I^{\text{CM}}_{\text{in}} = \sum_i I^{\text{CM}}_i$. If the CM components of the synaptic outputs are equal, the normalized input current becomes

$$I^X_{\text{in}} = \frac{I^D_{\text{in}}}{I^{\text{CM}}_{\text{in}}} = \frac{\sum_i I^D_i}{\sum_i I^{\text{CM}}_i} = \frac{1}{N}\sum_i I^X_i. \tag{8}$$

Since each of the I^X_i is limited in absolute value by 1, I^X_{in} can become very small for a large number of connected synapses (or, equivalently, $I^{\text{CM}}_{\text{in}} \gg I^D_{\text{in}}$). This poses difficulties to the neuron circuit implementation that has to accommodate signals with large dynamic range. As a solution, we consider a neuron circuit as consisting of two stages (Fig. 9). The first stage (also known as the gain control) adjusts the input current to use the full range of the neuron's activation function. More specifically, the CM component of the input signal is attenuated to obtain an intermediate signal I_s, which more uniformly spans its range and has a well-defined CM current. The second stage performs a *tanh*-like nonlinear conversion of the intermediate signal.

This gain control produces an intermediate signal I_s by subtracting a feedback current I_{fb} from mirrored copies of I^+_{in} and I^-_{in}. I_{fb} is formed as a difference between $I^{\text{CM}}_{\text{in}}$ (created by N1–N3 and N4–N6 current mirrors) and I_{gain}, which is stored locally in a CSC. This difference is split between the diode-connected P8 and P9, which reproduce halved copies in the identically sized P7 and P10, that is, $I_{\text{fb}} = \max\{1/2(I^{\text{CM}}_{\text{in}} - I_{\text{gain}}), 0\}$. As a result, the intermediate current components become $I^+_s = \max\{I^+_{\text{in}} - I_{\text{fb}}, 0\}$ and $I^-_s = \max\{I^-_{\text{in}} - I_{\text{fb}}, 0\}$. To proceed with the analysis, we consider two cases. In the first case, the input signal has a small CM component, that is, $I^{\text{CM}}_{\text{in}} \le I_{\text{gain}}$. As a result, the feedback current is zero and the input passes without change, that is, $I^X_s = I^X_{\text{in}}$. Now consider the case when $I^{\text{CM}}_{\text{in}} > I_{\text{gain}}$. It is easy to show that the

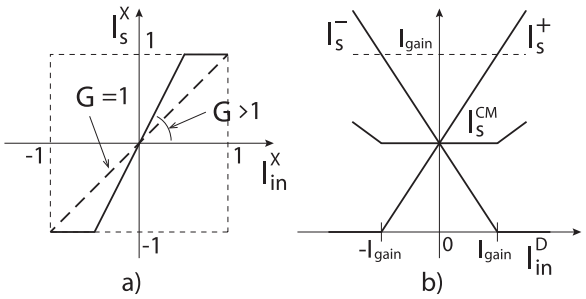

Fig. 10. Gain control circuit's behavior. (a) Input–output relationship in terms of normalized currents. (b) Individual components versus differential input current.

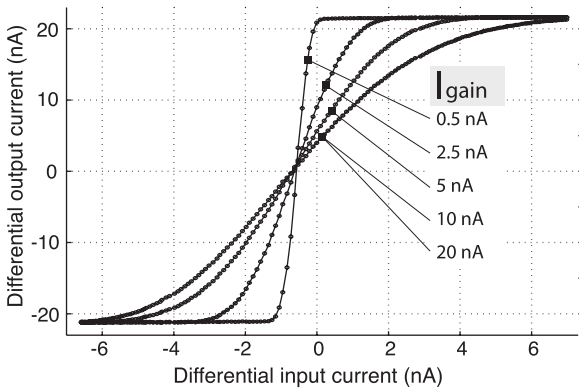

Fig. 11. Measured neuron transfer characteristic for $I_{\text{neur}} = 20$ nA, $I^{\text{CM}}_{\text{in}} = 10$ nA and $I^D_{\text{in}} \in [-7, 7]$ nA.

intermediate current components become $I^+_s = \max\{(1/2)I^D_{\text{in}} + (1/2)I_{\text{gain}}, 0\}$ and $I^-_s = \max\{-(1/2)I^D_{\text{in}} + (1/2)I_{\text{gain}}, 0\}$. The CM current is $I^{\text{CM}}_s = I_{\text{gain}}$ for $I^D_{\text{in}} \le I_{\text{gain}}$ and grows as $I^{\text{CM}}_s = (1/2)I_{\text{gain}} + (1/2)I^D_{\text{in}}$ for $I^D_{\text{in}} > I_{\text{gain}}$. In terms of normalized currents, we have

$$I^X_s = \min\left\{\frac{I^D_{\text{in}}}{I_{\text{gain}}}, 1\right\} = \min\left\{\frac{I^{\text{CM}}_{\text{in}}}{I_{\text{gain}}} \cdot I^X_{\text{in}}, 1\right\} = \min\left\{G \cdot I^X_{\text{in}}, 1\right\} \tag{9}$$

where $G = I^{\text{CM}}_{\text{in}}/I_{\text{gain}} \ge 1$ is the gain factor. This equation is shown in Fig. 10: a small-valued input I^X_{in} is amplified by the gain G and clipped at ± 1. A minimum gain of 1 (broken line) is achieved when $I^{\text{CM}}_{\text{in}} \ge I_{\text{gain}}$ with the input current being copied to the output. Fig. 10(b) shows individual components of I_s as a function of the differential input current.

The top part of Fig. 9 shows a *tanh*-like activation function [26]. Assuming the transistors P1–P6 are identical and using the translinear principle, the output differential and CM currents are obtained by

$$I^D_{\text{out}} = \frac{(I^+_s)^{\frac{1+\kappa}{\kappa}} - (I^-_s)^{\frac{1+\kappa}{\kappa}}}{(I^+_s)^{\frac{1+\kappa}{\kappa}} + (I^-_s)^{\frac{1+\kappa}{\kappa}}} \cdot I_{\text{neur}}, \quad I^{\text{CM}}_{\text{out}} = I_{\text{neur}} \tag{10}$$

where κ is the subthreshold slope and I_{neur} is the CM output current. Figs. 11 and 12 show the measured characteristics of the entire neuron circuit. The input current is generated by GM with $I^{\text{CM}}_{\text{in}} = 10$ nA, and the output is measured by ITOV. For low values of I_{gain}, the characteristic resembles a step-like

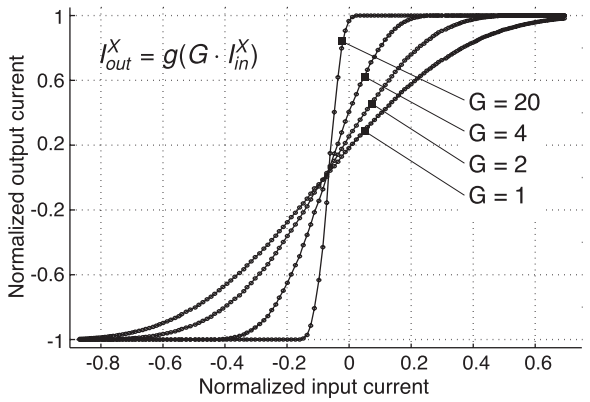

Fig. 12. Measured neuron transfer characteristic displayed in normalized current notation.

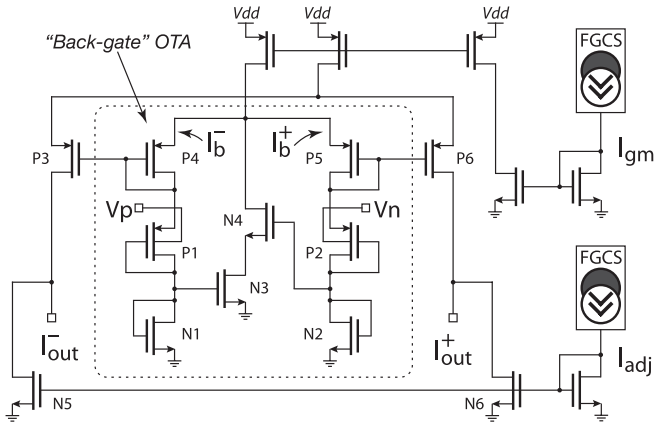

Fig. 13. Schematic of the differential transconductor GM employing a wide-linear-range back-gate OTA and a wrapper circuit for input range adjustment. All pMOS and nMOS transistors have size $4 \times 1\ \mu m^2$.

function because high G. As expected, setting I_{gain} larger than 10 nA, that is, $I_{gain} > I_{in}^{CM}$, results in $G = 1$, which is why the last two curves appear overlaid (for $I_{gain} = 10$ and 20 nA).

VII. Implementation of Peripheral Circuits

A. Differential Transconductor (GM)

The neural platform accepts voltage-encoded signals as its input features, while performing the necessary voltage-to-current mapping internally with the help of GM circuits (Fig. 3). It is desirable that the GM circuit has a wide linear input range (low transconductance) and produces a differential current with controlled CM signal. A conventional operational transconductance amplifier (OTA) is not suitable due to an extremely narrow linear input range (several thermal voltage constants) when biased in the subthreshold region. Instead, we designed a circuit shown in Fig. 13. The core of the circuit is a back-gate subthreshold OTA (shown inside the dashed box), the detailed description of which is given in [28]. The wide linear range is achieved by employing the back-gate effect {P1, P2}, source {P4, P5} and gate {N1, N2} degeneration, and bump linearization {N3, N4}.

The original back-gate OTA features a fixed transfer characteristic regardless of its CM current I_{gm}. In fact, the branch currents I_b^+ and I_b^- never reach zero, which means

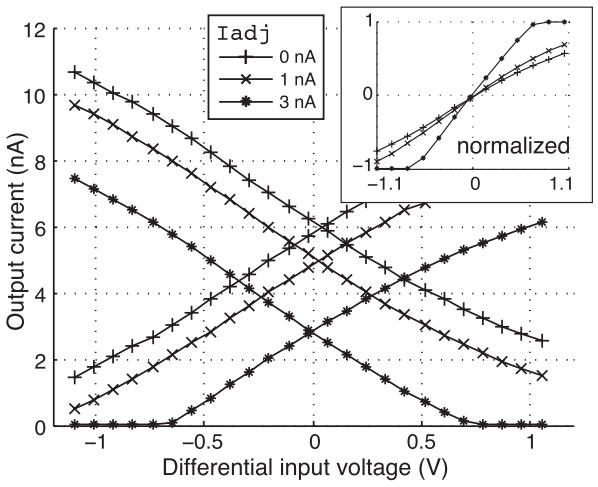

Fig. 14. GM circuit measured characteristic: I^+_{out} and I^-_{out} versus differential input voltage for various values of I_{adj}. Vp is swept from 1.1 to 3.3 V, and Vn is fixed at 2.2 V. Inset: Y-axis in the normalized current notation.

that $-1 < I_b^X < 1$. To adjust the input range of GM, we employ a wrapper circuit around the back-gate OTA which generates I_{out} by subtracting I_{adj} from the scaled version of I_b generated by P3–P6. Fig. 14 shows the measured I^+_{out}, I^-_{out} plotted against the differential input voltage for various values of I_{adj}. Without the adjustment current ($I_{adj} = 0$), the two output components are always greater than zero for any allowed range of input voltages. Alternatively, nonzero values of I_{adj} limit the range of input voltages for which the output spans its entire range (inset of Fig. 14). This is crucial when GMs are connected to sensors with a limited output range, as is the case in the target applications. Note that the V–I mapping represents a part of the neural signal flow with inherent nonlinearity (as seen in Fig. 14) and is subject to process variation similar to the synapse and neuron circuits.

B. Digitally Controlled Current Source

DCCS is responsible for generating target currents for the dynamic memory programming. The high-level block diagram of DCCS is shown in Fig. 15(a). Its operation is based on a current splitter [29], which divides the input current down by each of its branches with ratio N, as shown in Fig. 15(b). The first splitter coarsely divides the on-chip reference bias to produce several range currents according to the values of $\mathbf{w}_{range}$, each defining a separate interval of generated currents. A precise value within each interval is produced by two identical 12-b splitters connected in parallel. This combined splitter greatly reduces differential nonlinearity by introducing a multitude of redundant codes, filling the gaps of a single splitter. A detailed description of this approach is given in [30]. To make the best use of redundancy, the DCCS is first characterized by measuring its 24 branch currents for each interval with a picoammeter and storing them in a table. To generate a target current, this table is used to find a subset of branches whose combined current closely matches the target value.

For better time response and accuracy of generated currents, we employed an active current mirror at the output stage

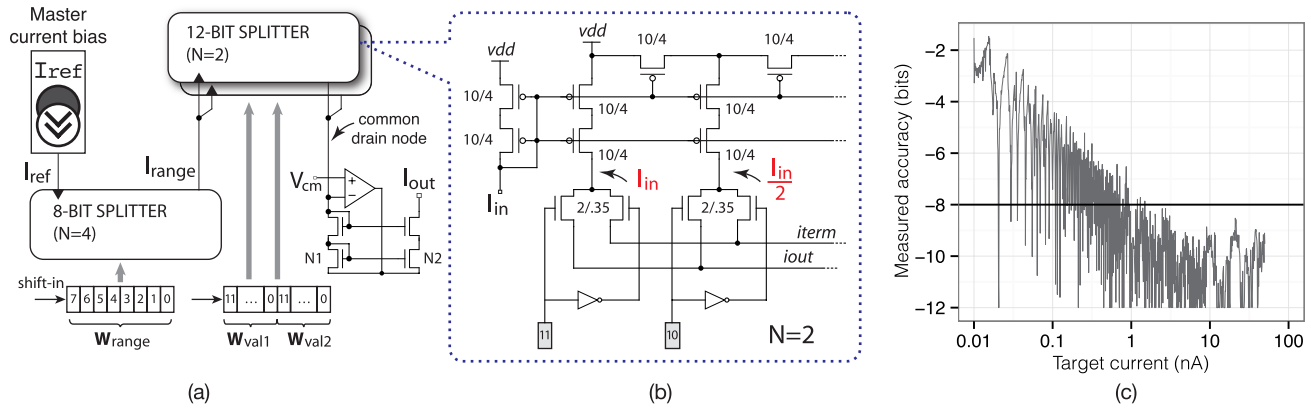

Fig. 15. DCCS is used to generate target currents for dynamic weight programming. (a) Block diagram of DCCS: an 8-b $\mathbf{w}_{\text{range}}$ sets the range of generated currents, and a combined 24-b [$\mathbf{w}_{\text{val1}}$, $\mathbf{w}_{\text{val2}}$] determines an exact value of the output current. (b) First two branches of a 12-b current splitter. (c) Measured accuracy of currents generated from 10 pA to 50 nA for a fixed $\mathbf{w}_{\text{range}} = 2^4$. The Y-axis is shown in the form of $\log_2|I_{\text{tar}} - I_{\text{meas}}|/I_{\text{tar}}$, where I_{tar} is the current being programed and I_{meas} is the actual current measured by Keithley 6485 picoammeter. Assuming that the error is an LSB of the generated value, this form can be interpreted as bit resolution at each particular current value.

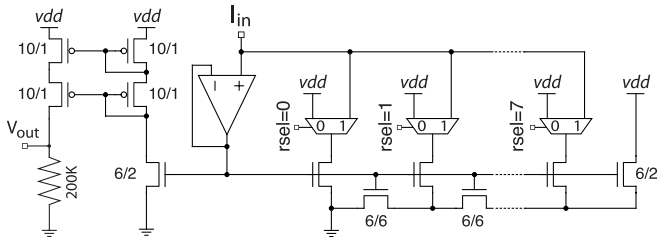

Fig. 16. Schematic of ITOV converter.

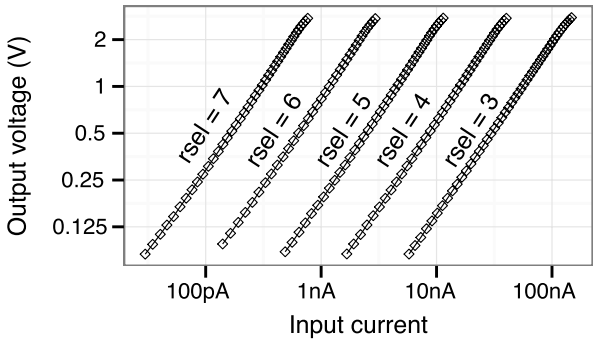

Fig. 17. Output voltage measurements of ITOV for *rsel* from 3 to 7 covering the range of currents from 20 pA to 150 nA.

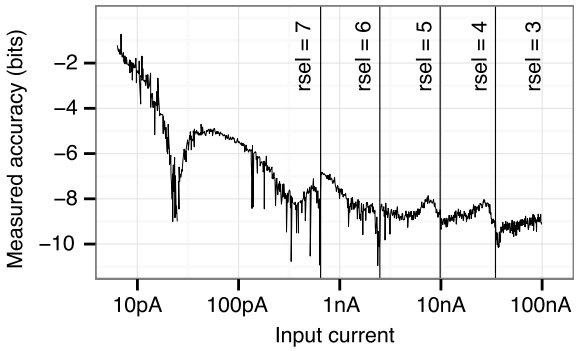

Fig. 18. Measured accuracy of ITOV. The input current is swept from 10 pA to 100 nA. Y-axis: accuracy expressed as $\log_2|I_{\text{in}} - \hat{I}_{\text{in}}|/I_{\text{in}}$, where $\hat{I}_{\text{in}}$ is the current inferred via ITOV and I_{in} is the current measured by Keithley 6485.

of the DCCS [31]. The negative feedback clamps the drain voltage of all branches to V_{cm}, eliminating the channel-length modulation in the branch transistors and greatly reducing the delay owing to charging parasitic capacitance of the common drain node. The fact that the source voltage of N1 can rise above its gate voltage allows us to reliably generate currents in the sub-nA range [27]. The accuracy of DCCS is evaluated by measuring the error between the target I_{tar} and the measured I_{meas} output currents. We consider currents from the range given by $\mathbf{w}_{\text{range}} = 4$, which covers the default range of operating currents, that is, up to 10 nA. Fig. 15 shows the measured accuracy expressed as $\log_2|I_{\text{tar}} - I_{\text{meas}}|/I_{\text{tar}}$, which can be interpreted as bit resolution at each particular current value. Note that decreasing accuracy for currents below 1 nA is primarily due to lower density of redundant codes, which can be fixed by decreasing the range $\mathbf{w}_{\text{range}}$. The absolute error is limited by 40 pA, which for the default range of currents corresponds to an 8-b resolution.

C. Current-to-Voltage Converter

Our fast current measurement system consists of an on-chip ITOV converter and an off-chip mapping from voltage values to corresponding currents. The ITOV circuit consists of a programmable current amplifier followed by a resistor, as shown in Fig. 16. The current amplifier is derived from a basic eight-input current splitter by rearranging it in a diode-connected fashion. The control word *rsel* choses which branch an input current flows in, thus controlling the amplification factor. Fig. 17 shows five measured characteristics for *rsel* from 3 to 7, that together cover an input range from 20 pA to 150 nA. We fit polynomials to all five characteristics that are later used to infer a current value from V_{out}. To test the accuracy of the current measurement system, we sweep the input current (via one of the FGTs) across the entire range of interest and measure it using both the ITOV and an external picoammeter. The measured accuracy in the form of bit resolution is shown in Fig. 18. Note that the resolution remains fairly constant

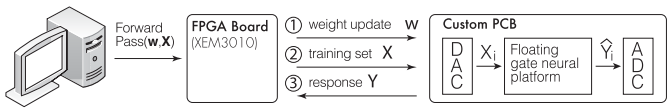

Fig. 19. Chip-in-the-loop experimental setup. A custom printed circuit board provides the stimuli generation/response measurement functionality as well as the high-voltage interface for nonvolatile programming. An FPGA board implements the logic behind the forward pass, dynamic, and nonvolatile programming.

Algorithm 1 Resilient Backpropagation for MLP

Initialize Δ
Randomly initialize $\mathbf{w}$
repeat
  $E_{\text{nopert}} \leftarrow \sum_i (Y_i - \mathbf{F}(\mathbf{w}, X_i))^2$
  $E_{\text{pert}} \leftarrow \sum_i (Y_i - \mathbf{F}(\mathbf{w} + \mathbf{w}^{\mathbf{pert}}, X_i))^2$
  $sign(\partial E/\partial \mathbf{w}) \leftarrow sign((E_{\text{pert}} - E_{\text{nopert}})/\mathbf{w}^{\mathbf{pert}})$
  for each w_k
    Update Δ_k
    $w_k \leftarrow w_k - sign(\partial E/\partial w_k) \cdot \Delta_k$
  end
until stopping criterion

(about 8 b) for currents above 1 nA and drops significantly for currents below 30 pA. The latter is due to the output voltage of ITOV being close to ground, which can be improved using a splitter with higher number of branches, that is, a higher multiplication factor.

VIII. Hardware-Friendly Learning Strategy

Learning a classification problem is a search process in the space of possible network topologies and their weight values, which results in a decision boundary that best describes the training data. Off-the-shelf training algorithms commonly available for software neural networks are hardly suitable for the neural platform due to physical constraints imposed by the analog implementation. Instead, we employ a chip-in-the-loop strategy (shown in Fig. 19), whereby the training is done by a computer which implements a gradient descent search by measuring a response of the neural platform to an input vector of features. Essentially, the neural platform implements a forward pass, which includes: 1) dynamically programming a set of weights $\mathbf{w}$; 2) presenting the neural chip with an input vector X_i; and 3) measuring its response $\hat{Y}_i$. From the software perspective, it is just a function call $\mathbf{F}(\mathbf{w}, X_i)$, which is used by the following training algorithms.

A. MLP Training Algorithm

For training MLP, we employ the method presented in Algorithm 1, which is a version of resilient back propagation known as RPROP [32]. The direction of each weight update is based on the sign of the error gradient $\partial E/\partial w_k$. Here, the error measure E is given by

$$E(\mathbf{w}) = \sum_i (Y_i - \mathbf{F}(\mathbf{w}, X_i))^2 \tag{11}$$

where X_i and Y_i are the input vector and the target value of the ith observation. The gradient itself is estimated by perturbing the weights by a small random vector $\mathbf{w}^{\mathbf{pert}}$ and calculating a first-order difference

$$\partial E/\partial w_k \approx \frac{E(\mathbf{w} + \mathbf{w}^{\mathbf{pert}}) - E(\mathbf{w})}{w_k^{\text{pert}}} \tag{12}$$

which requires two forward passes of the training set through the neural network. The size of each weight update is represented by a local parameter Δ_k, which is adjusted in each iteration using the following rule: if the gradient does not change sign for two consecutive iterations, Δ_k is increased, otherwise, it is decreased. The step-size adaptation allows for an accelerated descent along shallow regions, while reducing the step near local minima for better convergence. As far as the algorithm is concerned, the weights and input features are treated as dimensionless variables limited in absolute values by 1. The translation into physical current components is done by $I_w^{\pm} = 0.5 I_w^{\text{CM}}(1 \pm w)$ with I_w^{CM} set to 10 nA (default operating range). The minimum step-size value $\Delta_{\min}$ and perturbation vector components are determined by the resolution of the dynamic programming and are set to 0.005 for subsequent experiments.

B. ONN Training Algorithm

For the ONN network topologies, we employ a CC algorithm [33]. Training starts with a minimum size topology including only an output neuron. Each subsequent neuron adds on to the learning capacity of the model, improving the error on the training set. Each iteration of the training algorithm consists of two steps. Suppose that our current topology has $M-1$ hidden neurons [Fig. 2(b) for reference]. Let $Y_j^H(X_i)$ represent the output of the jth hidden neuron for the training vector X_i and $Y_k^O(X_i)$ be the network output when it has k hidden neurons. In the first step, an Mth hidden neuron is added at the bottom so that it receives the primary inputs $X_1, \ldots, X_P$ as well as the outputs of all preexisting neurons $Y_1^H, \ldots, Y_{M-1}^H$. This neuron is trained to maximize the covariance between its output $Y_M^H(X_i)$ and the residual training error of the previous iteration $Y_{M-1}^E(X_i) = Y_i - Y_{M-1}^O(X_i)$. This covariance is given by

$$C = \sum_i \left(Y_M^H(X_i) - \overline{Y_M^H}\right)\left(Y_{M-1}^E(X_i) - \overline{Y_{M-1}^E}\right) \tag{13}$$

where $\overline{Y_M^H}$ and $\overline{Y_{M-1}^E}$ are averaged quantities over the entire training set. The covariance maximization is equivalent to the minimization of $-C$, which can be accomplished by the RPROP algorithm with the error function $E = -C$. Once the covariance is maximized, the weights of this neuron become permanent and the output layer is retrained to minimize the error on the training set (11). The second step is again completed by the RPROP algorithm. Note that in each iteration, only the weights of the neuron being added undergo modification, followed by the weights of the output neuron, while the other weights are kept unchanged. This feature

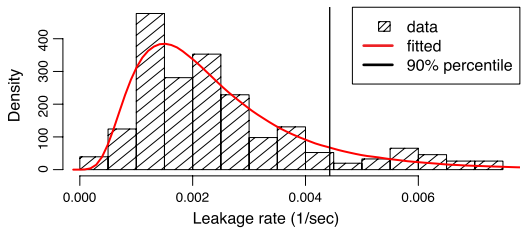

Fig. 20. Histogram of weight leakage rate estimates with a corresponding fitted lognormal density function. Vertical line: 90th percentile of the fitted distribution.

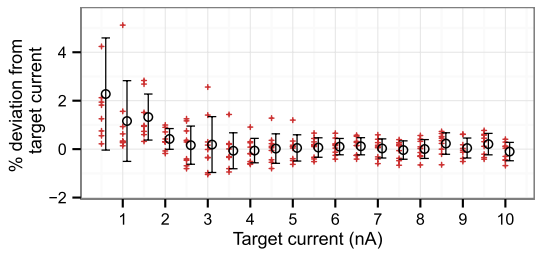

Fig. 21. Deviation of programmed currents from their target values measured on a 20 × 10 array of FGTs.

greatly simplifies the gradient estimation by the hardware and leads to stable performance even for large-sized topologies. Hidden neurons are added until one of the following two events is achieved, depending on the classification task: a zero error on the training set or a target error on a validation set.

IX. Experimental Results

In this section, we put the neural platform to the test to evaluate its performance and learning ability on several classification tasks. First, we present the results of weight leakage characterization in dynamic mode and the accuracy of weight programming in nonvolatile mode. Next, the network is trained on an XOR2 task and its performance is compared with similar implementations reported in literature. Finally, we consider higher dimensional N-input parity tasks and a 2-D two-spirals task and train analog neural networks alongside their software counterparts for comparative analysis.

A. Weight Decay

Errors in dynamic programming directly affect the learning ability of the analog neural network. Reverse-bias leakage current of the switch transistor P3 (Fig. 4) is particularly important because it determines a time interval during which the weight change remains insignificant. We consider the weight leakage rate as a relative change of the weight current ΔI_w per unit time, or $\Delta I_w/(\Delta t \cdot I_w)$. Fig. 20 shows a histogram of the leakage rate estimates collected for 35 randomly selected CSCs. Each cell was programmed to a number of initial currents from 0.1 to 30 nA. The normal distribution of the process variation parameters suggests that the leakage rate belongs to the family of lognormal distributions (leakage current affects the exponential part of the output drain current). Indeed, the Kolmogorov–Smirnov test does not reject the null hypothesis ($p-$value $= 0.8$). The leakage rate corresponding to the 90th percentile is 0.00443 s^{-1}. If we assume that the weight value is represented by an 8-b word, a 1-b change occurs in 0.88 s. It has been observed that the change of 1-b results in virtually no change in the network output. To limit the exposure to weight decay, large datasets (>500 observations) are split into smaller chunks during the forward pass and separated by reprogramming the dynamic memory.

B. FGT Programming Accuracy

To characterize the injection accuracy, a 20 × 10 array of FGTs was programmed to different target currents and measured with an external picoammeter. In particular, the programmed currents varied from 0.5 to 10 nA with a 0.5 nA step along the vertical direction, so that each row received identical currents. To avoid overinjection α was set to 0.5 (Section V-A). The injection sequence stopped after the programmed currents reached their target values within 20 pA accuracy. The number of injection pulses varied in the range 5–20 depending on the values of target currents. Fig. 21 shows percentage errors of measured currents from their target values. Average and maximum programming errors are 21 and 70 pA implying an accuracy of 8.9 and 7.1 bs for the average and worst cases, respectively, for the range of currents from 10 pA to 10 nA. This precision turned out to be sufficient for the purpose of retaining the network's classification accuracy after copying weights onto the nonvolatile memory. The residual programming error could be further reduced by improving the accuracy of the on-chip ITOV converter.

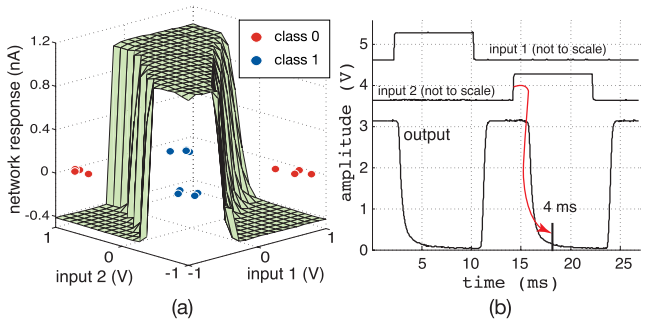

Fig. 22. 2-1-1 ONN trained on XOR2 task. (a) Decision surface obtained by measuring network response on a fine grid in input space. Also shown are input vectors with added noise. (b) Transient response recorded from the voltage output pin of ITOV.

C. Performance Evaluation on XOR2

The purpose of the first experiment is to demonstrate the low power ability and to make comparison with other similar implementations, which also provide results for the two-input XOR. For this experiment, we employed an ONN classifier and limited the operating currents to 1 nA, which is below the intended range of 10 nA. Fig. 22(a) shows the output produced by the trained ONN with one hidden neuron and weight values stored in the nonvolatile memory. The output neuron is programmed for high gain by setting I_{gain} to 0.2 nA, which explains the sharp rail-to-rail response. The transient characteristic for all input combinations is presented in Fig. 22(b). The worst case response time is 4 ms, which

TABLE I
SYSTEM PERFORMANCE AND COMPARISON

	ETANN [16]	[18]	this work
Technology	1 um CMOS	0.35 um, DP	0.35 um, DP
Weight storage	floating gate	floating gate	FG + dynamic
Learning models	MLP	VMM+WTA	MLP+ONN
Learning strategy	off-chip	off-chip	chip-in-the-loop
Synapse current	20 um @5V	10 nA @2.4V	2 nA @3.3V
Response time	5 us	NA	4 ms
Total power (XOR2 taks)	NA	700 nW	66 nW
Computational efficiency	1.3 GMAC/s/W	11–14 TMAC/s/W (theoretical)	57 GMAC/s/W (measured)

also includes the propagation delay owing to the GM and ITOV converters. The total power consumed by the trained 2-1-1 ONN classifier is 66 nW, resulting in the computational efficiency of 57 GMAC/s/W.[1] Note that when in standby mode, *vcasp* is tied to *avdd* (Fig. 4) which effectively shuts off internal current sources, bringing the power consumption to zero. On the basis of the measured data, Table I summarizes performance characteristics of the introduced neural platform along with two other similar implementations employing FGTs for weight storage. The key advantages of our chip are the extremely low power consumption and the robust chip-in-the-loop training, which effectively avoids inaccuracies introduced by off-chip emulation models and, thus, has greater potential of learning more complicated classification tasks.

D. Learning N-Input Parity Tasks (XOR2-6)

N-input parity is a popular benchmark used in evaluating the learning ability and convergence speed of new training algorithms, both in software and in hardware. Learning this task is computationally challenging as it requires allocating a boundary between any pair of adjacent codes that differ in a single bit. In this experiment, we trained both of the supported hardware classifiers alongside their software counterparts for a series of benchmarks from XOR2 to XOR6. In all cases, the software classifiers had identical topologies and served as a base line for comparison. The software version of MLP was trained by the MATLAB Neural Networks toolbox by using an RPROP algorithm. For the software ONN classifier, we implemented a CC algorithm described in [33]. Software synapses and neurons were ideal multipliers and hyperbolic tangent functions, respectively, while the signals and weights were represented in a double-precision format. In hardware classifiers, we used the default range for operating currents, that is, $I_{\text{neur}} = I_{\text{gm}} = I_{w}^{\text{CM}} = 10$ nA. The values of neuron gain currents were determined empirically by analyzing convergence rates for various I_{gain}. The best values have been found to grow linearly with the number of connected synapses and amounted to a small fraction of the total input current, resulting in sharp boundary transitions between classes.

Fig. 23 shows the results of training MLP classifiers. The range of networks varied 1–12 hidden neurons and for each topology the number of converged trainings out of ten runs is reported. Surprisingly, several hardware topologies (e.g., 3-3-1) demonstrated better success rates, indicating the susceptibility of software training algorithms to local minimum traps. On average, the convergence rate of hardware is somewhat lower, which is expected given the physical constraints of the analog implementation. These results are consistent with another analog implementation [17] which reported 4/5 (XOR3) and 48/50 (XOR4) convergence rates attained by 3-4-1 and 4-7-1 networks, respectively. An average power achieved by the ten hidden neurons topology trained on the XOR6 task was measured at 8 μW with a 0.5 ms classification latency.

[1]This results in 57 GMAC/W/s × 66 nW × 4 ms = 15 MAC per classification distributed as follows: one MAC per synapse and four MAC per neuron (using four terms of tanh's Taylor series [18]).

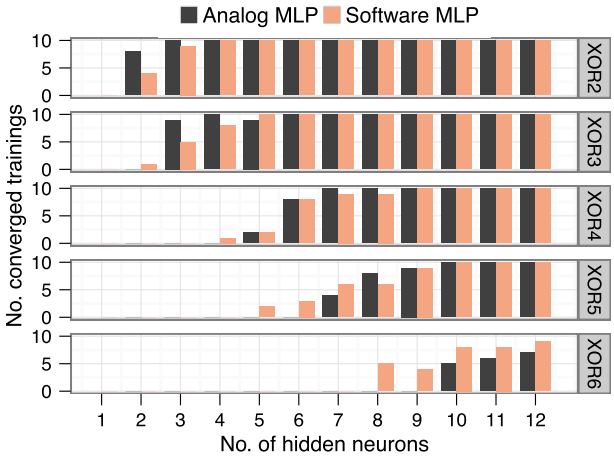

Fig. 23. Training MLP classifiers on XOR-N tasks. Each bar represents the number of converged trainings (zero classification error) out of ten attempts.

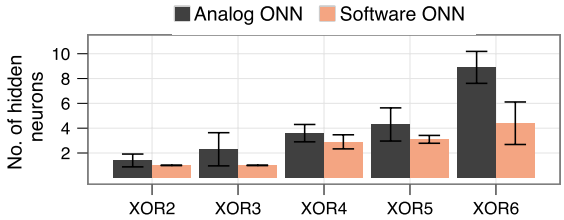

Fig. 24. Training ONN classifiers on XOR-N tasks. Plot shows an average and standard deviation of the number of hidden neurons in converged networks. Training for each task has been repeated ten times.

The results of training ONN classifiers are shown in Fig. 24. Starting with just an output layer, the training proceeded by adding hidden neurons until all input patterns were classified correctly. Note the difference in the number of hidden neurons required by the hardware and software networks to perfectly separate the spirals. This example demonstrates the need for prototyping in hardware, because selecting network topologies based on software training alone is an underestimation of the acquired hardware resources. Also note that the hardware ONN outperforms the hardware MLP for the N-input parity problems, resulting in more compact networks.

E. Two-Spirals Problem

In the final experiment, the neural platform was trained to distinguish between two interlocking spirals in

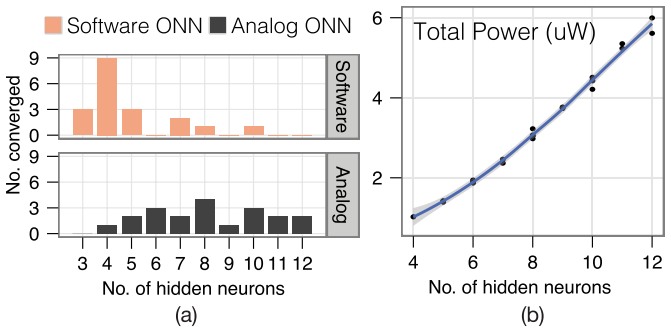

Fig. 25. Two-spirals task. (a) Distribution of the sizes of converged networks for 20 runs. (b) Power profile of the trained networks computed from weight values.

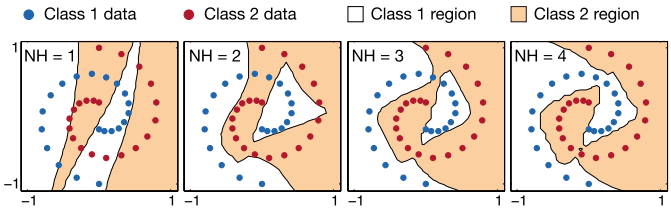

Fig. 26. Decision boundaries for the two-spirals problem drawn for each stage of the training algorithm. The boundary is created using the same method as in Fig. 22(a).

a 2-D input space. The challenges this benchmark presents to back propagation networks are often used as an opportunity to showcase the advantage of CC networks [33]. Thus, we set out to evaluate the learning ability of our hardware ONN classifiers and compare their performance to equivalent software CC networks. The training set consists of 20 points belonging to each spiral that makes a single turn around the origin in the x–y plane. Fig. 25(a) shows the distribution of converged networks for a total of 20 runs. In all trials, the hardware networks converged successfully to perfectly separate between the two spirals. Fig. 26 shows snapshots of the decision boundary between two classes produced by the hardware training, which achieved zero classification error in four steps. The power consumption can be easily obtained knowing the topology, the weight values, and the range of operating currents, which is the same as in the N-input parity problems. Fig. 25(b) shows the power profile for all trained networks, indicating a trend proportional to the network size and exhibiting little dependence on the actual weight values. Each neuron is responsible for roughly two-third of the power consumed by a single hidden unit. This becomes evident by noting that the total current produced by the synapses is copied twice inside the neuron circuit (N2–N5 and N3–N4 current mirrors in Fig. 9).

The final step of every successful training is copying weight values onto the FG memory. We verified the functionality of the trained networks immediately after FG programming and periodically for the next several days. Although the perfect separation between classes remained unchanged, the weight values underwent a small drift not exceeding 1%, which was especially noticeable during the first few minutes. High robustness of the network output to small changes in weight currents can be explained by the sharp transition of the output neuron, which is insensitive to small variations of its input, except for a narrow area around the boundary.

X. Conclusion

We presented a reconfigurable experimentation platform aimed at prototyping low-cost neural networks for on-chip integration. Comprising a large array of synapses and neurons, the platform is used to identify hardware network topologies and sizes that match the complexity of a target application. The circuit implementation addresses several cost-efficiency requirements, such as compact area and low power, nonvolatile weight storage used in long-term retention of trained functionality, and dynamic weight storage, facilitating fast bidirectional weight updates during training. Power efficiency is achieved by biasing all circuits in the subthreshold region and employing the translinear principle for analog computation. Training results from the N-input parity and two-spirals classification problems confirmed: 1) robustness of training in presence of implementation nonidealities and 2) adequate learning ability consistent with other reported analog implementations. Future work will focus on integrating custom neural networks (prototyped via this experimentation platform) into target ICs to enable BIST, trust and aging monitoring, self-healing, and other applications that may benefit from the provided capabilities.

Acknowledgment

The authors would like to thank Prof. Hasler and the members of the Integrated Computational Electronics Laboratory at Georgia Institute of Technology for their invaluable assistance with the FGT technology.

References

[1] D. Maliuk, H.-G. Stratigopoulos, H. He, and Y. Makris, "Analog neural network design for RF built-in self-test," in *Proc. IEEE Int. Test Conf. (ITC)*, Nov. 2010, pp. 1–10.

[2] N. Kupp, H. Huang, P. Drineas, and Y. Makris, "Post-production performance calibration in analog/RF devices," in *Proc. IEEE Int. Test Conf. (ITC)*, Nov. 2010, pp. 1–10.

[3] Y. Jin, D. Maliuk, and Y. Makris, "Post-deployment trust evaluation in wireless cryptographic ICs," in *Proc. IEEE Design Test Comput. (DATE)*, Mar. 2012, pp. 965–970.

[4] K. Huang, J. M. Carulli, and Y. Makris, "Parametric counterfeit IC detection via support vector machines," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFT)*, Oct. 2012, pp. 7–12.

[5] H.-G. Stratigopoulos and Y. Makris, "Nonlinear decision boundaries for testing analog circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 11, pp. 1760–1773, Nov. 2005.

[6] L. Abdallah, H.-G. Stratigopoulos, S. Mir, and C. Kelma, "RF front-end test using built-in sensors," *IEEE Des. Test. Comput.*, vol. 28, no. 6, pp. 76–84, Nov./Dec. 2011.

[7] H.-G. Stratigopoulos and Y. Makris, "Error moderation in low-cost machine-learning-based analog/RF testing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 2, pp. 339–351, Feb. 2008.

[8] D. Han, B. S. Kim, and A. Chatterjee, "DSP-driven self-tuning of RF circuits for process-induced performance variability," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 18, no. 2, pp. 305–314, Feb. 2010.

[9] D. Maliuk and Y. Makris, "A dual-mode weight storage analog neural network platform for on-chip applications," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2012, pp. 2889–2892.

[10] J. Misra and I. Saha, "Artificial neural networks in hardware: A survey of two decades of progress," *Neurocomputing*, vol. 74, nos. 1–3, pp. 239–255, 2010.

[11] B. V. Benjamin *et al.*, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proc. IEEE*, vol. 102, no. 5, pp. 699–716, May 2014.
[12] G. Indiveri *et al.*, "Neuromorphic silicon neuron circuits," *Frontiers Neurosci.*, vol. 5, no. 73, 2011.
[13] E. Painkras *et al.*, "SpiNNaker: A 1-W 18-core system-on-chip for massively-parallel neural network simulation," *IEEE J. Solid-State Circuits*, vol. 48, no. 8, pp. 1943–1953, Aug. 2013.
[14] J. V. Arthur *et al.*, "Building block of a programmable neuromorphic substrate: A digital neurosynaptic core," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jun. 2012, pp. 1–8.
[15] G. Zhang and Y. Shen, "New algebraic criteria for synchronization stability of chaotic memristive neural networks with time-varying delays," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 24, no. 10, pp. 1701–1707, Oct. 2013.
[16] H. A. Castro, S. M. Tam, and M. A. Holler, "Implementation and performance of an analog nonvolatile neural network," *Analog Integr. Circuits Signal Process.*, vol. 4, no. 2, pp. 97–113, 1993.
[17] A. J. Montalvo, R. S. Gyurcsik, and J. J. Paulos, "Toward a general-purpose analog VLSI neural network with on-chip learning," *IEEE Trans. Neural Netw.*, vol. 8, no. 2, pp. 413–423, Mar. 1997.
[18] S. Ramakrishnan and J. Hasler, "Vector-matrix multiply and winner-take-all as an analog classifier," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 2, pp. 353–361, Feb. 2014.
[19] C. R. Schlottmann and P. E. Hasler, "A highly dense, low power, programmable analog vector-matrix multiplier: The FPAA implementation," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 1, no. 3, pp. 403–411, Sep. 2011.
[20] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Englewood Cliffs, NJ, USA: Prentice-Hall, 1998.
[21] V. Honavar and L. Uhr, "Generative learning structures and processes for generalized connectionist networks," *Inf. Sci.*, vol. 70, nos. 1–2, pp. 75–108, 1993.
[22] T. Serrano-Gotarredona, B. Linares-Barranco, and A. G. Andreou, "A general translinear principle for subthreshold MOS transistors," *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 46, no. 5, pp. 607–616, May 1999.
[23] P. Hasler and J. Dugger, "Correlation learning rule in floating-gate pFET synapses," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 48, no. 1, pp. 65–73, Jan. 2001.
[24] A. Bandyopadhyay, G. J. Serrano, and P. Hasler, "Adaptive algorithm using hot-electron injection for programming analog computational memory elements within 0.2% of accuracy over 3.5 decades," *IEEE J. Solid-State Circuits*, vol. 41, no. 9, pp. 2107–2114, Sep. 2006.
[25] J. Brewer and M. Gill, *Nonvolatile Memory Technologies With Emphasis on Flash: A Comprehensive Guide to Understanding and Using Flash Memory Devices*. New York, NY, USA: Wiley, 2008.
[26] M. Valle and F. Diotalevi, "A dedicated very low power analog VLSI architecture for smart adaptive systems," *Appl. Soft Comput.*, vol. 4, no. 3, pp. 206–226, 2004.
[27] B. Linares-Barranco and T. Serrano-Gotarredona, "On the design and characterization of femtoampere current-mode circuits," *IEEE J. Solid-State Circuits*, vol. 38, no. 8, pp. 1353–1363, Aug. 2003.
[28] R. Sarpeshkar, R. F. Lyon, and C. Mead, "A low-power wide-linear-range transconductance amplifier," *Analog Integr. Circuits Signal Process.*, vol. 13, nos. 1–2, pp. 123–151, 1997.
[29] K. Bult and G. J. G. M. Geelen, "An inherently linear and compact MOST-only current division technique," *IEEE J. Solid-State Circuits*, vol. 27, no. 22, pp. 1730–1735, Dec. 1992.
[30] R. Serrano-Gotarredona, L. Camuas-Mesa, T. Serrano-Gotarredona, J. A. Leero-Bardallo, and B. Linares-Barranco, "The stochastic I-pot: A circuit block for programming bias currents," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 54, no. 9, pp. 760–764, Sep. 2007.
[31] T. Serrano-Gotarredona, B. Linares-Barranco, and A. G. Andreou, "Voltage clamping current mirrors with 13-decades gain adjustment range suitable for low power MOS/bipolar current mode signal processing circuits," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, vol. 1. May/Jun. 1998, pp. 551–554.
[32] C. Igel and M. Husken, "Empirical evaluation of the improved Rprop learning algorithms," *Neurocomputing*, vol. 50, pp. 105–123, Jan. 2003.
[33] S. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, vol. 2. 1990, pp. 524–532.

Dzmitry Maliuk (M'13) received the Diploma degree in computer security from Belarusian State University, Minsk, Belarus, in 2007, and the M.S. and Ph.D. degrees in electrical engineering from Yale University, New Haven, CT, USA, in 2010 and 2013, respectively.

He spent two internships with the IBM T.J. Watson Research Center, Yorktown Heights, NY, USA, in 2011 and 2012, where he was with the Circuit Test and Diagnostics Technologies Group. He is currently a Quantitative Developer with Quantlab Financial, Houston, TX, USA. His current research interests include neuromorphic engineering and hardware/software implementation of learning systems. He holds a patent on scan chain latch design that improves testability of integrated circuits.

Yiorgos Makris (SM'08) received the Diploma degree from the University of Patras, Patras, Greece, in 1995, and the M.S. and Ph.D. degrees from the University of California at San Diego, La Jolla, CA, USA, in 1998 and 2001, respectively, all in computer engineering.

He joined the University of Texas at Dallas (UT Dallas), Richardson, TX, USA, after spending a decade with the faculty of Yale University, New Haven, CT, USA. He is currently a Professor of Electrical Engineering and leading the Trusted and RELiable Architectures Research Laboratory with UT Dallas. His current research interests include the applications of machine learning and statistical analysis in the development of trusted and reliable integrated circuits and systems, with particular emphasis on the analog/RF domain.

Prof. Makris served as the Program Chair of the IEEE VLSI Test Symposium, from 2013 to 2014, and the Test Technology Educational Program from 2010 to 2012. He also served as the Guest Editor of the IEEE TRANSACTIONS ON COMPUTERS and a Topic Coordinator and/or Program Committee Member for several IEEE and ACM conferences. His research activities have been supported by NSF, ARO, SRC, DARPA, Boeing, IBM, LSI, Intel, and Texas Instruments. He was a recipient of the 2006 Sheffield Distinguished Teaching Award and the Best Paper Award from the 2013 Design Automation and Test in Europe conference.