A Low Power Trainable Neuromorphic Integrated Circuit That Is Tolerant to Device Mismatch

Chetan Singh Thakur, *Member, IEEE*, Runchun Wang, *Member, IEEE*, Tara Julia Hamilton, *Member, IEEE*, Jonathan Tapson, *Member, IEEE*, and André van Schaik, *Fellow, IEEE*

***Abstract*—Random device mismatch that arises as a result of scaling of the CMOS (complementary metal-oxide semiconductor) technology into the deep submicrometer regime degrades the accuracy of analog circuits. Methods to combat this increase the complexity of design. We have developed a novel neuromorphic system called a trainable analog block (TAB), which exploits device mismatch as a means for random projections of the input to a higher dimensional space. The TAB framework is inspired by the principles of neural population coding operating in the biological nervous system. Three neuronal layers, namely input, hidden, and output, constitute the TAB framework, with the number of hidden layer neurons far exceeding the input layer neurons. Here, we present measurement results of the first prototype TAB chip built using a 65 nm process technology and show its learning capability for various regression tasks. Our TAB chip is tolerant to inherent randomness and variability arising due to the fabrication process. Additionally, we characterize each neuron and discuss the statistical variability of its tuning curve that arises due to random device mismatch, a desirable property for the learning capability of the TAB. We also discuss the effect of the number of hidden neurons and the resolution of output weights on the accuracy of the learning capability of the TAB. We show that the TAB is a low power system—the power dissipation in the TAB with 456 neuron blocks is 1.38 μW.**

***Index Terms*—Analog integrated circuit design, neural network hardware, neuromorphic engineering, stochastic electronics.**

I. Introduction

OVER time, electronic devices have witnessed a higher packing density of transistors in accordance with Moore's law [1]. Owing to this, computing devices have become smarter, faster, and more efficient. The increase in the number of transistors has been made possible due to a decrease in the minimum feature size, which has already reduced below 22 nm. However, keeping up with Moore's law has not been easy. In submicrometer technologies, factors such as minor variations of process, external unknown fields, minor layout changes, and leakage currents have large effects on the performance of analog circuits, making them difficult to design and, thus, creating significant challenges. These effects may be minimized by increasing device size; however, this increases the size of an integrated circuit (IC) and as a result increases its cost [2], [3]. Further, the failure of a few transistors may result in the failure of the entire chip, rendering it unusable. Thus, there is a trade-off between performance yield and costs in the submicrometer technology.

The brain is an incredible computational device that surpasses today's modern computers in various tasks such as vision and audition. Similar to the problems of transistor failure and device mismatch in an IC, the brain is faced with the problems of heterogeneity of neuronal responses to stimuli and neuronal cell death. The biological nervous system has been able to resolve these problems over the course of evolution, and provides an excellent model for IC implementation. Neuromorphic systems, inspired by neurobiological processing systems, offer an attractive alternative to conventional analog IC design technology in terms of power efficiency and computation using stochastic components [4]–[6].

In many regions of the brain, information is encoded by patterns of activity occurring over populations of neurons, a phenomenon referred to as population coding [7]. We have developed a novel neuromorphic system called a trainable analog block (TAB) that works in a similar manner by using a large pool of neurons for encoding the input, and linearly combining the neuronal responses to achieve decoding. The TAB chip architecture explicitly uses random device mismatch to its advantage, and is thus ideally suited for submicrometer technologies. The TAB attempts to incorporate the features of neurobiological systems, such as low power consumption, fault tolerance, and adaptive learning. Owing to adaptive learning, the designs will be portable across technologies and applications, eliminating the need for custom IC design for those functions that can be implemented with our TAB. We envisage that the TAB will contribute to a considerable speed-up in IC design by shortening the design cycle for analog circuits, and result in a drastic decrease in design costs. The TAB framework may be used to design systems that will employ hardware variability to achieve their engineering goal, thus qualifying as a design circuit paradigm for stochastic electronics [8]. The TAB circuits are effectively universal function approximators [9], thereby allowing for complex processing on a simple and repeatable substrate.

Other research groups have proposed systems that exploit device mismatch for computation in silicon [10]–[12]. Many recent papers [13]–[15] have used architectures similar to our design, inspired either by the neural engineering framework (NEF) [16] or the extreme learning machine (ELM) [17], and implemented using spiking neurons to process the spike

Manuscript received July 10, 2015; revised October 19, 2015; accepted December 16, 2015. Date of publication February 11, 2016; date of current version March 16, 2016. This paper was recommended by Associate Editor T. S. Gotarredona.

The authors are with Biomedical Engineering and Neuroscience, The MARCS Institute, Western Sydney University, Sydney, NSW 2751, Australia (e-mail: C.SinghThakur@uws.edu.au; mark.wang@uws.edu.au; T.Hamilton@uws.edu.au; J.Tapson@uws.edu.au; A.VanSchaik@uws.edu.au).

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/TCSI.2015.2512743

inputs. Chen *et al.* [13] have developed a machine learning co-processor (MLCP) that performs spike-based computation using ELM. The MLCP encodes the ELM algorithm for spike inputs in many stages, and the decoding is done separately on a microcontroller (MCU). The MLCP takes the spike inputs and then converts them into the analog domain using a counter and a DAC circuit. The output of the DAC is then sent as a current input to the analog neurons, which convert it back into spikes, and the resultant spikes are then counted using a digital counter with some extra logic blocks to implement a saturating nonlinearity (such as *sigmoid*). The output of the counter is then sent to an MCU for decoding of the ELM algorithm. In an ELM architecture, the number of hidden layer neurons is quite large—at least ten times of the input dimensions, to achieve good accuracy [17], [18]. However, the MLCP supports 128 inputs with only 128 hidden nodes. This system will thus have a low encoding capacity due to a low number of hidden nodes for such a high number of input dimensions.

Our TAB system is designed to be tolerant to device mismatch to create an analog signal processor, which does not use spikes. We have simply used a differential pair to generate the saturating nonlinearity. Our system does not require any extra circuits such as a digital counter, spiking neurons, or an input pre-processing unit for encoding of the input, unlike in the case of the MLCP [13]. The TAB also performs the decoding of the ELM framework on to the same chip—it does not require a separate MCU or an FPGA for the decoding logic. The encoding per neuron of a single input requires a very small area ($3.5\ \mu\text{m} \times 3.5\ \mu\text{m}$).

This paper is organized as follows: Section II explains the framework of the TAB. VLSI (very large scale integration) implementation of the TAB is described in Section III. The algorithm setup for offline learning in Section IV and the constrained algorithm in Section V. We present the measurements of the building blocks of our TAB implementation in Section VI and conclusions in Section VII.

II. Framework of TAB

The TAB framework draws inspiration from the phenomenon of neural population coding [19]. In population coding, biological neurons in several parts of the brain encode information in a collective and distributed manner using spike rates. The accuracy of information processing in the cortex depends on the quality of population coding, which in turn is affected by the heterogeneity of neuronal responses and the shape of neuronal tuning curves [21], [22]. The tuning curve of a neuron is a plot of its average firing rate as a function of the input stimulus. As examples of population coding, neurons in monkeys, cricket, barn owl, cats, bats, and rats encode the direction of arm movements [23], the direction of a wind stimulus [24], the direction of a sound stimulus [25], saccade direction [26], echo delay [27], and the position of the rat in its environment [28], respectively. The TAB framework is designed to use neuronal tuning curves instead of individual spikes. Further, we have used a heterogeneous population of neurons in the TAB architecture. Heterogeneity of the tuning curves of the neurons increase the encoding capacity of the network [29].

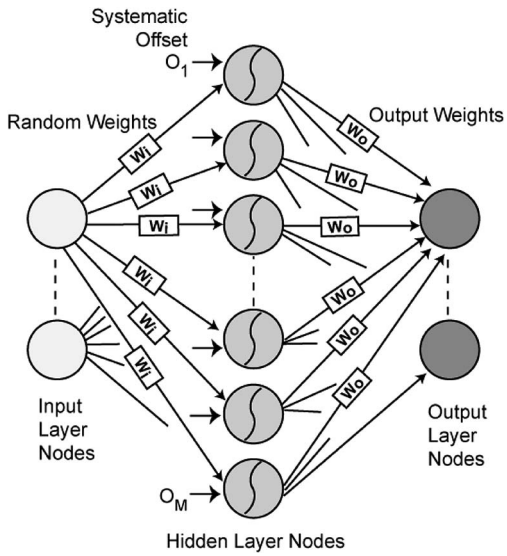

Fig. 1. **Architecture of the TAB framework**. The connections from the input layer neurons/nodes to the non-linear hidden neurons are via random weights and controllable offsets, O_1 to O_M. The hidden layer neurons are connected linearly to the outer layer neurons via trainable weights. The output neurons compute a linearly weighted sum of the hidden layer values. Adapted from [29].

In order to decode the response of a neuronal population, it is required to combine the firing rates of neurons into a population ensemble estimate. Generally, the tuning curve of each neuron contributes a basis function and the best estimate of the physical variables is computed from the sum of these functions weighted by the spike rate occurring in each neuron [30]. In our TAB system, we have used a similar approach to decode the stimulus.

Accurate encoding of an input occurs when a population of neurons covers the entire range of the input variable. This is best achieved if the neuronal tuning curves are equally spaced [31], and may be imposed in a neural system by encoding the defined physiological properties of neurons in each population. However, the resulting costs to the system are unreasonably high. Instead, randomly chosen parameters from the distribution are likely to perform an equally good approximation [31]. Recently, Caron *et al.* showed the existence of such randomness in the olfactory system, where inputs from the glomeruli to individual Kenyon cells lack any organization with respect to their odor tuning, anatomical features, or developmental origins [32]. In our TAB framework too, we have projected the input from the input layer neurons to the hidden layer neurons in a random manner. Random device mismatch cannot be avoided in smaller process technology, and instead we are using it in the TAB framework to encode the input variable.

The TAB is a feed-forward network of three neuronal layers, namely input, hidden, and output, structured on the linear solutions of higher dimensional interlayers (LSHDI) principle [33] (Fig. 1). The input layer neurons are connected to a larger number of hidden layer neurons via fixed random weights. Consequently, the inputs are projected randomly and transformed to a higher dimensional feature space by the nonlinear hidden layer of neurons. In effect, the input data points, which are not linearly separable in their current space, find a linear hyperplane in the higher dimensional space that approximates

a desired function as a regression solution, or represents a classification boundary for the input-output relationship. The output layer neurons derive a solution by computing only a linearly weighted sum of the hidden layer values. These linear weights are evaluated analytically by computing the product of the desired output values and the pseudoinverse of the hidden layer neurons output [34]. The TAB also employs systematic offset (O_i, Fig. 1) and preferred direction (PD), both of which help to diversify the tuning curves of the hidden layer neurons. Preferred direction (PD) implies that the activation function for one half of the hidden layer neurons may be $sigmoid$, and $-sigmoid$ for the other half, and this may be assigned deterministically or randomly. Previously proposed networks based on the LSHDI principle include the functional-link net computing (FLNN) by Pao *et al.* in 1992 [35], the extreme learning machine (ELM) by Huang *et al.* in 2006 [17], and the neural engineering framework (NEF) [16], which performs spike-based computation and is quite popular in the neuromorphic engineering community.

III. VLSI Implementation of TAB

In order to demonstrate that the TAB is effective in smaller process nodes that are normally prohibitive to analog design (at and beyond 65 nm) [36], we have designed the TAB prototype in a 65 nm technology. Further, a substantial section of the TAB was designed using minimum feature sizes so as to maximise mismatch among transistor parameters. Differences among the hidden layer neuronal responses can be enlarged by using an additional distinct systematic offset for each hidden layer neuron. As a proof of concept we have implemented a single input-single output (SISO) version, with a single input voltage and a single output current. We elucidate below the VLSI implementation of the major building blocks of the TAB, namely the hidden neuron and the output weight [29].

A. Hidden Neuron

Evidence has shown that neurons in a population respond to the same stimuli heterogeneously [37]. We use a differential pair to implement a simple neuron model in the TAB. The differential pair performs a nonlinear operation on its input, similar to the sigmoid tuning curve of the stereo V1 neurons in the cortex [38]. In Fig. 2, V_{in} (input voltage) and V_{ref} (constant reference voltage) are the gate voltages for the differential pair of transistors, M_1 and M_2, and influence the sharing of currents between them. These transistors operate in the weak-inversion regime, with the slope factor (n) ranging from 1.1 to 1.5 [39]. The currents in transistors M_1 and M_2 can be described as:

$$I_1 = I_b\left[exp\left(\frac{V_{in}}{nU_T}\right)\right] \Big/ \left[exp\left(\frac{V_{in}}{nU_T}\right) + exp\left(\frac{V_{ref}}{nU_T}\right)\right] \quad (1)$$

$$I_2 = I_b\left[exp\left(\frac{V_{ref}}{nU_T}\right)\right] \Big/ \left[exp\left(\frac{V_{in}}{nU_T}\right) + exp\left(\frac{V_{ref}}{nU_T}\right)\right] \quad (2)$$

where I_b is the maximum bias current, V_{in} is the ramp input voltage, V_{ref} is the constant input voltage, and U_T is the thermal voltage.

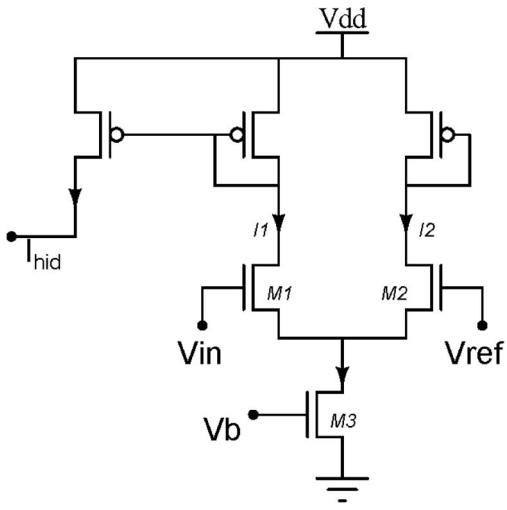

Fig. 2. **Hidden Neuron Block**. Schematic of the hidden neuron block that implements the $sigmoid$ nonlinear activation function for the TAB framework. Adapted from [29].

The current I_1 is copied to I_{hid} via a current mirror, which acts as a sigmoid tuning curve for a hidden neuron. The voltage at the M_3 transistor, V_b, sets the bias current ($\sim$few nanoamperes). In the TAB, each neuron has a distinct tuning curve depending on the process variations such as offset mismatch between the transistors in the differential pairs, bias current mismatch due to variability in M_3 and current mirror mismatch. Each neuron may receive a systematically different V_{ref} in the TAB, which is a failsafe method to achieve a distinct tuning curve for each neuron. This may be required in the case of insufficient random variations, which is likely in higher feature size process technology.

B. Output Weight

The output weight block connects the hidden layer and the output layer via linear weights. These are controlled by a 13-bit binary number, which is stored in digital flip-flops that regulate the amount of current flowing from the hidden layer neurons to the output layer neurons. We have implemented binary weighted connections using a splitter circuit (Fig. 3) [40]. The output from the hidden neuron block, I_{hid}, is the input current for the output weight block. I_{hid} is divided successively to form a geometrically-spaced series of smaller currents. A digital binary switch controls each current branch. A fixed fraction of the current is split off at each branch, and the remnant continues to the later branches. There are a total of N stages in the splitter circuit. The current at the kth stage is given by $(I_{hid}/2^k)$. The master bias voltage V_{gbias} is the reference voltage for the p-FET gates in the splitter [40]. As shown in Fig. 3, two transistor switches in the lower half of the circuit route the branch current to either useful current, I_{good}, or to current that goes to ground, I_{dump}. I_{good} is mirrored to generate a current, I_{out}, which is further routed to currents I_{pos} (positive current) or I_{neg} (negative current), as determined by the $signW$ signal. The $signW$ signal, stored in flip-flop, indicates the polarity of the output weight connected between the hidden neuron and the output neuron. In a hidden neuron, the 13-bit output

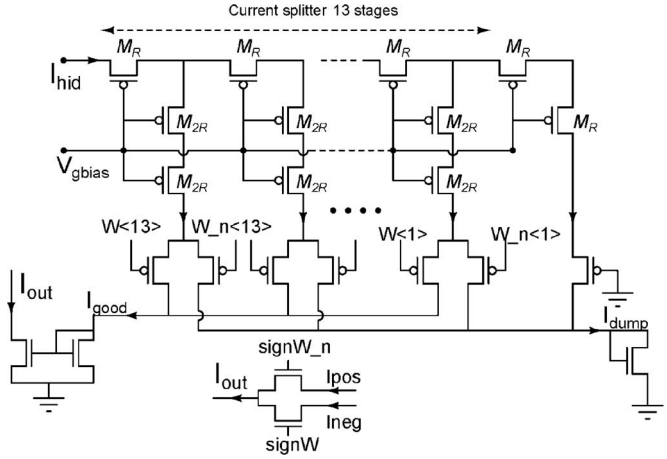

Fig. 3. **Output Weight Block**. Schematic of the output weight block, comprising a splitter circuit wherein M_R and the two M_{2R} transistors form an R2R network, which gets repeated 13 times in the block. The octave splitter is terminated with a single M_R transistor. Adapted from [29].

weights and $signW$ are connected as shift registers. Internal shift registers of all the hidden neurons are connected serially as a long chain at the top level. The shift registers are loaded with off-chip calculated weights and are used to regulate the current in the output weight block.

This TAB chip was designed for a single input and a single output configuration with 456 neuron blocks, the number of neuron blocks being constrained by the chip area. Each neuron block integrates a hidden neuron, an output weight block and a 13-bit shift register, which is used for loading the learned weights. At a particular time, each neuron block receives the same input voltage, V_{in}, which is weighted by a random weight and a random offset arising due to process variations. Additionally, each neuron may exhibit a distinct reference voltage, V_{ref}, in the differential pairs of the hidden neuron. This leads to different differential voltages for each neuron block, and as a result different currents, I_{hid}, are generated for each block. Each V_{ref} is tapped from a long poly-silicon wire, the end points of which are connected to the top level voltage pins, V_{ref1} and V_{ref2}. The poly-silicon wire behaves as a long distributed resistor element that acts as a voltage divider and generates different reference voltages, V_{ref}, for each neuron block. For each new input, the hidden neuron block calculates I_{hid}, which passes to the output weight block. In the output weight block, I_{good} is mirrored to make I_{out} (Fig. 3), which is further routed to currents I_{outP} (positive current) or I_{outN} (negative current), as determined by $signW$ signal. I_{outP} and I_{outN} currents of each neuron block are connected globally to each other, and they are summed up to provide the final current that is the final output of the TAB. We use an off-chip current-to-voltage converter and amplifier circuits to convert the final output current into voltage for ease of measurement.

IV. Learning Set-Up

We now discuss the algorithm used for the offline learning of the TAB IC. In the TAB framework, learning is achieved by computing output weights to train the system for desired regression/classification tasks. The output weights (between the large hidden layer and the linear output neurons) are estimated analytically by calculating the product of the pseudoinverse of the hidden layer activations with the target outputs [18]. We briefly summarize the learning setup below. More details can be found in [29].

Let us consider a three-layer feed-forward TAB network with L number of hidden neurons. Let $G(.,.,.,.)$ be a real-valued function so that $G(w_i^{(1)}, b_i^{(1)}, o_i^{(1)}, x, d_i^{(1)})$ is the output of the ith hidden neuron, corresponding to the input vector $x \in \mathbb{R}^m$. The random input weight vector $w_i^{(1)} = (w_{i1}^{(1)}, \ldots, w_{im}^{(1)})$, where $w_{iS}^{(1)}$ is the weight of the connection between the ith hidden neuron and sth neuron of the input layer, with random bias $b_i^{(1)} \in \mathbb{R}$, both arising due to random mismatch of the transistors. The random input weight, $w^{(1)}$, varies according to a log-normal distribution (Fig. 6(c)) due to the exponential relationship between the voltage and the current of a transistor, while the random bias, $b_i^{(1)}$, exhibits a Gaussian distribution (Fig. 6(b)). Preferred direction (PD), denoted as $d_i^{(1)} \in [-1, 1]$ is added to incorporate flexibility of changing the direction of the tuning curve either towards positive or negative values. PD assignment to the hidden neurons could be chosen either randomly or deterministically. Systematic offset $o_i^{(1)} \in \mathbb{R}$ is added to ensure that each neuron exhibits a distinct tuning curve, which is an essential requirement for learning in the LSHDI framework [29]. The output vector $y \in \mathbb{R}^k$ can be written as:

$$y = \Sigma_{i=1}^{L} w_i^{(2)} G\left(w_i^{(1)}, b_i^{(1)}, o_i^{(1)}, x, d_i^{(1)}\right) \tag{3}$$

where $w_i^{(2)} = (w_{1i}^{(2)}, \ldots, w_{ki}^{(2)}) \in \mathbb{R}^k$ is the weight vector where $w_{ji}^{(2)} \in \mathbb{R}$ is the weight connecting the ith hidden neuron with the jth neuron of the output layer. Here, $G(.,.,.)$ takes the following form:

$$G\left(w_i^{(1)}, b_i^{(1)}, o_i^{(1)}, d_i^{(1)}, x\right) = \left(g\left(w_i^{(1)}.x + b_i^{(1)} + o_i^{(1)}\right)\right).d_i^{(1)} \tag{4}$$

where, g: $\mathbb{R} \rightarrow \mathbb{R}$ is the activation function.

The output weight vector, $w_i^{(2)}$, can be written in matrix form for all the hidden neurons as $W^{(2)}$. The least squares solution of the output weight matrix, $W^{(2)}$, as described in [17] is:

$$W^{(2)} = H^{+}Y \tag{5}$$

where, H^+ is the Moore-Penrose generalized pseudoinverse of the matrix H. The matrix H is the output of all the hidden neurons (G) for all the input training data samples. The matrix Y is the collection of the output vectors for the training dataset.

V. Constrained Algorithm

The pseudoinverse algorithm is a quick and easy method to estimate the parameters of a linear regression problem with a quadratic cost function. It is an unconstrained algorithm that can compute output weights in any range, and thus is not suited for hardware implementation of the TAB system. In a pseudoinverse operation, some of the weights may go to large values. As

a result, the spread of the weights becomes very large, which reduces the dynamic range of the output weights and requires a higher number of bits. In order to keep the weights in the range $[-1, 1]$ and to minimize the spread, we have used a constraint-based algorithm for the weight calculations. As shown in Fig. 3, the output weight block acts as a current divider, implying that the ratio of the output to the input current is always less than 1. Thus, we used the method of least squares to calculate the output weights, with additional constraints to calculate weights in the range of $(-1, 1)$. As mentioned in the previous section, we collect the tuning curves of all the hidden neurons for all the training inputs. We created a function, $fcost\_grad$, which calculates the squared error cost, $fcost$ (6), and the gradient, $fgrad$ (7), for a given training set. Our purpose is to estimate $W^{(2)}$ while minimising the cost function $fcost$. The function $fcost\_grad$ is passed as an argument along with the weight constraints to the $fmincon$ optimisation solver of MATLAB, which uses sequential quadratic programming [41]. We define the cost function and gradient of the linear regression problem as:

$$fcost\left(W^{(2)}\right) = \left(\frac{1}{2C}\right)\Sigma_{i=1}^{C}\Sigma_{i=1}^{L} \times \left(w_i^{(2)} g\left(w_i^{(1)}.x + b_i^{(1)} + o_i^{(1)}\right).d_i^{(1)} - y\right)^2 \quad (6)$$

$$fgrad\left(W^{(2)}\right) = \left(\frac{1}{C}\right)\Sigma_{i=1}^{C}\Sigma_{i=1}^{L} \times \left(w_i^{(2)} g\left(w_i^{(1)}.x + b_i^{(1)} + o_i^{(1)}\right).d_i^{(1)} - y\right).x. \quad (7)$$

The function $fmincon$ is an optimization solver that finds the minimum of a constrained function. For linear regression, we want to minimize the cost function $fcost$ with parameters $W^{(2)}$ using a given fixed training set.

VI. Results

A. TAB Learning in Software Simulation

Here, we show the software simulation results of the TAB network using a single input and a single output configuration. We built the TAB network with 50 hidden neurons with 13-bits output weight, and tested its ability to learn different functions such as $sine$ and $square$, using constrained offline learning. In the TAB, we used the $sigmoid$ function as the nonlinear activation function (tuning curve) for each hidden neuron. We used the offline learning setup as mentioned in Section IV, and the $fmincon$ solver from MATLAB to calculate the output weights externally. We presented the training data to the network, each training pair containing an input, x and an output, y. Each input training value is randomly and systematically offset and multiplied by random weights for each hidden neuron, and is projected randomly to 50 hidden neurons in this manner. For every input data point, we collected the response of the hidden neurons and created a matrix H, as shown in (5). We used the constrained algorithm to calculate the output weights. In the testing phase, we presented the test input to the network and obtained a predicted output. We show that the TAB system is able to learn the various functions successfully (Fig. 4). We normalised our input range between -1 to $+1$ V to simplify the function argument. From the simulation results, we can see that the learned function closely matches with the target function. Fig. 4(a) and (b) show the results from training the network for the sine and square functions, with a test accuracy of 0.67% and 0.53% with respect to the RMS of the target signal, respectively.

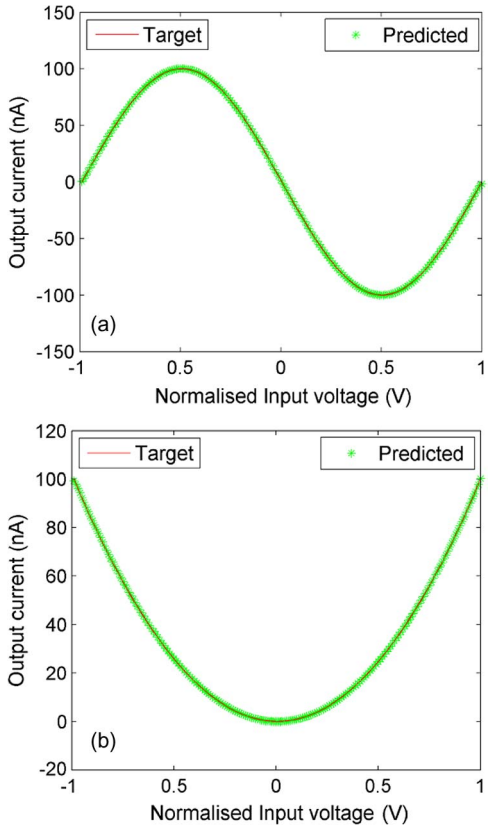

Fig. 4. Learning curves for the regression functions. (a) sine. (b) square. The red curve represents the target function, and the green curve represents the learnt function.

B. Analysis of Hidden Nodes and Output Weights

The size of a circuit grows linearly with the number of hidden nodes and the number of bits in the output weights. We first examined the number of hidden nodes and bits needed in our system using software simulations. The optimum number of hidden nodes and bits was found to vary with the desired target function, with some functions, such as $sinc(x)$, proving much more difficult to learn than others, such as x^2. Fig. 5(a) shows the RMS error between the target function and the learned function as a function of the number of hidden neurons, for both $y = sinc(6\pi x)$, and $y = x^2$. The mean RMS error was calculated for 10 simulations with different random weights from the input to the hidden nodes. In these simulations, we used 13-bits for the output weights. It can be seen that the standard deviation is quite high for low number of hidden nodes, but for a larger number of hidden nodes the result becomes largely independent of the random weights. It is also clear that the $y = sinc(6\pi x)$ function needs significantly more hidden nodes than $y = x^2$ to be learned accurately.

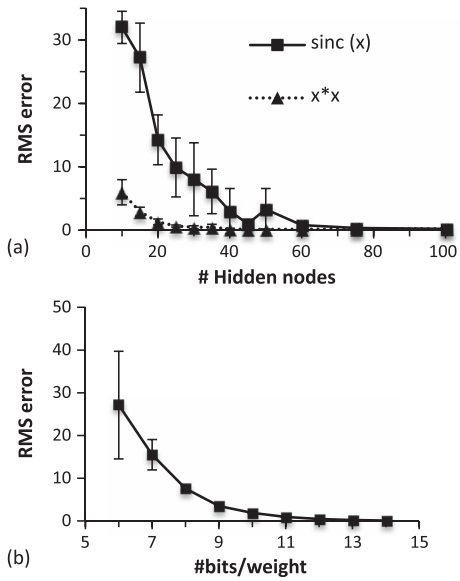

Fig. 5. (a) Plot of the RMS error between the target function and the learned function versus the number of hidden nodes. The error bars show the standard deviation. (b) The RMS error versus the number of bits/weight used for the output weights for the function $y = sinc(6\pi x)$. The error bars show the variance.

TABLE I
FEATURES OF THE TAB SISO CHIP

Technology	TSMC 65nm
Supply voltage	1.2 V
Total number of neuron blocks	456
Random variations in current amplitude (Fig.6C)	ln $\mathcal{N}(\mu = 0, \sigma = 0.5962)$
Random offset (in %) (Fig.6B)	$\mathcal{N}(\mu = -2.422, \sigma = 9.999)$
Time constant of the TAB	2.3 µS
Power dissipation	
Power per neuron block	3 nW
Total power without gain circuit	1.38 µW
Total power	16.56 µW
Area of each circuit per block (µm)	
Hidden neuron	3.5 x 3.5
Output weight circuit	15 x 6.5
Shift registers (13bits weight + Sign bit)	36 x 4

The RMS error as a function of the number of bits used for the output weights, for $y = sinc(6\pi x)$ for a network with 100 hidden nodes is shown in Fig. 5(b). Again, these are the results of 10 simulations with different random weights from the input to the hidden nodes. Clearly, the higher the resolution in the output weights, the closer the digital weights can approach the weights found by the offline learning (which are real-valued numbers) and the better the function can be implemented. From about 8-bits onwards, the standard deviation is negligibly small, and the RMS error becomes almost totally independent of the random weights. Increasing the number of bits per weight is a matter of diminishing returns, and 11-bits seem sufficient, even to learn this difficult function.

C. Neuron Characterization in the TAB IC

A TAB prototype was fabricated in the 65 nm process technology with 456 neuron blocks. Table I summarizes the system level features of the TAB chip. In the TAB, we have used a current gain circuit (100×), consisting of two sets of current mirrors each with a gain of 10, to amplify the final output current

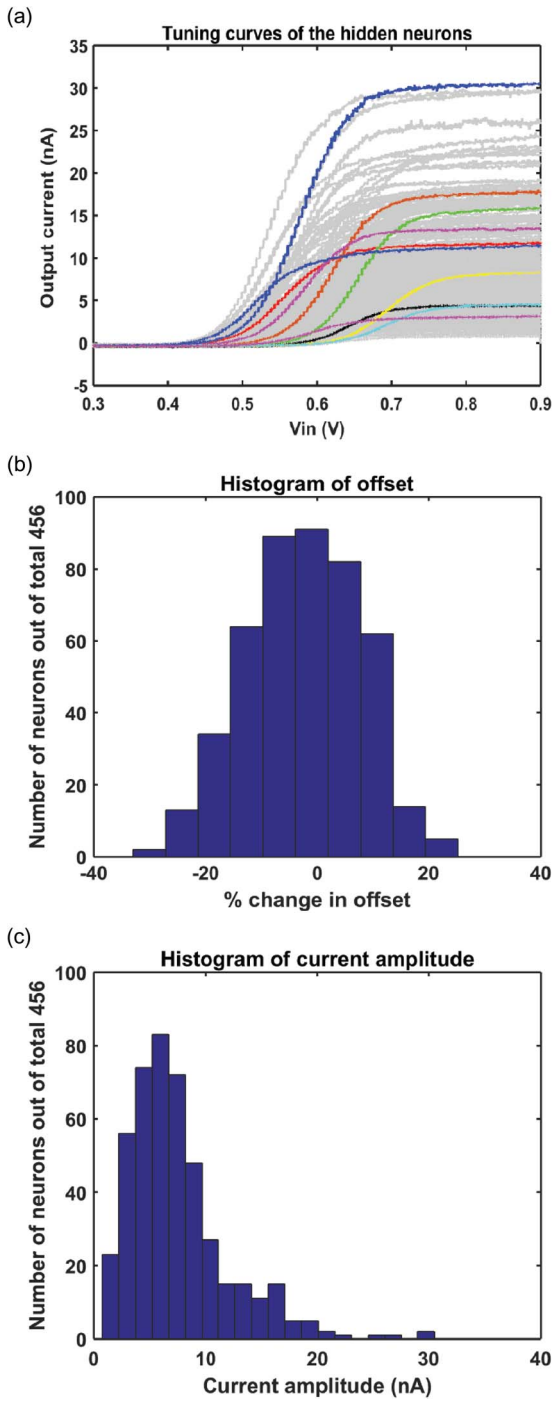

Fig. 6. Hidden neuron characterisation results for a total of 456 neurons. (a) *sigmoid* tuning curves of all the neurons show the variation in offset and the minimum current value. (b) Random offset present in the chip across all the 456 neurons. (c) Variation in the current amplitude of the tuning curves of all the 456 neurons exhibits a log-normal distribution, as expected.

for measurement purposes, which may not be required in actual applications. The actual power dissipation of the TAB core is much smaller than that of the output stage. Thus in Table I we report the power consumption with and without the output gain stage. We have also characterized the speed of the TAB by the step response. The dominant pole of the system is dominated by the first output current gain (10×) circuit. The measured time constant of the TAB for the step response is 2.3 μS. The

actual time constant of the TAB without the output current gain mirrors would be much smaller than this, but we cannot measure it in the current system.

We characterized the tuning curve of each neuron to analyze the mismatch and differences between the tuning curves of the hidden neurons without any systematic offset by connecting the V_{ref} node (Fig. 2) of each hidden neuron to the same voltage. Learning is better if there is a high diversity between the tuning curves of neurons [29], [31]. As shown in Fig. 6, we obtained heterogeneity in the neuronal tuning curves due to random device mismatch and process variations in the fabrication. Each neuron block contains a hidden neuron and output weight block (Section III) with shift registers. The shift registers of all the hidden neurons are connected serially as a long chain. Due to the large number of hidden neurons, it is not feasible to have dedicated output ports to probe the output current for each hidden neuron. Therefore, the output current of each hidden neuron is probed indirectly through the "OUT" port of the IC one-by-one. The output weight block behaves as a current splitter, i.e., if all the bits of a weight are one, the output current of the output weight block would be almost the same as its input current, and if all the bits of the weight are zeros, the output weight block will produce nearly zero output. We characterized each neuron sequentially. We loaded the output weight between the hidden neuron of interest and the "OUT" port to be 13'h1FFF, and all other weights between the remaining hidden neurons and OUT port to be 13'h0000. The MSB (most significant bit, 13th bit here) represents the sign of the output weight, where "1" represents negative weight. Then, we provided the ramp input to the TAB and measured the current at the output port. We collected the tuning curves of all the neurons (Fig. 6(a)) and plotted the statistical variation in voltage offset and current amplitude of the tuning curves (Fig. 6(b) and (c)). We have normalized each current by dividing it by the geometric mean of currents from all the hidden neurons, and then modelled it as a log-normal distribution with location (μ) and scale (σ) parameters (Table I). In order to calculate the random voltage offset (Fig. 6(b)), we have used the same V_{ref} for all the hidden neurons. Ideally, when the input for a neuron is equal to V_{ref}, the neuron reaches half of its output current and the slope of the tuning curve is maximal at this value. However, the actual maximal slope is typically obtained at a slightly different input voltage. The offset shown in Fig. 6(b) is taken as the difference between this input voltage and V_{ref}, and expressed as a percentage of V_{ref}.

D. Learning Capability of the TAB IC

In this section, we show the ability of the TAB IC to exploit device mismatch. To learn a particular mapping from input to output, we calculate the weights externally. In order to do this, the output of hidden neurons for a given input sample needs to be probed. For L hidden neurons and C distinct input training values $(x_1, \ldots, x_C)$, the hidden layer activation matrix H_{CxL} is obtained, and the value of weights is calculated using the offline learning setup as described in Section IV. We collected the tuning curves of all the hidden neurons sequentially by measuring the current at the output port, as described above in Section VI-C. After collecting all the data, we calculated the output weights using the constrained algorithm. We trained the TAB IC for various regression tasks such as $sine$, $cube$, $square$ functions as shown in Fig. 7. The output for the functions matched with the desired output with an RMS error of 9.4%, 4.5%, and 5.6% for the $sine$, $cube$, and $square$ functions, respectively, with respect to the RMS of the target signal. The error here is larger as compared to that obtained in the software simulations because the response of the output weight

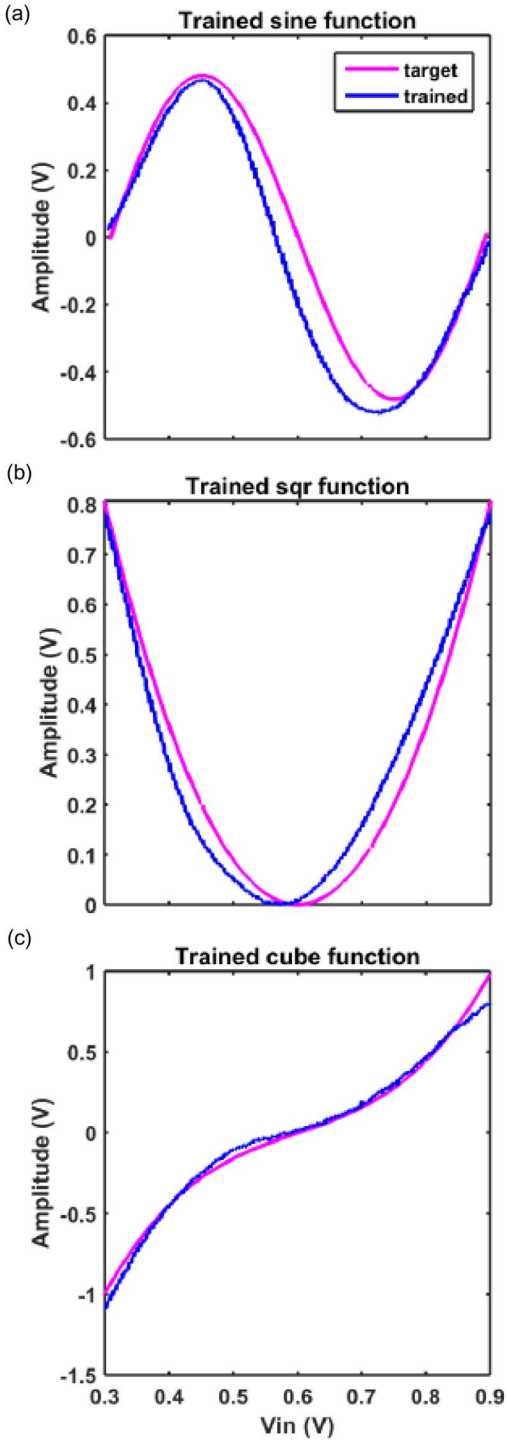

Fig. 7. Learning in the TAB chip for the functions: (a) `sine`, (b) square, (c) *cube*. Top and bottom graph in each figure represent input to the TAB and trained output from the TAB.

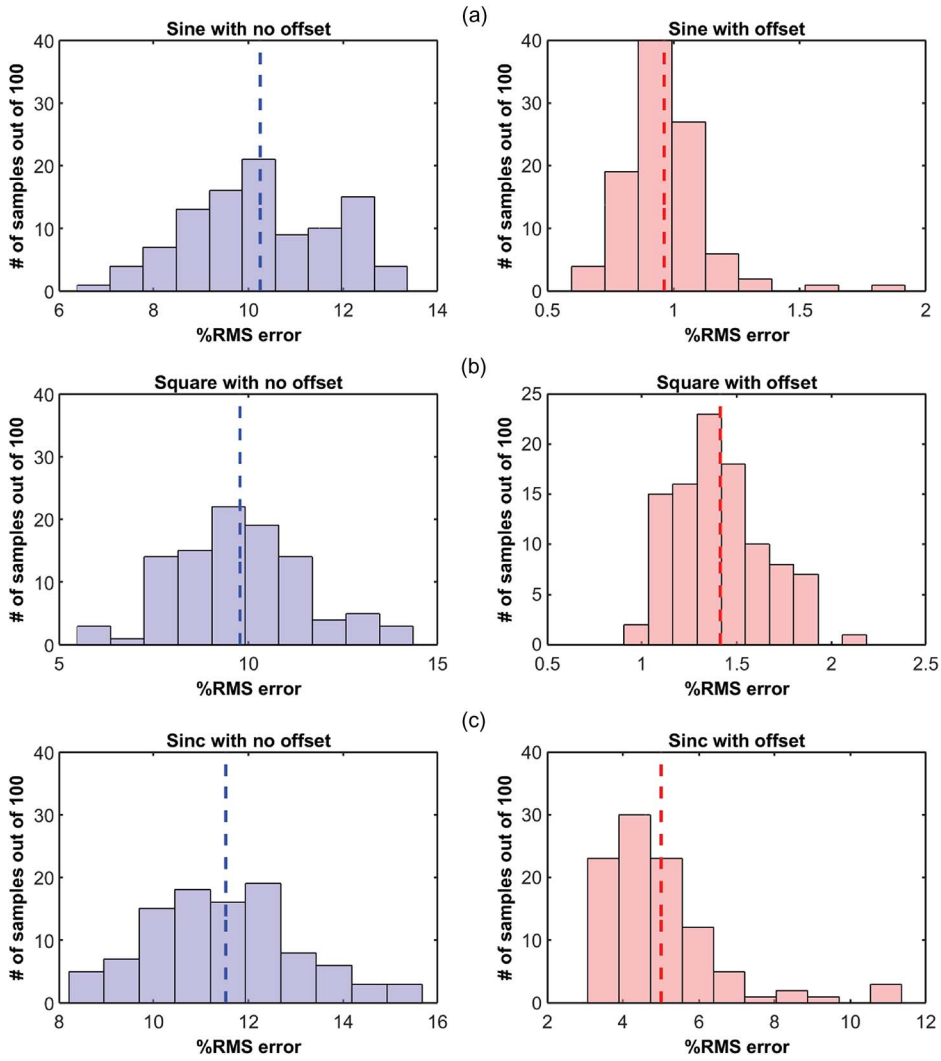

Fig. 8. Comparison of the regression error for the functions without (left) and with (right) systematic offset: (a) $sine$, (b) $square$, (c) $sinc$.

block has some nonlinearity [29]. This nonlinearity may be compensated by characterizing each output weight block response and including these as constraints when solving for the output weights. However, this would be very cumbersome to implement for each IC. In order to overcome this issue, we have implemented an online learning rule in the chip, which automatically adapts to the mismatch of each output weight block. This implementation is described in [42].

E. Encoding Capacity of the TAB IC Due to Random Mismatch (RM) With and Without Offset

The variations in the tuning curves, as shown in Fig. 6, are crucial in learning any regression or classification task. It is evident from Fig. 6 that the heterogeneity in the tuning curves is not sufficient to cover the entire range of the input. For instance, below 0.4 V or above 0.8 V the output of each neuron hardly changes. Thus, it is difficult to learn a target function in those input ranges when just exploiting the inherent device mismatch. For this reason, we introduce a systematic offset to the neurons, so that over the full input range some neurons' output is always changing. Fig. 8 shows the improvement in accuracy resulting from this by comparing the error of the TAB using 30 hidden neurons for a few regression tasks for both configurations—i) using the inherent device random mismatch only, and ii) using an additional systematic offset with the device mismatch.

In Fig. 8, we show the distribution of the RMS errors for 100 different combinations of 30 hidden neurons chosen randomly out of a total 456 neurons without any systematic offset $(blue)$, and with systematic offset $(pink)$, for $y = sin(x)$, $y = x^2$ and $y = sinc(4x)$ functions. The results show that some combinations of tuning curves of hidden neurons encode better than others. It is also evident that the regression error is much lower when we use systematic offset than without.

It can be concluded that the TAB chip has a large encoding capacity as a result of the diversity of neuronal tuning curves, which in turn arises from random device mismatch and systematic offset. The TAB is able to perform the regression tasks using only device mismatch without using systematic offsets, i.e., exploiting just the inherent mismatch, but its performance is suboptimal in this mode. A better solution is obtained by using systematic offset of the tuning curves. In this configuration the TAB is tolerant to significant amounts of device mismatch, allowing it to operate on smaller IC technologies. The TAB is thus a unique framework that overcomes the limitations of random device mismatch and employs them to its advantage.

VII. Conclusion

In this paper, we have presented a novel neuromorphic TAB architecture inspired from the phenomenon of population coding present in the nervous system, which is tolerant to random device mismatch (fixed-pattern mismatch) and variability in the fabrication process. We have presented measurement results of our first prototype IC designed in 65 nm for a single input and a single output configuration of the TAB system. The TAB also incorporates systematic offset as a failsafe method to spread the tuning curves of the neurons. Systematic offset may be required when there is insufficient random variation among transistors to produce a distinct tuning curve for each neuron, which is highly likely in higher feature size process technology. We have also shown the learning capability of the TAB system for various regression tasks.

The implementation of our framework in the analog domain offers various advantages over digital implementations [43]–[47]. For example, addition in an analog circuit is computed simply by connecting the common output line to sum the currents and multiplication in the TAB is implemented using output weight circuits with a few transistors (Fig. 3), while a digital implementation requires several thousands of transistors for the same computations. Although the output weight circuit is not linear, this can be compensated by an on-chip learning rule, which is described in our other work [42]. Our system offers very low power consumption in the range of a few μW (Table I) with a very high encoding capacity. Also, the analog implementation of the TAB is easy to interface with the real-world sensors, which by their nature, are analog, as compared to digital implementations, which always require an analog-to-digital converter (ADC). Moreover, the implementation of the tuning curves in digital requires much higher transistor counts [48], [49]. As compared to other analog implementations [50]–[52], the TAB framework uses random input weights and thus does not need any additional input weight circuits. The TAB can be used as a low power analog signal processor, using very small and simple circuits, which can be used to learn any arbitrary functions and perform classification tasks. Unlike other spike based implementations [13]–[15], the TAB performs all the computation in the analog domain using the digital weights, which saves extra conversion circuits.

The TAB is inspired by neural population coding which is very robust in the face of damage of a few neurons, and does not have a disastrous effect on the encoded representation as the information is encoded across many neurons. The TAB system is designed using neuromorphic principles based on stochastic computation. We envisage the TAB to overcome the limitations of analog IC design at low process nodes and drive the integration process with digital blocks in the same circuit and process node. This may find applications in analog/digital converters (ADCs) and digital-to-analog converters (DACs) for submicrometer mixed signal chips such as those used in mobile processor chips and data acquisition chips.

The main significance of our TAB is that it solves the problem that increased device mismatch in modern IC manufacturing technologies causes for analog design. It works better with a considerable amount of device mismatch, and accurate mappings from input values to output values can be obtained without needing to engineer the effect of device mismatch out of the circuit, as is done currently. A further significant advantage of this approach is that the same TAB may be reused for many different purposes once manufactured, and the same architecture may be used in different manufacturing technologies. This will lead to a significantly reduced design cycle for analog circuits, with an associated reduction in design cost, and a speed-up of technological progress. The TAB may also be (re-)trained "on the job." This could be a major advantage in systems where the input-output mapping of a TAB needs to be changed because of changes in the system. An example would be in a communication system where a TAB is used as a filter to process the analog signal before digitization, in which the communication channel changes over time. The TAB can be re-trained with the communication channel in the loop to compensate for the changes. Furthermore, as the TAB framework desires large random mismatch among devices, and as mismatch is inversely proportional to device area, it could lead to significant reductions in chip area and manufacturing costs. Also, the failure of a few neurons would not affect the performance of the TAB as information is encoded into a large ensemble of neurons. Recently, we have built a multi-input TAB chip, where inputs are randomly weighted and combined using a follower circuit, as described by Vittoz [5]. Future work will aim to test and verify multi-input TAB chip for classification tasks.

References

[1] G. E. Moore, "Cramming more components onto integrated circuits," *IEEE Solid-State Circuits Newsl.*, vol. 20, no. 3, pp. 33–35, Sep. 2006, reprinted from *Electronics*, vol. 38, no. 8, Apr. 19, 1965, pp. 114 ff.

[2] P. R. Kinget, "Device mismatch and tradeoffs in the design of analog circuits," *IEEE J. Solid-State Circuits*, vol. 40, no. 6, pp. 1212–1224, Jun. 2005.

[3] A. Marshall, "Mismatch and noise in modern IC processes," *Synth. Lect. Digit. Circuits Syst.*, vol. 4, no. 1, pp. 1–140, Jan. 2009.

[4] C. A. Mead, *Analog VLSI and Neural Systems*,1st ed. Reading, MA, USA: Addison-Wesley, 1989.

[5] E. A. Vittoz, "Analog VLSI implementation of neural networks," in *Proc. IEEE Int. Symp. Circuits Syst.*, 1990, pp. 2524–2527.

[6] G. Indiveri, B. Linares-Barranco, T. J. Hamilton, A. Van Schaik, R. Etienne-Cummings, T. Delbruck, S.-C. Liu, P. Dudek, P. Häfliger, S. Renaud, J. Schemmel, G. Cauwenberghs, J. Arthur, K. Hynna, F. Folowosele, S. Saighi, T. Serrano-Gotarredona, J. Wijekoon, Y. Wang, and K. Boahen, "Neuromorphic silicon neuron circuits," *Front. Neurosci.*, vol. 5, pp. 1–23, May 2011.

[7] T. J. Sejnowski, *Neural Populations Revealed*, vol. 332, pp. 21218, Mar. 1988.

[8] T. J. Hamilton, S. Afshar, A. van Schaik, and J. Tapson, "Stochastic electronics: A neuro-inspired design paradigm for integrated circuits," *Proc. IEEE*, vol. 102, no. 5, pp. 843–859, May 2014.

[9] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Netw.*, vol. 4, pp. 251–257, 1991.

[10] K. Cameron, V. Boonsobhak, A. Murray, and D. Renshaw, "Spike timing dependent plasticity (STDP) can ameliorate process variations in neuromorphic VLSI," *IEEE Trans. Neural Netw.*, vol. 16, no. 6, pp. 1626–1637, 2005.

[11] A. Basu, S. Shuo, H. Zhou, M. Hiot Lim, and G.-B. Huang, "Silicon spiking neurons for hardware implementation of extreme learning machines," *Neurocomputing*, vol. 102, pp. 125–134, Feb. 2013.

[12] C. Merkel and D. Kudithipudi, "Neuromemristive extreme learning machines for pattern classification," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, pp. 77–82.

[13] Y. Chen, E. Yao, and A. Basu, "A 128 channel 290 GMACs/W machine learning based co-processor for intention decoding in brain machine interfaces," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2015, pp. 3004–3007.

[14] O. Richter, F. Reinhart, S. Nease, J. J. Steil, and E. Chicca, "Device mismatch in a neuromorphic system implements random features for regression," in *Biomed. Circuits Syst. Conf.*, 2015, in press.
[15] F. Corradi, C. Eliasmith, and G. Indiveri, "Mapping arbitrary mathematical functions and dynamical systems to neuromorphic VLSI circuits for spike-based neural computation," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2014, pp. 269–272.
[16] T. C. Stewart and C. Eliasmith, "Large-scale synthesis of functional spiking neural circuits," *Proc. IEEE*, vol. 102, no. 5, pp. 881–898, May 2014.
[17] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, no. 1–3, pp. 489–501, Dec. 2006.
[18] J. Tapson and A. van Schaik, "Learning the pseudoinverse solution to network weights," *Neural Netw.*, vol. 45, pp. 94–100, Sep. 2013.
[19] A. Pouget, P. Dayan, and R. S. Zemel, "Inference and computation with population codes," *Annu. Rev. Neurosci.*, vol. 26, pp. 381–410, 2003.
[20] A. S. Ecker, P. Berens, A. S. Tolias, and M. Bethge, "The effect of noise correlations in populations of diversely tuned neurons," *J. Neurosci.*, vol. 31, no. 40, pp. 14272–14283, 2011.
[21] M. I. Chelaru and V. Dragoi, "Efficient coding in heterogeneous neuronal populations," *Proc. Natl. Acad. Sci. U.S.A.*, vol. 105, no. 42, pp. 16344–16349, Oct. 2008.
[22] M. Rigotti, O. Barak, M. R. Warden, X.-J. Wang, N. D. Daw, E. K. Miller, and S. Fusi, "The importance of mixed selectivity in complex cognitive tasks," *Nature*, vol. 497, no. 7451, pp. 585–590, May 2013.
[23] A. P. Georgopoulos, A. B. Schwartz, and R. E. Kettner, "Neuronal population coding of movement direction," *Science*, vol. 233, pp. 1416–1419, 1986.
[24] J. P. Bacon and R. K. Murphey, "Receptive fields of cricket giant interneurones are related to their dendritic structure," *J. Physiol.*, vol. 352, pp. 601–623, 1984.
[25] E. I. Knudsen and M. Konishi, "A neural map of auditory space in the owl," *Science*, vol. 200, pp. 795–797, 1978.
[26] J. A. Van Gisbergen, A. J. Van Opstal, and A. A. Tax, "Collicular ensemble coding of saccades based on vector summation," *Neuroscience*, vol. 21, pp. 541–555, 1987.
[27] W. E. O'Neill and N. Suga, "Target range-sensitive neurons in the auditory cortex of the moustache bat," *Science*, vol. 203, pp. 69–73, 1979.
[28] J. O'Keefe and J. Dostrovsky, "The hippocampus as a spatial map. Preliminary evidence from unit activity in the freely-moving rat," *Brain Res.*, vol. 34, pp. 171–175, 1971.
[29] C. S. Thakur, T. J. Hamilton, R. Wang, J. Tapson, and A. van Schaik, "A neuromorphic hardware framework based on population coding," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, 2015, pp. 1–8.
[30] E. Salinas and L. F. Abbott, "Vector reconstruction from firing rates," *J. Comput. Neurosci.*, vol. 1, no. 1–2, pp. 89–107, Jun. 1994.
[31] C. Eliasmith and C. H. Anderson, *Neural Engineering (Computational Neuroscience Series): Computational, Representation, Dynamics in Neurobiological Systems*. Cambridge, MA, USA: MIT Press, 2002.
[32] S. J. C. Caron, V. Ruta, L. F. Abbott, and R. Axel, "Random convergence of olfactory inputs in the Drosophila mushroom body," *Nature*, vol. 497, no. 7447, pp. 113–117, May 2013.
[33] J. C. Tapson, G. K. Cohen, S. Afshar, K. M. Stiefel, Y. Buskila, R. M. Wang, T. J. Hamilton, and A. van Schaik, "Synthesis of neural networks for spatio-temporal spike pattern recognition and processing," *Front. Neurosci.*, vol. 7, pp. 153, Jan. 2013.
[34] J. Tapson and A. van Schaik, "Learning the pseudoinverse solution to network weights," *Neural Netw.*, vol. 45, pp. 94–100, Sep. 2013.
[35] Y.-H. Pao and Y. Takefuji, "Functional-link net computing: Theory, system architecture, functionalities," *Computer*, vol. 25, no. 5, pp. 76–79, May 1992.
[36] A. B. Kahng, "Design challenges at 65 nm and beyond," in *Proc. Design, Autom., Test Eur. Conf. Exhib.*, 2007, pp. 1–2.
[37] L. F. Abbott, "Theoretical neuroscience rising," *Neuron*, vol. 60, no. 3, pp. 489–495, Nov. 2008.
[38] D. G. Albrecht and D. B. Hamilton, "Striate cortex of monkey and cat: Contrast response function," *J. Neurophysiol.*, vol. 48, no. 1, pp. 217–237, 1982.
[39] E. A. Vittoz, "Weak inversion for ultra low-power and very low-voltage circuits," in *Proc. IEEE Asian Solid-State Circuits Conf.*, 2009, pp. 129–132.
[40] T. Delbrück and A. Van Schaik, "Bias current generators with wide dynamic range," *Analog Integr. Circuits Signal Process.*, vol. 43, no. 3, pp. 247–268, Jun. 2005.
[41] M. C. Biggs, "Constrained minimization using recursive quadratic programming," in *Towards Global Optimization*, L. C. W. Dixon, and G. P. Szergo, Eds. Amsterdam, The Netherlands: North-Holland, 1975, pp. 341–349.
[42] C. S. Thakur, R. Wang, S. Afshar, T. J. Hamilton, J. Tapson, and A. van Schaik, "An online learning algorithm for neuromorphic hardware implementation," *arXiv:1505.02495*, p. 8, May 2015.
[43] R. Gadea, J. Cerda, F. Ballester, and A. Macholi, "Artificial neural network implementation on a single FPGA of a pipelined on-line backpropagation," in *Proc. 13th Int. Symp. Syst. Synthesis*, 2000, pp. 225–230.
[44] M. Bahoura and C.-W. Park, "FPGA-implementation of high-speed MLP neural network," in *Proc. 18th IEEE Int. Conf. Electron., Circuits, Syst.*, 2011, pp. 426–429.
[45] S. Himavathi, D. Anitha, and A. Muthuramalingam, "Feedforward neural network implementation in FPGA using layer multiplexing for effective resource utilization," *IEEE Trans. Neural Netw.*, vol. 18, no. 3, pp. 880–888, 2007.
[46] A. R. Omondi and J. C. Rajapakse, *FPGA Implementations of Neural Networks*. New York: Springer, 2006.
[47] R. Wang, C. S. Thakur, T. J. Hamilton, J. Tapson, and A. van Schaik, "A neuromorphic hardware architecture using the neural engineering framework for pattern recognition," *arXiv:1507.05695*, Jul. 2015.
[48] R. Andraka, "A survey of CORDIC algorithms for FPGA based computers," in *Proc. 1998 ACM/SIGDA 6th Int. Symp. Field Program. Gate Arrays (FPGA '98)*, pp. 191–200.
[49] O. Mencer and W. Luk, "Parameterized high throughput function evaluation for FPGAs," *J. VLSI Signal Process. Signal, Image, Video Technol.*, vol. 36, no. 1, pp. 17–25, 2004.
[50] S. Eberhardt, T. Duong, and A. Thakoor, "Design of parallel hardware neural network systems from custom analog VLSI 'building block' chips," in *Proc. Int. Joint Conf. Neural Netw.*, 1989, vol. 2, pp. 183–190.
[51] P. H. W. Leong and M. A. Jabri, "A low power trainable analog neural network classifier chip," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC '93)*, pp. 4.5.1–4.5.4.
[52] Y. Wang, "Analog CMOS implementation of backward error propagation," in *Proc. IEEE Int. Conf. Neural Netw.*, 2011, vol. 46, no. 6, pp. 701–706.

Chetan Singh Thakur (M'14) received the M.Tech. degree in biomedical engineering from the Indian Institute of Technology, Bombay, in 2007. Following this, he has worked with Texas Instruments as a Senior Integrated Circuit Design Engineer in the area of mobile processor. In 2013, he has joined as a Ph.D. student in Bioelectronics and Neuroscience group in the MARCS Institute at the Western Sydney University, Australia. His research interests include neuromorphic engineering, stochastic electronics, and computational neuroscience.

Runchun Wang (M'13) received the M.Sc. degree in electrical engineering from the Shanghai Jiaoong University, Shanghai, China, in 2008 and the Ph.D. degree in neuromorphic engineering from the Western Sydney University, Sydney, Australia, in 2013. He is a Postdoctoral Fellow in the Biomedical Engineering and Neuroscience (BENS) research program. His research focuses on neuromorphic engineering, mixed-signal/analog VLSI design, ASIC/SoC/FPGA design, computational neuroscience, deep network, machine learning, cognition systems, and signal processing.

Tara Julia Hamilton (S'97–M'00) received the B.E. degree in electrical engineering with first class honors and the B.Com. degree from the University of Sydney, Australia, in 2001, the M.Eng.Sc. degree in biomedical engineering from the University of New South Wales, Sydney, in 2003, and the Ph.D. degree from the University of Sydney in 2008. She is currently a Senior Research Lecturer in bioelectronics and neuroscience within the MARCS Institute at the Western Sydney University, Australia. Her current research interests include neuromorphic engineering, mixed-signal integrated circuit design, and biomedical engineering.

Jonathan Tapson (M'05) received the B.Sc. (physics), B.Sc. (electrical engineering), and Ph.D. degrees in engineering from the University of Cape Town, South Africa. He is a Professor of the Western Sydney University, Australia, which he joined in June 2011, having previously been Head of Electrical Engineering at the University of Cape Town. His research interests are in bio-inspired sensors and systems; with his coauthors he is currently working on a major program of analog and mixed signal IC design in the area of stochastic electronics. He is a former President of the South African Council on Computation and Automation, and is a Fellow of the South African Academy of Engineering.

André van Schaik (M'00–SM'02–F'14) received the M.Sc. degree in electrical engineering from the University of Twente, Enschede, The Netherlands, in 1990 and the Ph.D. degree in electrical engineering from the Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland, in 1998. He is a Professor of Bioelectronics and Neuroscience in the MARCS Institute at the Western Sydney University. His research focuses on three main areas: neuromorphic engineering, bioelectronics, and neuroscience. He has authored more than 150 papers and is an inventor of more than 30 patents. He is a founder of three start-up companies: VAST Audio, Personal Audio, and Heard Systems.