Large-Scale Memristive Associative Memories

Eero Lehtonen, Jussi H. Poikonen, Mika Laiho, *Member, IEEE*, and Pentti Kanerva

***Abstract*— Associative memories, in contrast to conventional address-based memories, are inherently fault-tolerant and allow retrieval of data based on partial search information. This paper considers the possibility of implementing large-scale associative memories through memristive devices jointly with CMOS circuitry. An advantage of a memristive associative memory is that the memory elements are located physically above the CMOS layer, which yields more die area for the processing elements realized in CMOS. This allows for high-capacity memories even while using an older CMOS technology, as the capacity of the memory depends more on the feature size of the memristive crossbar than on that of the CMOS components. In this paper, we propose the memristive implementations, and present simulations and error analysis of the autoassociative content-addressable memory, the Willshaw memory, and the sparse distributed memory. Furthermore, we present a CMOS cell that can be used to implement the proposed memory architectures.**

***Index Terms*— Associative memory, memristors, mixed analog digital integrated circuits.**

I. Introduction

THE memristor is a passive programmable resistive component that was theoretically discovered and described in 1971 by Chua [1]. In 2008, scientists from HP Labs revealed to have found a nanoscale device which can be classified as a memristive device [2]. Since this announcement, many physical memristive devices have been reported [3]–[7]; it has been stated that all two-terminal memory devices based on resistive switching are memristors [8]. Memristors are particularly well-suited for memory applications; they are practically nonvolatile, which means that they retain their state when unpowered, and the feature size of a memory structure realized using memristive devices is minimal, as each memory element consists of a single memristive device.

Memristors are naturally fabricated within a nanowire crossbar where a memristor is placed at each crossing of two nanowires [9]. The nanowires are driven by CMOS circuitry, which is proposed to be located physically below the nanowire crossbar [10]. This CMOS/molecular hybrid (CMOL) architecture makes memristive nanowire crossbars ideally suited for facilitating programmable communication and memory within a CMOS chip. Neuromorphic systems, which are inspired by the functionality and architecture of the nervous system, are particularly well-suited to be implemented as CMOL architectures. In these architectures, neurons are implemented using CMOS components, whereas the intercellular communication and synapses are realized through memristive nanowire crossbars [11].

In this paper, we consider various memristive implementations of a neuromorphic memory structure called the associative memory. In contrast to random access memories, where the information is stored to and retrieved from explicitly given locations, the information is retrieved through a search in associative memories; given an input vector one wants to obtain the stored vector that has been previously associated with the input. Such a search typically requires computations of many thresholded inner products between the input vector and the contents of the memory. In a parallel hardware implementation of a large-scale associative memory, one thus needs many such inner product units, which in neuromorphic terms can be regarded as artificial neurons. In [12], it is shown that to improve energy efficiency, such neurons should be implemented in the analog rather than in the digital domain—we apply this approach in the associative memory circuits proposed in this paper.

The prospect of high-capacity associative memory architectures is the prime motivation in this paper. In particular, in this paper it is assumed that the wordlength L of the memory input—and output—is very large, in the order of thousands of bits. Such a memory architecture would be well suited, for example, for real-time pattern recognition in natural images, which would be useful in autonomous robotics, to name one particular field of application. Moreover, there is a reason to believe that the associative memory in the brain uses high-dimensional input and output data [13], and thus such a memory may be needed for implementing a whole-brain model such as the one discussed in [14].

In the following, we propose CMOL implementations of three associative memory structures, namely the memristive ACAM described in Section III-C, and the memristive Willshaw and sparse distributed memories presented in Sections IV-A and IV-C. All of these memory structures are well suited to be implemented as CMOL architectures, as they require large synaptic networks to achieve high capacities.

II. Devices, Architectures, and Definitions

A. Memristive Devices

A memristor is a nonvolatile programmable resistor, whose resistance, or memristance to be precise, can be changed by applying a voltage across, or current through, the device. The number of allowed resistive states of a memristor depends

Manuscript received March 5, 2012; revised December 6, 2012; accepted February 5, 2013. Date of publication April 30, 2013; date of current version February 20, 2014. This work was supported by the GETA Graduate School, by the Fulbright Foundation, and by the Academy of Finland, under Grant 140108 and Grant 253596.

E. Lehtonen and M. Laiho are with the BID Technology, University of Turku, Turku 20520, Finland (e-mail: elleht@utu.fi; mika.laiho@utu.fi).

J. H. Poikonen is with the Department of Communications and Networking, Aalto University, Espoo 02150, Finland (e-mail: jussi.poikonen@aalto.fi).

P. Kanerva is with the Redwood Center for Theoretical Neuroscience, University of California, Berkeley, CA 94720 USA (e-mail: pkanerva@csli.stanford.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/TVLSI.2013.2250319

on its fabrication. For example, bistable devices called binary memristors have been reported in [2] and [4], and an analog memristor with a seemingly continuous range of memristances is presented in [7]. The associative memory implementations described in this paper use both binary and analog memristors. Binary memristors are used as programmable switches, whereas analog memristors are used to implement the counters needed in the sparse distributed memory (SDM) discussed in Section IV.

A binary memristor is modeled in this paper as a linear resistor with two possible resistive states, R_{OFF} and R_{ON}. A binary memristor's state can be switched by applying a sufficiently large positive or negative voltage across it. More precisely, the resistance state $R(t)$ of a binary memristor at time t satisfies

$$R(t) = \begin{cases} R_{\mathrm{OFF}}, & \text{if } v(t) < -V^T \\ R_{\mathrm{ON}}, & \text{if } v(t) > V^T \\ R(t-1), & \text{otherwise} \end{cases} \tag{1}$$

where V^T is the programming threshold of the memristor and $v(t)$ is the voltage across it at time t.

For the analog memristor, we apply the qualitative model presented in [15]. Accordingly, the behavior of a memristor can be defined by the following two equations:

$$i = c_1 w \sinh(d_1 v) \tag{2}$$

$$\frac{\mathrm{d}w}{\mathrm{d}t} = c_2 \sinh(d_2 v) \tag{3}$$

where i and v are the current through and voltage across the device, c_k and d_k, $k = 1, 2$ are positive constants, and $w \in [0, 1]$ is the state variable of the memristor.

We assume that the analog memristors used in this paper have an effective programming threshold V^T [15], which means that there exists a range of voltages across the device that do not cause the conductivity state of the memristor to change rapidly. In the qualitative model, the programming threshold corresponds to a large value of the exponent d_2 in (3) relative to the timescale of the programming operation.

We consider two approaches to programming, or changing the state of conductivity, of memristors. As noted above, binary memristors are programmed to the low-conductivity state denoted by 0 by applying a voltage smaller than $-V^T$ across the device. Conversely, programming to the high-conductance state 1 is performed by applying a voltage larger than $+V^T$ across the memristor. Using a positive voltage below the threshold voltage makes it possible to read the state of the memristor without changing it. Analog memristors can be programmed by pulse-based programming in which square voltage pulses are set across the memristor to change its state of conductivity by an amount depending on the number and the amplitude of the pulses.

B. CMOL Circuits

An example of a CMOL-type CMOS/memristor hybrid architecture [10], [16] is illustrated in Fig. 1. It consists of a CMOS layer stacked vertically with a memristive crossbar. The CMOS layer is used for signal restoration, gain, and

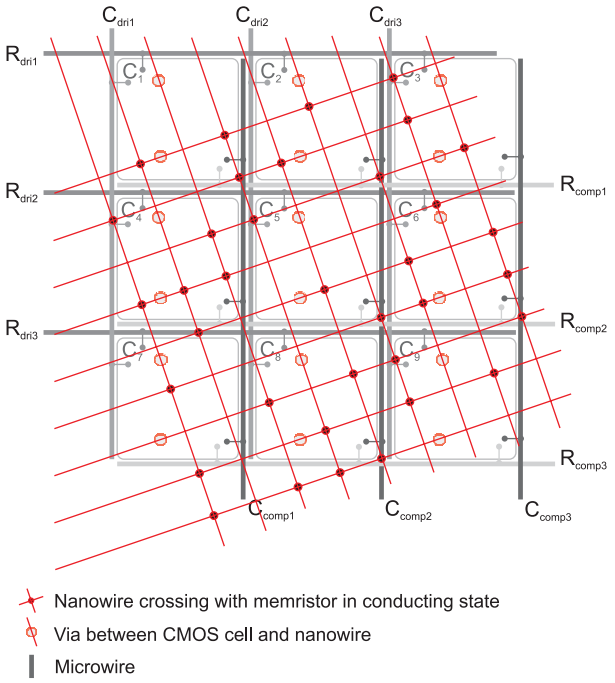

Fig. 1. Example CMOL circuit with 9 CMOS cells connected to a memristive crossbar containing $9 \cdot 9 = 81$ memristors. The CMOS cells are depicted as square tiles, with circular interfaces to the nanowire crossbar. The nanowire crossbar is tilted with respect to the array of CMOS cells to allow each CMOS cell to access a unique pair of nanowires. Each CMOS cell is addressed by four microwires: C_{dri}, R_{dri}, C_{comp}, and R_{comp}.

processing, and it is interfaced with the memristive crossbar, which acts as a memory layer and as a programmable communication network for the chip, by vertical vias. The CMOS layer is divided into cells whose designs depend on the functionality of the system. For example, if a CMOL architecture is used as a random access memory, the CMOS cells comprise merely of two pass-transistors, which are used to select two mutually perpendicular nanowires and thus the memristor located between them. As illustrated in Fig. 1, each CMOS cell can drive one horizontal and one vertical nanowire, and thus two CMOS cells are required to address a single memristor located at the crossing of a horizontal and a vertical nanowire. To simultaneously control only these two CMOS cells, four microwires are needed—thus the CMOL architecture generally requires double addressing compared, for example, to an array of CMOS cells. It is also crucial to note that in a CMOL architecture N CMOS cells can control at most N^2 memristors. This upper limit can be attained in the ideal case of nonsegmented nanowires that cover the whole CMOS cell array. This approach is assumed in this paper where we consider a 100×100 CMOS cell array used to control 10^8 memristors. Another solution to implement the CMOL architecture is to vertically stack segmented nanowire crossbars as explained in [9]. The largest fabricated CMOL-type random access memories published to date are presented in [17] and [18]. In [17], a 64-Mb memristor memory is presented, while [18] describes a 8-Gb phase-change memory, a technology closely related to memristors. Although these examples demonstrate the viability of the CMOL architecture,

more experimental data is still required to determine the practical limitations of reliable fabrication of CMOL circuits. From the perspective of in this paper, the most significant open question is the maximum length of unsegmented nanowires. However, large-scale associative memories can be implemented using relatively wide nanowires, which should increase the reliability of the fabrication process. In the examples of Sections III and IV, the nanowire width is chosen to be 125 nm, which is attainable for interconnects in modern CMOS processes.

In this paper, the CMOS cells are used to implement artificial neurons that compute thresholded sums of input currents, which represent inner products of vectors as described in Section III-A. Vectors are stored into the memristive crossbar as the states of the memristors, and the input vector is communicated to the CMOS cells by address decoders at the edges of the CMOL architecture.

C. Associative Memory

Let $\mathbf{u}_i$ and $\mathbf{v}_i$, where $i = 1, \ldots, M$, be binary vectors of length L. An associative memory can store a set of associations $\mathbf{u}_i \rightarrow \mathbf{v}_i$ between these vectors. Formally, this means that when the memory is searched by a vector $\mathbf{z}$, it returns the index i (or vector $\mathbf{v}_i$) for which the Hamming distance

$$d_H(\mathbf{z}, \mathbf{u}_i) = \sum_j^L (\mathbf{z}(j) - \mathbf{u}_i(j))^2 \tag{4}$$

is minimized. Here, $\mathbf{x}(j)$ is the jth element of $\mathbf{x}$.

A memory satisfying the definition above is called heteroassociative, and besides simply storing key-value pairs, it can also be used to store sequences of vectors provided that the key and value vectors have the same length. A sequence is formed by letting the value of a previous pair to be the key of the next pair. When $\mathbf{u}_i = \mathbf{v}_i$ for all i, the memory is called autoassociative, and it allows for pattern completion or error correction.

An early review and system-theoretical formulation of associative information structures was given in [19]; notably also physical realizations of content-addressable and distributed memory structures are presented in this monograph. Furthermore, a first broad review and analysis of content-addressable memories (CAMs) and their hardware implementations were presented in [20]. Further discussions on the theoretical basis of the memory architectures considered in this paper can be found for example in [21]–[25].

D. Data Representation

In this paper, the vector length L is assumed to be very long, in the order of thousands of bits. Furthermore, all stored vectors are assumed to be either sparse or dense. A sparse vector contains only a small fraction of ones, for example, 50 ones out of a total of $L = 10\,000$ b, while in dense binary vectors the numbers of zeros and ones are close to $L/2$.

Raw data is often inherently dense, as well as compressed data is dense, as it contains a maximal amount of information. There is thus use for associative memory architectures storing dense input patterns. The usefulness of sparse representation may be understood by considering a data-processing method called dimensionality reduction. In this scheme, a dense vector, such as a gray-scale bitmap image, is mapped into a sparse vector whose nonzero coordinates contain most of the variance in the original data. Sparse representation is suitable for pattern recognition and completion, since the nonzero entries correspond to specific features in the data. In sparse coding of natural images [26], the basis functions correspond to oriented local structures.

It is known that the neural activity in parts of the brain is sparse [27]. In the visual and auditory cortex, this may serve pattern recognition, but sparse activity of neurons is also beneficial in terms of metabolic efficiency [27]. This suggests that sparse representation may be useful in reducing the power consumption of hardware implementations of associative memories. Such reduction is observed in the power consumption of the Willshaw memory discussed in Section IV-A.

E. Unary and Distributed Architectures

Neural associative memory architectures can be divided into two categories: unary—also known as grandmother cell—architectures and distributed architectures [28]. In the former, each stored vector is allocated to a specific neuron which activates only when a vector close to it is given as input to the network. It follows from this that the number of neurons, N, is the upper limit to the number of binary vectors, M, the network can store, that is, $M \leq N$. Generally, a unary associative memory can be implemented as a lookup table or as a CAM as discussed in Section III.

In distributed memory architectures, multiple neurons are activated for each input. In principle, this enables storing more input vectors than there are neurons in the network, that is, it is possible that $M > N$. The upper limit for the number of vectors that can be retrieved without errors depends not only on the memory architecture but also on the data distribution of the input vectors. When operating with sparse vectors the storage capacity of the network may be much greater than N when N is large. Memristive implementations of two distributed memory architectures are proposed in Section IV.

F. Capacities of Associative Memories

The network capacity C of an associative memory is defined as the maximum quantity of stored information per synapse [29]. By definition, C is always non-negative, and for binary synapses it satisfies $C \leq 1$. To calculate the vector capacity M, which is the number of vectors that can be stored into the memory, one needs to know C, the total number of synapses S, and the information content in a single vector I, assuming that the input vectors are independent and contain an equal quantity of information. Then

$$M = CS/I. \tag{5}$$

For example, in a square associative memory with N neurons, the number of synapses equals $S = N^2$. If logarithmically sparse vectors of length $L = N$ are used, each vector contains

TABLE I
NETWORK CAPACITIES OF ASSOCIATIVE MEMORY ARCHITECTURES

Architecture	C_D	C_S
Autoassociative CAM	1	N/A
Willshaw memory	N/A	0.69
Sparse distributed memory	0.15	≥ 0.69

$I \approx \log(N)^2$ bits of information. Then the vector capacity of the memory equals

$$M \approx \frac{CN^2}{\log^2(N)}. \tag{6}$$

On the other hand, if dense vectors are stored, then $I = N$ and

$$M = CN. \tag{7}$$

In Table I, we have collected network capacity values of the associative memories discussed in this paper. These results are adopted from [23], [29], and [30]. Here C_D and C_S denote the capacity for dense and sparse data vectors, respectively. Notice that for ACAM we do not consider sparse vectors, but Willshaw memory is assumed to operate only on sparse vectors.

G. Previous Work

Memristive associative memories were presented in literature previously in [31]–[34]. Of these, the most relevant to the current work is [31], which discusses the implementation of a classic associative memory architecture called the Hopfield network [22] as a CMOL circuit. This implementation requires analog synaptic weights, which are proposed to be realized as small crossbars of binary devices, and it yields a vector capacity of $M = 0.118L$ [35]. Being a CMOL-based neuromorphic network, this architecture closely resembles the ones described in Sections III and IV. The Hopfield network has several disadvantages when compared with the memory designs described in this paper. As noted above, the capacity of the memory is directly proportional to the length of the input vector. With the memristive ACAM described in Section III or the SDM discussed in Section IV, this is not the case, as their capacities depend on the number of rows in the memory matrices, and can therefore be chosen independently from the input vector length. Moreover, the Hopfield network is a dynamical system, whose artificial neurons need to communicate with each other multiple times during an associative search to find the equilibrium state that corresponds to the output of the search. In contrast to this, the memory structures described in this paper yield the output of a search in a single step. The capacity of a Hopfield network using dense input vectors equals that of a similarly sized SDM, whereas on logarithmically sparse vectors, the capacities of a Hopfield network and the Willshaw memory described in Section IV are roughly the same [29]. Therefore, we conclude that the associative memory architectures proposed in this paper are better suited for a memristive implementation than the Hopfield network, since they yield the same or higher capacities with a simpler implementation.

Memristive implementation of a CAM is described in [32]. The CAM is a memory architecture which compares input search data with a table of stored data, and returns the address of the matching data [36]. It is not an associative memory as defined in the context of this paper, as it recognizes an input only if it exactly matches a stored vector. Therefore, despite its name, the design of the ACAM in Section III differs significantly from the one described in [32].

In [33] and [34], the dynamics of analog memristors are used to create associative responses in small-scale memristive circuits. For example, [33] presents a three-memristor circuit, whose dynamics are reminiscent of the famous Pavlov's experiment. These considerations are not relevant to our work, since here we consider memristors only as synaptic weights and do not exploit their dynamics except when the devices are programmed. Moreover, in our approach it is crucial that the number of synapses is much larger than the numbers of devices considered in [33] and [34].

III. MEMRISTIVE AUTOASSOCIATIVE CAM

A. Inner Product as a Sum of Memristor Currents

As mentioned in Section I, the inner product is a basic operation used in the associative search. In the following, we describe how to implement a specific type of inner product with memristors; this method will be an integral part of all the memory implementations proposed in this paper.

Let $\mathbf{a}$ and $\mathbf{b}$ be vectors of length L, where the values of $\mathbf{a}$ are nonnegative real numbers, and the values of $\mathbf{b}$ are binary, taken from the set $\{0, 1\}$. Consider the inner product

$$\mathbf{b} \circ \mathbf{a} = \sum_{j=1}^{L} \mathbf{b}(i)\mathbf{a}(j) = \sum_{\mathbf{b}(j)=1} \mathbf{a}(j) \tag{8}$$

where, as before, $\mathbf{a}(j)$ is the jth component of $\mathbf{a}$.

The elements of $\mathbf{a}$ are represented in the following by the currents flowing through memristors when a unit voltage V_{unit} is set across them. More precisely, let the current I_j flowing through the jth memristor equal

$$I_j = \hat{c} \cdot \mathbf{a}(j) \tag{9}$$

for all $j = 1, \ldots, L$, when the unit voltage is set across it, where $\hat{c}$ is a scaling constant common to all of the memristors.

Let $V_j = \mathbf{b}(j) \cdot V_{\text{unit}}$ for all j in the circuit of Fig. 2. By Kirchoff's current law, the current flowing through the resistor R_{ref} is

$$\sum_{\mathbf{b}(j)=1} \hat{c} \cdot \mathbf{a}(j) \tag{10}$$

and therefore the voltage V_{out} equals

$$V_{\text{out}} = c \cdot \mathbf{b} \circ \mathbf{a} \tag{11}$$

where $\circ$ denotes the inner product, and the constant $c = -\hat{c} \cdot R_{\text{ref}}$.

For the autoassociative CAM described in this section, the vector $\mathbf{a}$ is binary, and its elements can thus be represented with binary memristors. Here, we assume that the resistance ratio $R_{\text{OFF}}/R_{\text{ON}}$ is large enough so that a memristor in the OFF-state yields a negligible amount of current when compared

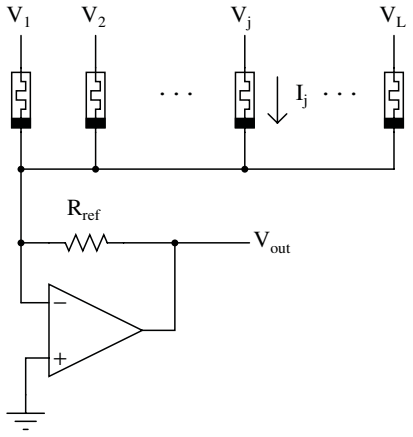

Fig. 2. Memristive circuit for computing the inner product $\mathbf{b} \circ \mathbf{a}$. The elements of the binary vector $\mathbf{b}$ are represented by the voltages V_j, $j = 1, \ldots, L$, whereas the elements of the real-valued vector $\mathbf{a}$ are represented by the currents I_j. The voltage V_{out} is directly proportional to the value of the inner product.

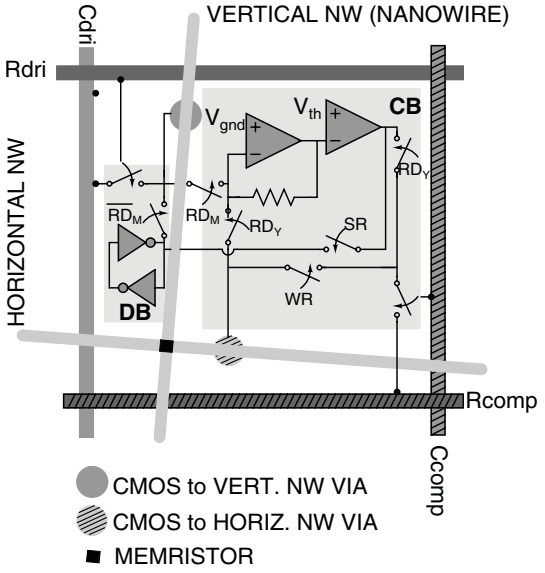

Fig. 3. Schematic of a single CMOS cell of the memristive ACAM.

with a memristor in the ON-state. Analog memristors are used in Section IV for representing counters of the SDM.

B. Autoassociative CAM

An ACAM [28] compares input search data against a table of stored data, and returns the addresses of the stored data, which are nearest to the input data in Hamming distance. As ACAM is a unary architecture, its vector capacity equals $M = N$, where N is the number of memory rows, independent of the distribution of the input data. In this section, we assume dense representation of data; associative memories operating on sparse data are considered in Section IV.

Let the contents of the memory be represented by a binary matrix U, whose rows $\mathbf{u}_i$ correspond to the stored vectors, and let $\mathbf{z}$ be the input vector for the associative search. Therefore, the search should yield the indices i for which the Hamming distance $H(\mathbf{z}, \mathbf{u}_i)$ is minimized. Since

$$H(\mathbf{z}, \mathbf{u}_i) = ||\mathbf{z}||^2 - 2\mathbf{z} \circ \mathbf{u}_i + ||\mathbf{u}_i||^2 \tag{12}$$

it follows that the vector $\mathbf{u}_i$ that minimizes the Hamming distance is the one which maximizes the inner product $\mathbf{z} \circ \mathbf{u}_i$. Indeed, since the vector length L is large and the vectors were assumed to be dense, it follows that:

$$||\mathbf{z}||^2 \approx ||\mathbf{u}_i||^2 \approx L/2. \tag{13}$$

For the hardware implementation, it is very convenient that the inner product is sufficient for measuring the distance between two vectors, and thus, only ones, and not zeros, need to be matched in the input vector and the stored vectors. As inner products can be computed with the method discussed in the previous subsection, we are now ready to present a memristive implementation of the ACAM.

C. Implementation of a Memristive ACAM

We consider a CMOL-type implementation of an ACAM whose memory elements are binary memristors. The contents matrix U is represented by the resistance states of the memristors of the nanowire crossbar, as depicted in Fig. 1. The vertical nanowires are used for communicating the input and output vectors, and input and stored vectors are compared with the horizontal nanowires. All of the nanowires are interfaced with CMOS blocks as depicted in Fig. 3—the vertical nanowires are interfaced with driver blocks (DB), whereas the horizontal nanowires are interfaced with comparison blocks (CB).

In general, the length L of an input vector and the number N of the stored vectors need not be equal. Consider, for example, the case $N > L$. This corresponds to N horizontal and L vertical nanowires, and thus $N - L$ of the CMOS cells consist only of a CB depicted in Fig. 3. Notice here that we use the word cell instead of a neuron, since each of the CMOS cells includes a local memory latch and inputs of multiple global signals, and therefore contains more functionality than a generic artificial neuron. This choice of terminology is maintained in Section IV where the CMOS cell is extended to function as a part of the SDM. In the following, we describe in detail the operation of the proposed memristive ACAM.

1) Storing a Vector: When a binary vector $\mathbf{u}$ is stored into the memory, an available row of the ACAM is chosen. The corresponding horizontal nanowire is driven to a negative voltage $-V_{\text{PROG}}$, and the vertical nanowires are driven to voltages that correspond to the bits of $\mathbf{u}$: the jth vertical nanowire is driven to a positive voltage V_{PROG} if the corresponding bit $\mathbf{u}_j = 1$, and to ground otherwise. The voltage V_{PROG} is chosen to satisfy

$$V_{\text{PROG}} < V^T < 2V_{\text{PROG}} \tag{14}$$

where V^T is the programming threshold of the memristor. This assignment of voltages programs the bits of the input vector as the resistances of the memristors on this row of the nanowire crossbar, assuming that the memristors are initially in the OFF-state: bit zero is represented as a memristor in the OFF-state, and bit one is represented as a memristor in the ON-state. If the row has been used previously, one may first initialize

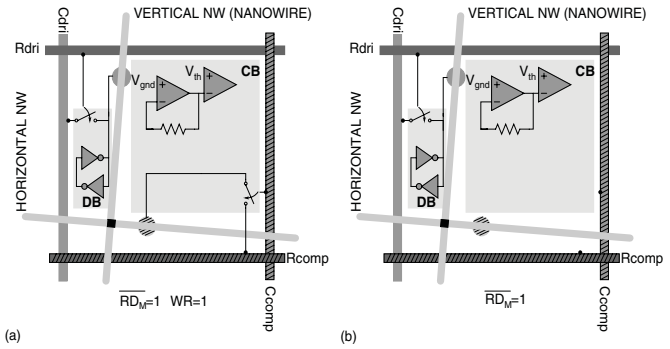

Fig. 4. Memristive ACAM CMOS cell configured for the write-in operations. (a) Storing the input vector as the states of the memristors on a horizontal nanowire. The programming of the memristors is performed by driving the vertical nanowires sequentially by voltages $\{V_{\mathrm{PROG}}, \mathrm{GND}\}$ corresponding to the input bits, and by driving the selected horizontal nanowire at a constant negative voltage $-V_{\mathrm{PROG}}$. The horizontal nanowires not attending to this operation can be connected to ground, or they can be left floating. On these nanowires, the memristors are not programmed, as the voltage across them does not exceed their programming threshold. (b) For the search operation, the input vector bits are stored into the latches at the DBs.

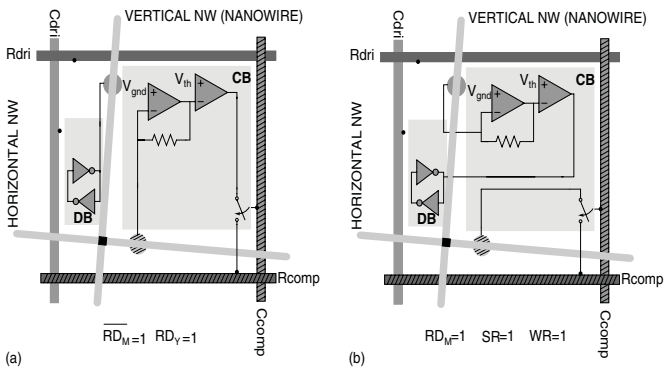

Fig. 5. Memristive ACAM CMOS cell configured for the read-out operations. (a) Search operation. The input vector bits are driven to the vertical nanowires by the DB latches. The horizontal nanowires are connected to virtual ground, and the incoming current is measured and thresholded. The result of the threshold comparison can be read from the Rcomp microwire. (b) Readout of a stored vector. A selected horizontal nanowire is driven with a read voltage, and the vertical nanowires are connected to the virtual grounds. The incoming currents from the vertical nanowires are thresholded, and the results are stored into the latches at the DBs, from where they can be read out.

it by driving a negative voltage across the corresponding memristors. The selection of the nanowires is accomplished with the CMOS microwires denoted by C_{dri}, R_{dri}, C_{comp}, and R_{comp}, as is depicted in Figs. 1 and 3. In particular, R_{dri} controls the switch connecting C_{dri} to the latch in the DB, whereas C_{comp} controls the switch connecting R_{comp} either to the CMOS circuitry in the CB or directly to the horizontal nanowire, depending on the selected global configuration of the memory circuit.

2) Search Operation: When the memristive ACAM is searched by a binary vector $\mathbf{z}$, the DBs are used to drive the vertical nanowires to voltages corresponding to $\mathbf{z}$: the jth vertical nanowire is driven either to a positive voltage V_{READ} if $\mathbf{z}_j = 1$, or to ground if $\mathbf{z}_j = 0$. For this, the search vector must be stored into the DB latches before the search as depicted in Fig. 4(b), since the addressing scheme does not allow providing data to all vertical nanowires simultaneously. Indeed, there are only of the order of $\sqrt{N}$ address lines for the N DBs. The currents coming in from the horizontal nanowires to the virtual ground of the comparison cells are then measured and thresholded with a negative threshold voltage V_{th}, which should not be confused with the memristor programming threshold V^T. As described in Section III-A, the voltage at the input of the comparator represents the value of an inner product $\mathbf{z} \circ \mathbf{u}_i$. Thus the horizontal nanowires whose currents exceed the threshold value correspond to the rows of the ACAM, which are within a desired distance of the search vector. The result can be read from the output of the comparator, as shown in Fig. 5(a).

In general, multiple stored vectors $\mathbf{u}$ may be close enough to the search vector $\mathbf{z}$ to be selected during the search. The number of selected vectors $\mathbf{u}$ depends on the threshold voltage V_{th} that can be correspondingly adjusted, for example, in a logarithmic search, to yield k vectors $\mathbf{u}$ closest to $\mathbf{z}$.

3) Reading Out: The readout of a vector $\mathbf{u}_i$ is achieved by driving the corresponding horizontal nanowire while measuring the currents at the vertical nanowires. In Fig. 5(b), the read-out operation is depicted. The value corresponding to a bit in the stored vector is written into the DB's latch, from which it can be driven onto the C_{dri} microwire.

D. Simulation of the ACAM Cell

To demonstrate the operation of the proposed memristive memory design, we simulated the 9×9 example circuit depicted in Fig. 1 using Cadence Spectre software with the HCMOS9GP process from ST Microelectronics. The CMOS cell depicted in Fig. 3 was implemented using two two-stage operational amplifiers, multiple pass-transistor switches, and some basic logic gates. We estimated the power consumption and area of a circuit with 10 000 CMOS cells and 10^8 memristors. In these simulations, we applied the binary memristor model described in Section II with parameters $R_{\mathrm{ON}} = 10\ \mathrm{M\Omega}$, $R_{\mathrm{OFF}} = 10\ \mathrm{G\Omega}$, and $V^T = 1$ V. These resistance values correspond to the physical memristor reported in [37]. The capacitance between perpendicular nanowires was estimated as a plate capacitance

$$C_{\mathrm{nw}} = \frac{N \epsilon_r \epsilon_0 W_{\mathrm{nw}}^2}{d} \tag{15}$$

where $N = 10\,000$, $\epsilon_0 \approx 8.85419 \cdot 10^{-12}$ F/m is the vacuum permittivity, $W_{\mathrm{nw}} = 125$ nm is the nanowire width, and d is the vertical distance between perpendicular nanowires, assumed to be 10 nm. The nanowire width assumed here is conservative, as for example in [38] an analog memristor with width 100 nm was reported. For the relative permittivity ϵ_r, we assumed the values 3.9 and 120, which, respectively, correspond to silicon dioxide and titanium dioxide two materials that can be used in the memristor switching layer. Thus, the approximate values 0.6 and 18 pF are obtained for C_{nw}. The simulations described in the following are performed using both of these capacitance values. The parasitic capacitances between parallel nanowires and between nanowires and the CMOS layer are modeled as 2 pF per nanowire. The voltage waveforms shown in Fig. 6 correspond to the more difficult case of $C_{\mathrm{nw}} = 18$ pF.

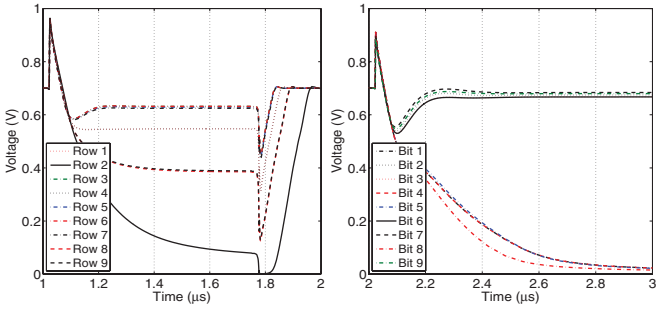

Fig. 6. Voltage waveforms at the comparator inputs of the CMOS cells in the 9 × 9 memristive ACAM example. The search operation is performed in the left subfigure from 1 to 2 μs and the readout of the second row of the memory matrix is performed in the right subfigure from 2 to 3 μs.

In the following simulations, we consider the search and read operations of the proposed ACAM circuit, but not memristor programming, which is analyzed in detail, for example, in [10] and [15], and also discussed in Section IV-D.

1) Example: 9 × 9 Memristive ACAM: Let the content of a 9 × 9 ACAM correspond to the matrix

$$U = \begin{pmatrix} 0&1&0&1&0&1&0&1&0 \\ 1&0&0&1&1&0&0&1&0 \\ 0&0&1&1&0&0&1&0&1 \\ 1&1&1&0&0&0&0&1&0 \\ 0&1&0&0&1&0&1&0&1 \\ 1&0&0&0&0&1&1&0&1 \\ 0&0&1&0&1&1&0&0&1 \\ 1&0&0&1&0&1&0&1&0 \\ 1&0&1&1&1&0&0&0&0 \end{pmatrix}$$

where 0 represents a memristor in OFF-state and 1 represents a memristor in ON-state, as shown in Fig. 1. Notice that each row contains exactly four ones, which means that the total resistance seen by each CB can be approximated by $R_{\mathrm{ON}}/4$. To keep this total resistance equivalent to the circuit containing 10^8 memristors, we choose here $R_{\mathrm{ON}} = 8$ kΩ, as 8 kΩ/4 = 10 MΩ/5000—assuming dense data, each row of a 10^8–memristor ACAM contains 5000 memristors in ON-state.

We simulated the search operation for the 9 × 9 circuit by setting the search input as the second row of U. The values of the inner product are in this case equal to (2, 4, 1, 2, 1, 1, 1, 3, 3), where the ith entry equals the inner product between the search input and the ith row. Fig. 6 shows the waveforms obtained from the comparator inputs of each of the CMOS cells. Not shown in the figure, the input bits are stored into the latches from 0 s to 1 μs in parallel for each of the CMOS cell rows. From 1 to 2 μs the circuit performs the search operation, whose result is read at 1.75 μs. The voltage at the comparator input corresponding to the second horizontal nanowire has the smallest value, reflecting the fact that the inner product between the search input and the second row of U has the maximum value 4. Voltages at the CMOS cells corresponding to horizontal nanowires 8 and 9 have the second smallest voltages at the comparator input, corresponding to the value 3 of the corresponding inner products. Rows 1 and 4 yield the third smallest voltages corresponding to the inner product value of 2, while the rest of the voltages correspond to the inner product value of 1. Notice that the sum node is maintained by the inverting operational amplifier at the virtual ground at 0.7 V.

From 2 to 3 μs, the circuit is configured to read the contents of the second row of the memory matrix U. In this case, the comparator input voltages in the CMOS cells controlling vertical nanowires 1, 4, 5, and 8 are low, corresponding to ones in the stored vector.

Note that in the simulated CMOS cell the feedback resistor of the inverting operational amplifier is implemented as a PMOS transistor. This explains the nonlinearity in the amplification of input currents visible in Fig. 6. Furthermore, the operational amplifiers were designed with very low bias currents so that in practice they only pull excess current from the virtual ground. This is possible since, as noted in Section III-B, only ones in the input and stored vectors are matched in the search operation. This helps to reduce the power consumption of the operational amplifiers.

E. Performance Analysis of a 10^8—Memristor ACAM

Using the 9 × 9 simulation, and a separate simulation of a single CMOS cell together with memristances $R_{\mathrm{ON}} =$ 10 MΩ, $R_{\mathrm{OFF}} = 10$ GΩ, we have estimated the speed, power consumption, and area requirements of a memristive ACAM containing 10^4 CMOS cells and 10^8 memristors. The speed of the search and readout operations can be seen from Fig. 6, as the memristances and nanowire capacitances were selected in the 9×9 example to yield equivalent performance to a large-scale ACAM circuit. We conclude that the search and readout operations can each be performed within 1 μs. For the search operation, this means that the effective number of bit comparisons per second performed by the proposed memristive ACAM with 10^8 memristors equals $10^6 \cdot 10^8 = 10^{14}$.

To estimate the power consumption of an N-cell ACAM circuit analytically, let us denote the idle power consumption (zero input current, not driving a wire) of the considered CMOS cell as P_{idle}. Our simulations indicate that for the chosen CMOS process and design, $P_{\mathrm{idle}} \approx 5.9$ μW. Assuming dense data, $N/2$ of the cells are driving an input one, and each of the driver cells are connected to $N/2$ memristors in ON-state and $N/2$ memristors in OFF-state. Denoting the driving voltage corresponding to the input one by V_{mem}, one obtains the following approximation of the power consumed by the entire ACAM circuit during the search operation:

$$P^{\mathrm{A}}_{\mathrm{search}} \approx N P_{\mathrm{idle}} + V_{\mathrm{DD}} V_{\mathrm{mem}} \left(1/R_{\mathrm{ON}} + 1/R_{\mathrm{OFF}}\right) (N/2)^2 . \tag{16}$$

In these simulations, $N = 10\,000$, $V_{\mathrm{DD}} = 1.2$ V, and $V_{\mathrm{mem}} \approx 0.35$ V. In the read-out operation, only one horizontal nanowire is driven, and therefore its power consumption can be approximated as

$$P^{\mathrm{A}}_{\mathrm{readout}} \approx N P_{\mathrm{idle}} + V_{\mathrm{DD}} V_{\mathrm{mem}} \left(1/R_{\mathrm{ON}} + 1/R_{\mathrm{OFF}}\right) (N/2) . \tag{17}$$

The power consumption of the proposed CMOS cell was also simulated by assuming different correlations of the input vector and the stored vector corresponding to that cell, and

TABLE II
CIRCUIT PERFORMANCE SIMULATED USING CADENCE SPECTRE WITH 0.13 μm CMOS TECHNOLOGY. EACH OF THE SIMULATED OPERATIONS WAS ALLOCATED 1 μs, AS DEMONSTRATED IN FIG. 6

	P_{search}	P_{readout}	Area
Simulated CMOS cell	0.11 mW	7.2 μW	$< 40 \times 40$ μm
Estimated 10^8-bit ACAM	1.1 W	72 mW	$< 4 \times 4$ mm
Analytical estimate	1.1 W	60 mW	–

by separately simulating the two cases of the cell driving an input one or an input zero. The results of the simulations are summarized in Table II. An estimate for the power consumption of the search operation of a 10^8 memristor ACAM is 1.1 W; because the effective number of bit comparisons per second equals 10^{14}, it follows that the energy per bit comparison for the search operation is approximately 1.1×10^{-14} J. To further verify our results, we note that in the 9×9 example, the mean power consumption of a single CMOS cell in the search operation was approximately 0.10 mW, which is reasonably close to the simulated and analytical values in the 10^8—memristor case.

To estimate the area of the proposed ACAM CMOS cell on the 0.13-μm process, we used a layout tool to place all transistors within one CMOS cell to an area approximately 21×21 μm. The transistors were placed next to each other without overlap in the drain or source areas. Therefore, we consider it safe to assume that the total layout area of the cell, including wiring, should not exceed 40×40 μm—thus an ACAM circuit containing 10 000 of the proposed CMOS cells should fit to a chip of size 4×4 mm. Additionally in each of the cells the inverting operational amplifier requires a 200 fF capacitor which in this process is implemented on a separate metal-insulator-metal (MIM) layer that does not increase the size of the CMOS cell. The capacitor fits into the available cell area, as it requires an area of 10×10 μm on the MIM layer. We also note that the required 10^8 memristors fit within the total circuit area with the assumed nanowire width of 125 nm and a nanowire spacing of 275 nm.

1) Comparison With Pure CMOS Implementations: In pure CMOS implementations of associative memories, such as those presented in [12] and [39]–[41], the artificial neurons must share die area with the memory elements, which can be, for example, SRAM cells or floating-gate memories. In contrast to this, in a CMOL implementation the memory elements, memristors, are located above the CMOS, which frees up CMOS die area and enables more artificial neurons to be fabricated. Memristive associative memories should be relatively cost efficient as the memristive crossbars can be post-processed on top of a CMOS layer fabricated with an older, less dense process.

Recently a pure CMOS implementation of the ACAM was proposed in [41]. The energy per bit comparison of this circuit is approximately 1.46×10^{-14} J, which is comparable to the simulated energy per bit comparison value of 1.1×10^{-14} J of the memristive ACAM. Furthermore, a single memory cell in the ACAM of [41] occupies an area of 6.6×23.4 μm^2, whereas the memristive ACAM proposed in this paper uses memristors whose nanowire width is 125 nm and nanowire spacing is 275 nm, yielding thus a memristor footprint 400×400 nm^2. It should, however, be noted that this density comparison is only tentative, because no physical implementation of the memristive ACAM has so far been fabricated.

IV. DISTRIBUTED ARCHITECTURES

As demonstrated by the network capacities given in Table I, the associative memory architecture used in any given application should depend on the distribution of the stored data. If dense data is used, ACAM as described in the previous section is recommended. For operation with sparse data, sparse distributed memories are preferred. These are artificial neural-net associative memories that can be seen as associative generalizations of the conventional random-access memory (RAM). In contrast to RAM, where each address refers to its own memory location, addresses to sparse distributed memories activate multiple memory locations. This yields a distributed representation of the input address, which is used in the subsequent read and write operations in the memory; the data vector that is associated with an address is stored in multiple locations. In the following, we present memristive implementations of two distributed memory architectures: the Willshaw memory and the SDM. SDM can be seen as a generalization of the Willshaw memory with analog weights and an auxiliary memristive ACAM-type read-only memory which is used to make the size of the memory—and therefore also its capacity—independent of the input vector length.

A. Memristive Willshaw Memory

The Willshaw memory [25] is a neuromorphic heteroassociative memory, which uses binary synapses and stores sparse data vectors. As the ACAM, its contents can be represented by a binary matrix, which is denoted by W. A key-value pair $\mathbf{u} \rightarrow \mathbf{v}$ of two sparse column vectors of lengths L and N, respectively, is stored into the memory by updating the matrix W according to the outer-product rule

$$W := \text{OR}(W, \mathbf{v}\mathbf{u}^T) \tag{18}$$

where the Boolean OR-operation is performed elementwise. As noted in Section II-C, autoassociation is achieved by setting $\mathbf{u} = \mathbf{v}$. If the stored vectors are sparse and have K ones, the vector capacity of the Willshaw memory equals

$$M \approx 0.69(N/K)^2. \tag{19}$$

As the number of ones in the matrix W never decreases when storing a new vector into the memory, one should use sparse representation of data in order to limit the amount of overwriting of previously stored data. In the information theoretical sense, the maximum capacity of this memory is obtained when $K = \log_2(N)$, as is shown in [29].

The search of a Willshaw memory is performed as L thresholded inner products between the rows of W and the search vector $\mathbf{z}$

$$\mathbf{v}(j) = H(\mathbf{w}_j \circ \mathbf{z} - \Theta) \tag{20}$$

where Θ is a threshold value and H is the Heaviside function

$$H(a) = \begin{cases} 1, & a \geq 0 \\ 0, & a < 0. \end{cases} \tag{21}$$

The optimal choice of the threshold depends on the types of bit errors present in the address $\mathbf{z}$. In the original Willshaw model [25], the threshold $\Theta = \sum \mathbf{z}_i$ was specified; this threshold is optimal in the case where $\mathbf{z}$ contains only miss-type errors, that is, only values $z_i = 0$ are potentially erroneous. For simplicity, we choose this threshold for the memristive implementation of a Willshaw memory.

The only difference between the search operation of the Willshaw memory and that of the ACAM is the distribution of the input data. Indeed, with the Willshaw memory the search vector is sparse, and thus the threshold should be smaller than with the ACAM. Storing vectors is also performed very similarly to the ACAM, the only difference being that in the Willshaw memory the input is stored on multiple rows. Therefore, the memristive ACAM architecture described in Section III and shown in Fig. 3 implements the Willshaw memory as well. As a conclusion, this memory architecture should be configured as an ACAM when dense vectors are used, and as a Willshaw memory when sparse vectors are used.

The advantage of using sparse data is evident when considering the power consumption of the search operation of the proposed memory circuit. The power consumption of the search operation with the Willshaw memory can be attained from (16) by substituting the number of driven input ones by K, resulting in

$$P^{\mathrm{W}}_{\mathrm{search}} \approx N P_{\mathrm{idle}} + V_{\mathrm{DD}} K V_{\mathrm{mem}} \left(1/R_{\mathrm{ON}} + 1/R_{\mathrm{OFF}}\right) (N/2)\,. \tag{22}$$

As with sparse data $K/N \ll 1$, it follows that $P^{\mathrm{W}}_{\mathrm{search}} \ll P^{\mathrm{A}}_{\mathrm{search}}$. It is important to notice that the Willshaw memory does not have a separate readout operation, as the search operation yields the output vector directly.

B. Structure and Operation of a SDM

The SDM is an artificial-neural-net associative memory whose circuit resembles that of the cerebellar cortex [24]. It also resembles the conventional RAM architecture more than the Willshaw memory does, as it uses an explicit address vector along with a word-in vector. A high-level view of the SDM architecture is depicted in Fig. 7. It consists of two parts: a memristive ACAM-type read-only address matrix $\mathbf{A}$, and a Willshaw-type content matrix $\mathbf{C}$, whose elements are integer counters of small absolute value. The binary address matrix is used to produce a sparse activation vector $\mathbf{y}$, which indicates rows of the content matrix used in the store and retrieve operations.

A vector $\mathbf{z}$ is stored into SDM by incrementing the jth counters of all activated rows if $\mathbf{z}_j = 1$, and by decrementing them otherwise. The retrieve operation is performed by the columnwise summing of contents of activated rows of the content matrix and by thresholding the result, thus yielding $\mathbf{w} = H(\mathbf{y}^\top \mathbf{C})$, where H is the Heaviside step function (21). In other words, the read operation works as with the Willshaw

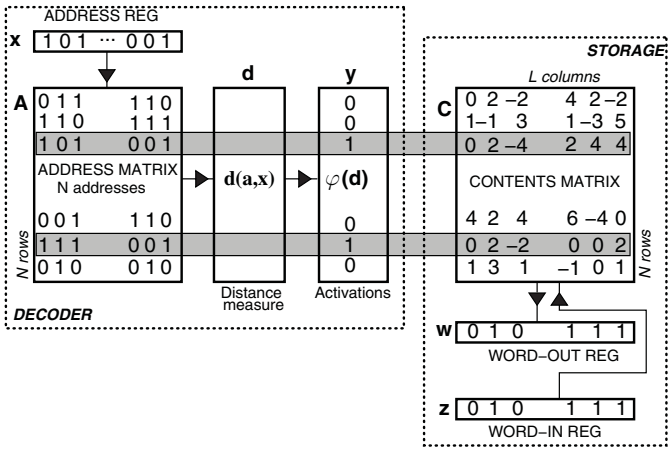

Fig. 7. High-level view of the SDM. The address vector $\mathbf{x}$ is compared with the rows of the address matrix A, for example, with respect to their Hamming distance, resulting in the distance measure vector $\mathbf{d}$. Thresholding $\mathbf{d}$ gives the activation vector $\mathbf{y}$, which is used to activate the corresponding rows in the content matrix C. In the search operation, these rows are summed elementwise, and the result is thresholded yielding the output vector $\mathbf{w}$. When the input vector $\mathbf{z}$ is stored into the memory, the corresponding activated counters on the content matrix rows are incremented or decremented as explained in the text.

memory, whereas the write operation differs in that the value of the counter is not binary and it is also allowed to decrease. It has been shown that five-bit counters with values in $[-16, 15]$ are sufficient for practical operation of the SDM [24]. Reducing the range of the counter reduces the capacity of the memory—SDM works even with one-bit counters but then only the most recently stored data can be retrieved reliably. As we propose implementing the counters by analog memristors, it follows that the capacity of the memristive SDM depends on the multilevel programming capability of the memristors. In particular, the programming rate of the state variable should be approximately constant at a fixed programming voltage pulse, and the number of allowed state variable values should be large. The generic analog memristor model described in Section II satisfies these requirements, as does the physical memristor reported in [38].

If the address register and the word-in register coincide then the SDM functions as an autoassociative memory. This gives rise to improved error-correction, as the word-out vector can be seen as a corrected version of the original input, and can be fed back as the input vector for an iterated search [24]. When the address register and the word-in register do not coincide but have the same length L, the feedback loop from the word-out register to the address register establishes a heteroassociative memory structure capable of storing sequences, as described in Section II-C. Also in this case, there is iterative error-correction; an erroneously begun sequence converges to the stored one [24].

In the rest of this section, we propose an implementation of the SDM using analog memristors, and consider in detail its autoassociative operation, for which the lengths of the address and word-in register both equal L. A major difference between the SDM and Willshaw memory architectures is that the capacity of the SDM is not limited by the input vector length L, because the address matrix yields a sparse activation

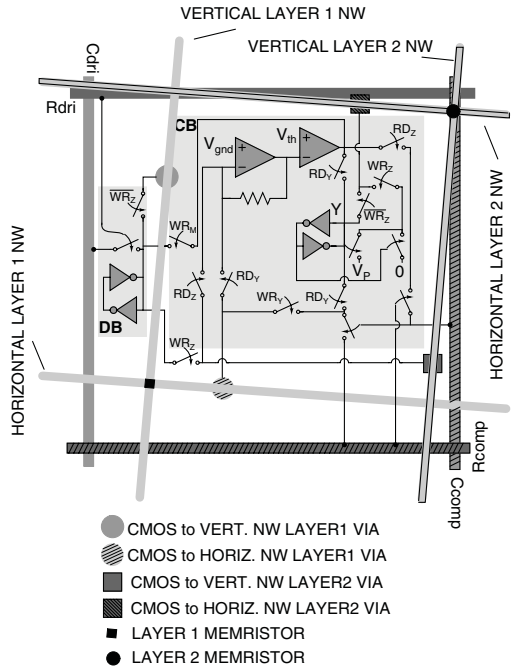

Fig. 8. CMOS cell of the memristive SDM. This layout extends the CMOS cell of Fig. 3(b) by an interface to a second nanowire layer and some required CMOS logic for the operation of the memory architecture. The iterative search of SDM is obtained by closing the switch WR_M and copying the result of the search from the latch Y to the input latch in the DB. Then a new search can be conducted with the updated search vector.

vector whose length N is independent of L, and this activation vector is used as the locations to store the word-in vector in the Willshaw-type content matrix. Thus one can design an SDM for which $N \gg L$, and since the vector capacity M of the SDM is a function of N, it is possible that $M \gg L$. However, M depends heavily on the distribution of the word-in data. When it is dense, the capacity of the SDM is of the order $M = 0.15N$ [30]. On the other hand, when the word-in data and the activation pattern are logarithmically sparse, the capacity is much higher, at least of the order of a Willshaw memory of that size

$$M \approx \frac{0.69LN}{\log(L)\log(N)} \tag{23}$$

the exact number depending among other things on the range of the counters in the content matrix and the required fidelity of the retrieval operation.

C. Implementation of a Memristive SDM

Fig. 8 shows the schematic of our proposed memristive SDM cell. CMOS circuitry is used to implement the address register, the distance vector **d**, the activation vector **y**, the word-out register, and the word-in register. The address and content matrices are mapped to two vertically stacked memristive crossbars, as this layout allows a more efficient utilization of silicon area. The address matrix is mapped to a crossbar array of binary memristors as in the ACAM, and the content matrix is mapped to a crossbar array of analog memristors. Compared with the ACAM implementation shown in Fig. 3, the CMOS cell has been appended with a latch for bitwise

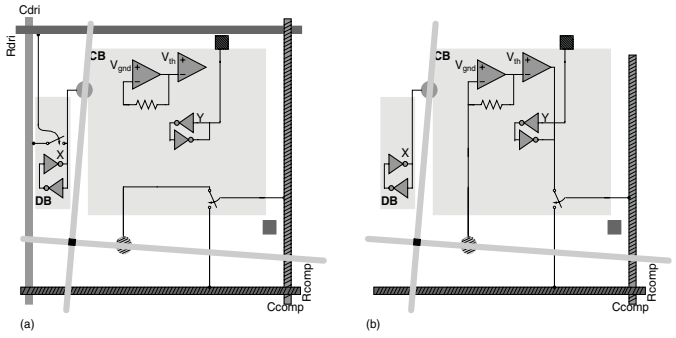

Fig. 9. SDM cell configured as an ACAM for the address matrix search. (a) Storing the input vector as the states of the memristors on a horizontal nanowire. Using this configuration, the input vector bits can also be stored into the latch at the DB. (b) Search operation. In contrast to the configuration depicted in Fig. 4(b), the result of the search is stored to the latch Y.

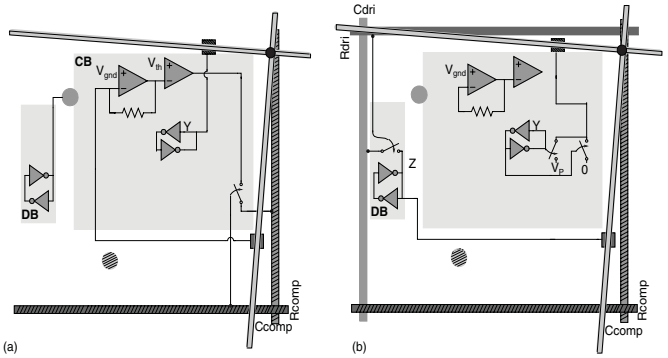

Fig. 10. SDM CMOS cell configured for reading and programming the content matrix. (a) Retrieval operation. The result of the address matrix search is driven from the Y latch to the second-layer horizontal nanowire, while the second-layer vertical nanowire is connected to virtual ground. The currents coming in from the second-layer vertical nanowires are measured and thresholded, and the result of the retrieval operation can be read from the R_{comp} microwire. (b) Incrementing and decrementing the counters in the content matrix. The rows selected for this operation are determined by the result of the address matrix search stored into latch Y. The direction of programming is determined by the bit values Z stored into the latch at the DB. Global square wave voltage signal V_P is used for the programming.

storing of the activation pattern **y**, and with multiple switches for selecting between the use of the address matrix and the content matrix, and furthermore between the write and read phases of the content matrix.

The search of the address matrix is shown in Fig. 9. During this phase, the SDM cell is configured as an ACAM cell, and the search operation is conducted as explained in Section III. In contrast to Fig. 3, the thresholded result that identifies the active rows in the SDM Decoder is stored into the latch Y instead of a direct readout. This result is used to read and write the content matrix. The retrieval of data from the content matrix is depicted in Fig. 10(a). The latched result Y is used to drive the second-layer horizontal nanowire, while the second-layer vertical nanowire is connected to the virtual ground at the input of the operational amplifier. As noted in the previous subsection, the retrieval is achieved by a thresholded matrix product between the activation vector **y** and the content matrix **C**, which in the memristive implementation consists of nonnegative analog values. Thus the inner product method described in Section III-A can be applied here. Notice that the threshold voltage V_{th} should be proportional to the number of

rows selected by the address matrix search, and to count the number of selected rows, additional analog CMOS circuitry needs to be used. For example, each activated CMOS cell can drive a constant current on a global microwire, and by measuring the sum current one obtains the number of activated rows. However, for simplicity we have omitted this part of the CMOS cell circuitry from the schematic of Fig. 8.

Content matrix is written by applying a voltage pulse on those second-layer horizontal nanowires, which are selected by the address matrix search explained above. The direction of the programming is determined by the word-in vector which must be written bitwise to the latches at the DBs in advance. The configuration of the memristive SDM cell during the writing of the content matrix is shown in Fig. 10(b).

D. Simulations and Error Analysis

The main difference in the operation of the proposed implementations of the SDM and the ACAM is the programming of the analog weights representing the SDM counters. The fidelity of the retrieve operation depends on the accuracy of the programming of the analog memristors—the retrieve operation itself is simply implemented by the inner product method described in Section III-A. Simulations of the ACAM cell presented in Section III-D apply here for the search and retrieve operations; the exact power consumption of the cell depends on the resistances of the analog memristors. However, here it should be noted that the activation pattern of the SDM decoder is sparse, which reduces the power consumption as noted for the Willshaw memory in (22). The total area of the transistors in the SDM cell implemented using a 0.13-μm CMOS process is approximately 25 $\times$ 25 μm, from which we estimate that the total area of one cell should not exceed 50 $\times$ 50 μm. In the following, we present a simulation of programming the analog memristor located at the crosspoint of the second-level horizontal and vertical nanowires contacted to the simulated cell. Furthermore, we have simulated the effect of mismatched programming rates of analog memristors on the capacity of the SDM.

In Fig. 11, voltage waveforms during the write operation are shown. Here, Z denotes the voltage at the DB latch and corresponds to the value of a single bit of the word-in vector, Y is the voltage at the CB latch corresponding to a single bit of the activation vector, and V_P is a square wave voltage signal used to program the analog memristors. The voltage signal V_P is propagated onto the second-layer horizontal nanowire only if $Y = 1$, that is, if the corresponding SDM row is selected at the SDM Decoder. If $Y = 0$, the second-layer horizontal nanowire is tied to ground. Moreover, due to an appropriate choice of the memristor's programming threshold, it is programmed only when the polarity of V_P is different from the polarity of Z. Thus the direction of the programming depends on the value of Z, that is, on the value of the corresponding bit of the word-in vector. The amount of change in the state of a memristor when programmed is determined by the amplitude and wavelength of V_P.

The accuracy of the counters depends on the characteristics of the analog memristors. The proposed write operation

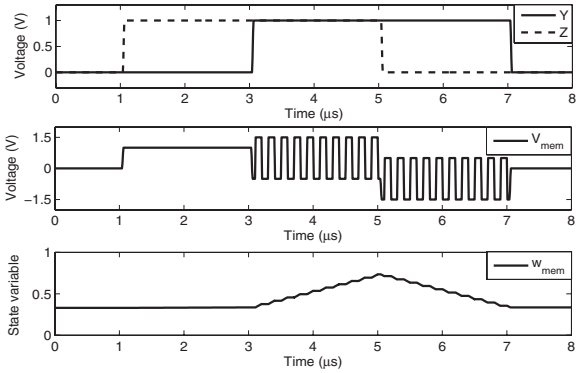

Fig. 11. SPICE simulation of the programming of an analog memristor in the content matrix. Top inset: different combinations of the latch voltages Y and Z. Middle inset: voltage across a simulated analog memristor within the content matrix. Bottom inset: state variable $w \in [0, 1]$ of the memristor during the programming. Programming takes place when the memristor is selected by the activation signal Y, and the direction of the programming depends on the value of the word-in bit Z. Square wave voltage signal V_P used in this simulation had dc value of 0.5 V, amplitude of 1.0 V and frequency of 5 MHz. The generic analog memristor model described in Section II-A is used with parameters $c_1 = 1 \times 10^{-3}$, $d_1 = 1 \times 10^{-2}$, $c_2 = 0.27$, and $d_2 = 10$, corresponding to a programming threshold of approximately $V^T = 1.1$ V at the timescale of 1 μs.

does not guarantee an integer change in the counter value, as the magnitude of the programming step is affected by several nonidealities, including device-to-device mismatch in the programming thresholds among the analog memristors. On the other hand, as the write operation is distributed over multiple second-layer horizontal nanowires corresponding to rows in the content matrix, the individual variations may average out. The advantage of the proposed design is that programming can be performed in parallel for each memristor on a given row, and simultaneously for all rows. If the accuracy of this pulse-based programming is not enough, we propose using cyclic programming discussed in [15] and [42]. However, this programming method requires considerably more complex CMOS circuitry than that presented above. It also requires multiple programming cycles per device, and cannot be easily applied to all memristors in the content matrix simultaneously.

Fig. 12 shows simulation results for the effect of inaccurate programming on the capacity of an SDM. In the simulations, a varying number of dense binary input vectors are stored in the content matrix with $L = N = 2^{11}$ using logarithmically sparse activation patterns, corresponding to the theoretical vector storage capacity $M = 0.15N$. This capacity is defined as the number of vectors storable in the memory with a bit error probability $P_E = 0.005$. Pulse-based programming of the memristors is assumed with 32 nominal states. Errors in the programming phase are approximated by assuming that the magnitude of state change in programming a memristor is drawn from the normal distribution $\mathcal{N}(1, \sigma_P^2)$. These magnitudes are assumed to be different for each memristor in the content matrix, but fixed for a given memristor. The state value of each memristor is limited to $[-16, 15]$; note that due to the random variation in the programming, not all devices have exactly 32 distinct states within this programming interval. This simulation indicates that the SDM is not very sensitive

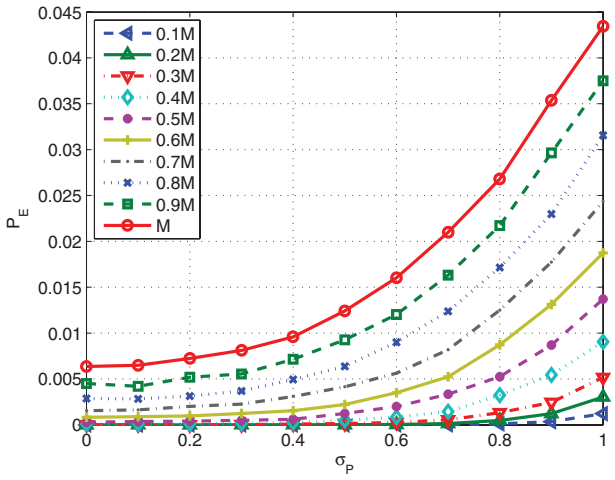

Fig. 12. Probability of bit error P_E versus standard deviation σ_P of the programming error in the SDM content matrix, simulated for $L = N = 2^{11}$, nominal memristances integer-valued in $[-16, 15]$, and number of stored vectors from $0.1M$ to M, where $M = 0.15N \approx 307$.

to error in programming the counter values; a 10% standard deviation of the programming magnitude has negligible effect on the bit error probability in the content matrix, and a standard deviation of over 80% is required to reduce the vector capacity by a factor of 0.5.

V. Concusion

We proposed memristive implementations of three associative memory architectures: the autoassociative content addressable memory, the Willshaw memory, and the SDM. Our work was motivated by the prospect of low-power memristive CMOL-circuits that allow scaling up the capacity and input word length of these memory architectures compared with software and pure CMOS implementations. We conclude that the proposed memristive SDM implementation provides an area-efficient associative memory circuit whose memory capacity can be dynamically maximized by configuring the memory to operate as an associative CAM with dense stored data, and as an SDM when the data is sparse. However, a full proof of the benefits of the proposed designs will be possible only when more empirical data on the characteristics of CMOL circuits becomes available.

Acknowledgment

The authors would like to thank Prof. B. A. Olshausen, Prof. F. T. Sommer, and Dr. J. K. Poikonen for useful discussions.

References

[1] L. O. Chua, "Memristor-the missing circuit element," *IEEE Trans. Circuit Theory*, vol. 18, no. 5, pp. 507–519, Sep. 1971.

[2] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, pp. 80–83, May 2008.

[3] S. H. Jo and W. Lu, "CMOS compatible nanoscale nonvolatile resistance switching memory," *Nano Lett.*, vol. 8, no. 2, pp. 392–397, Jan. 2008.

[4] M.-J. Lee, C. B. Lee, D. Lee, S. R. Lee, M. Chang, J. H. Hur, Y.-B. Kim, C.-J. Kim, D. H. Seo, S. Seo, U.-I. Chung, I.-K. Yoo, and K. Kim, "A fast, high-endurance and scalable non-volatile memory device made from asymmetric Ta_2O_{5-x}/TaO_{2-x} bilayer structures," *Nature Mater.*, vol. 10, pp. 625–630, Jul. 2011.

[5] J. J. Yang, M. D. Pickett, X. Li, D. A. A. Ohlberg, D. R. Stewart, and R. S. Williams, "Memristive switching mechanism for metal/oxide/metal nanodevices," *Nature Nanotechnol.*, vol. 3, pp. 429–433, Jun. 2008.

[6] S. H. Jo, K.-H. Kim, T. Chang, S. Gaba, and W. Lu, "Si memristive devices applied to memory and neuromorphic circuits," in *Proc. IEEE Int. Symp. Circuits Syst.*, Jun. 2010, pp. 13–16.

[7] S. H. Jo, K. H. Kim, and W. Lu, "Programmable resistance switching in nanoscale two-terminal devices," *Nano Lett.*, vol. 9, no. 1, pp. 496–500, 2009.

[8] L. O. Chua, "Resistance switching memories are memristors," *Appl. Phys. A*, vol. 102, no. 4, pp. 765–783, Mar. 2011.

[9] D. B. Strukov and R. S. Williams, "Four-dimensional address topology for circuits with stacked multilayer crossbar arrays," *Proc. Nat. Acad. Sci. United States Amer.*, vol. 106, no. 48, pp. 20155–20158, Dec. 2009.

[10] K. K. Likharev and D. B. Strukov, "CMOL: Devices, circuits, and architectures," in *Introducing Molecular Electronics*, G. Cuniberti, G. Fagas, and K. Richter, Eds. New York, USA: Springer-Verlag, 2005, pp. 447–478.

[11] G. S. Snider, "Self-organized computation with unreliable, memristive nanodevices," *Nanotechnology*, vol. 18, no. 36, p. 365202, Aug. 2007.

[12] S. George and P. Hasler, "HMM classifier using biophysically based CMOS dendrites for wordspotting," in *Proc. IEEE Biomed. Circuits Syst. Conf.*, Nov. 2011, pp. 281–284.

[13] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognit. Comput.*, vol. 1, no. 2, pp. 139–159, 2009.

[14] G. Snider, R. Amerson, D. Carter, H. Abdalla, M. S. Qureshi, J. Léveillé, M. Versace, H. Ames, S. Patrick, B. Chandler, A. Gorchetchnikov, and E. Mingolla, "From synapses to circuitry: Using memristive memory to explore the electronic brain," *Computer*, vol. 44, no. 2, pp. 21–28, Feb. 2011.

[15] E. Lehtonen, J. Poikonen, M. Laiho, and W. Lu, "Time-dependency of the threshold voltage in memristive devices," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2011, pp. 2245–2248.

[16] G. S. Snider and R. S. Williams, "Nano/CMOS architectures using a field-programmable nanowire interconnect," *Nanotechnology*, vol. 18, no. 3, p. 035204, 2007.

[17] C. J. Chevallier, C. H. Siau, S. F. Lim, S. R. Namala, M. Matsuoka, B. L. Bateman, and D. Rinerson, "A 0.13 μm 64 Mb multi-layered conductive metal-oxide memory," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, Feb. 2010, pp. 260–261.

[18] Y. Choi, I. Song, M.-H. Park, H. Chung, S. Chang, B. Cho, J. Kim, Y. Oh, D. Kwon, J. Sunwoo, J. Shin, Y. Rho, C. Lee, M. G. Kang, J. Lee, Y. Kwon, S. Kim, J. Kim, Y.-J. Lee, Q. Wang, S. Cha, S. Ahn, H. Horii, J. Lee, K. Kim, H. Joo, K. Lee, Y.-T. Lee, J. Yoo, and G. Jeong, "A 20 nm 1.8 V 8 Gb PRAM with 40 MB/s program bandwidth," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, Feb. 2012, pp. 46–48.

[19] T. Kohonen, *Associative Memory*. New York, USA: Springer-Verlag, 1977.

[20] T. Kohonen, *Content-Addressable Memories*. New York, USA: Springer-Verlag, 1980.

[21] M. A. Kramer, "Autoassociative neural networks," *Comput. Chem. Eng.*, vol. 16, no. 4, pp. 313–328, Apr. 1992.

[22] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Nat. Acad. Sci.*, vol. 79, no. 8, pp. 2554–2558, Apr. 1982.

[23] J. D. Keeler, "Comparison between Kanerva's SDM and hopfield-type neural networks," *Cognit. Sci.*, vol. 12, no. 3, pp. 299–329, Mar. 1988.

[24] P. Kanerva, *Sparse Distributed Memory*. Cambridge, MA, USA: MIT Press, 1988.

[25] D. J. Willshaw, O. P. Buneman, and H. C. Longuet-Higgins, "Non-holographic associative memory," *Nature*, vol. 222, pp. 960–962, Jun. 1969.

[26] B. A. Olshausen and D. J. Field, "Emergence of simple-cell receptive field properties by learning a sparse code for natural images," *Nature*, vol. 381, pp. 607–609, Jun. 1996.

[27] R. Baddeley, "An efficient code in V1?" *Nature*, vol. 381, pp. 560–561, Jun. 1996.

[28] E. B. Baum, J. Moody, and F. Wilczek, "Internal representations for associative memory," *Biol. Cybern.*, vol. 59, nos. 4–5, pp. 217–228, 1988.

[29] A. Knoblauch, G. Palm, and F. T. Sommer, "Memory capacities for synaptic and structural plasticity," *Neural Comput.*, vol. 22, no. 2, pp. 289–341, Feb. 2010.

[30] P. Kanerva, "Sparse distributed memory and related models," in *Associative Neural Memories, Theory Implement*, M. H. Hassoun, Ed. London, U.K.: Oxford Univ. Press, 1993, pp. 50–76.
[31] O. Turel, I. Muckra, and K. Likharev, "Possible nanoelectronic implementation of neuromorphic networks," in *Proc. Int. Joint Conf. Neural Netw.*, 2003, pp. 365–370.
[32] K. Eshraghian, K.-R. Cho, O. Kavehei, S.-K. Kang, D. Abbott, and S.-M. S. Kang, "Memristor MOS content addressable memory (MCAM): Hybrid architecture for future high performance search engines," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 8, pp. 1407–1417, May 2011.
[33] Y. V. Pershin and M. D. Ventra, "Experimental demonstration of associative memory with memristive neural networks," *Neural Netw.*, vol. 23, no. 7, pp. 881–886, Jul. 2010.
[34] A. Sinha, M. S. Kulkarni, and C. Teuscher, "Evolving nanoscale associative memories with memristors," in *Proc. 11th IEEE Int. Conf. Nanotechnol.*, Aug. 2011, pp. 860–864.
[35] O. Türel, "Devices and circuits for nanoelectronic implementation of artificial neural networks," Ph.D. dissertation, Dept. Phys. Astron., Stony Brook Univ., Stony Brook, NY, USA, 2007.
[36] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: A tutorial and survey," *IEEE J. Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, Mar. 2006.
[37] K.-H. Kim, S. H. Jo, S. Gaba, and W. Lu, "Nanoscale resistive memory with intrinsic diode characteristics and long endurance," *Appl. Phys. Lett.*, vol. 96, no. 5, pp. 053106-1–053106-3, Feb. 2010.
[38] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu, "Nanoscale memristor device as synapse in neuromorphic systems," *Nano Lett.*, vol. 10, no. 4, pp. 1297–1301, Mar. 2010.
[39] H. J. Mattausch, W. Imafuku, A. Kawabata, T. Ansari, M. Yasuda, and T. Koida, "Associative memory for nearest-hamming-distance search based on frequency mapping," *IEEE J. Solid-State Circuits*, vol. 47, no. 6, pp. 1448–1459, Jun. 2012.
[40] H. J. Mattausch, T. Gyohten, Y. Soda, and T. Koide, "Compact associative-memory architecture with fully-parallel search capability for the minimum hamming distance," *IEEE J. Solid-State Circuits*, vol. 37, no. 2, pp. 218–227, Feb. 2002.
[41] S. Nakahara and T. Kawata, "A design for a minimum hamming-distance search using aynchronous digital techniques," *IEEE J. Solid-State Circuits*, vol. 40, no. 1, pp. 276–285, Jan. 2005.
[42] M. Laiho, E. Lehtonen, and W. Lu, "Memristive analog arithmetic within cellular arrays," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2012, pp. 2665–2668.

Eero Lehtonen received the M.Sc. degree in mathematics and the D.Sc. degree in electrical engineering from the University of Turku, Turku, Finland, in 2006 and 2013, respectively. His doctoral thesis focuses on the use of memristors for computing.

Currently, his research is focused on the application of memristive computing in massively parallel mixed-mode processing architectures.

Jussi H. Poikonen received the M.Sc. and D.Sc. degrees in telecommunications from the University of Turku, Turku, Finland, in 2005 and 2009, respectively.

He was a Special Researcher and a Lecturer with the University of Turku from 2009 to 2011. Since 2011 he has been working as a Post-Doctoral Researcher with the Department of Communications and Networking, Aalto University, Espoo, Finland. His current research interests include simulation and analysis of wireless communication systems, cognitive radio systems, and applications of memristive circuits in signal processing.

Mika Laiho (M'04) received the M.Sc., Lic.Sc., and D.Sc. degrees in electrical engineering from the Helsinki University of Technology, Espoo, Finland, in 1999, 2001, and 2003, respectively.

He started his post-doctoral career with the University of Turku, Turku, Finland, in November 2003. Since 2008, he has been an Adjunct Professor with the University of Turku, where he is currently an Academy of Finland Research Fellow. He has published more than 100 papers in the areas of analog/mixed-mode processor arrays and massively parallel sensing/computing. His current research interests include harnessing emerging memory technologies to computing, especially using locally connected architectures, and associative memory for cognitive tasks.

Pentti Kanerva received the M.Sc. degree in forestry from the University of Helsinki, Espoo, Finland, in 1964, and the Ph.D. degree in philosophy from Stanford University, Palo Alto, CA, USA, in 1984.

He was the Principal Investigator of neural computing projects with the NASA Ames Research Center from 1985 to 1992 and the Swedish Institute of Computer Science, Kista, Sweden, from 1993 to 2002, a Senior Fellow with the Redwood Neuroscience Institute, Berkeley, CA, from 2003 to 2005, and is a Visiting Scholar with UC Berkeley's Redwood Center for Theoretical Neuroscience, Berkeley. His research is aimed at understanding how brains compute, it draws on the mathematical properties of hyperdimensional spaces, and is published in a book and 30 papers.