# A Robust Digital RRAM-Based Convolutional Block for Low-Power Image Processing and Learning Applications

Edouard Giacomin, *Student Member, IEEE*, Tzofnat Greenberg-Toledo, Shahar Kvatinsky, *Senior Member, IEEE*, and Pierre-Emmanuel Gaillardon, *Senior Member, IEEE*

***Abstract*— Currently, there is a growing attention toward developing efficient hardware convolutional blocks for several applications such as computer vision or image processing. Recent works have shown that using binary values in convolutional blocks can considerably reduce the overall power consumption while achieving a high degree of accuracy. In parallel, some works employed resistive random-access memory (RRAM) as an in-memory accelerator to directly store the convolution kernels and perform analog dot product operations in the array, reducing the overall power consumption by limiting the number of memory accesses. However, such architecture is hampered by the limited resistance precision and large intrinsic variability of RRAMs. In this paper, we present a purely digital robust RRAM-based convolutional block using single-ended XNOR sensing capable of performing dot product operations in a single cycle. By carefully considering physical design and RRAM limitations at the 28-nm technology node, we show that at the circuit level, our architecture can tolerate a resistance window as low as 1.09, ensuring reliable operations even under a high RRAM variability ($\sigma/\mu = 25\%$ for a resistance window between both states around 50). When integrated in ISAAC, a state-of-the-art learning accelerator, our block can reduce the power by 2.7× while guaranteeing robust operations.**

***Index Terms*— Resistive memory, binary neural network, convolution, reliability, circuit design, low-power.**

## I. Introduction

CONVOLUTION is a crucial operation in many image processing and computer vision applications. For instance, convolution is widely used in the image processing domain for edge detection [1], erosion and dilation [2], *etc*. In addition, convolution is a key component in machine learning techniques based on *Convolutional Neural Networks* (CNNs) such as image classification [3] or speech recognition [4]. However, convolution is a costly operator since it involves a significant amount of data movement between the computation unit and the memory which can be more expensive than the computation itself [5]. Therefore, architectures using convolutional blocks are generally unsuitable for smaller devices like embedded systems and smartphones due to their high power consumption. Consequently, several energy-efficient hardware accelerators have been proposed [6]–[10], making them more suitable for Internet-of-Things devices.

To further reduce the overall energy consumption of such systems, some recent works [11]–[15] on *Binary Neural Networks* (BNNs) have opened the paths to more efficient convolutional blocks. By using binary values ($-1$ and $+1$) instead of floating-points precision for both the input and kernel matrices, the multiply-accumulate units can be replaced by simple binary XNOR and bitcount blocks, considerably reducing the necessary computational resources and memory transfers while keeping a high accuracy (less than $\approx$10% reduction) compared to the full precision implementations.

In parallel, recent works proposed to use *Resistive Random-Access Memory* (RRAM) [13], [14], [16]–[22] passive crossbars in order to reduce the number of data transfers by storing the filter (in this paper, we will use both kernel and filter denotations) value as RRAM resistance and realize in-memory computation at the same time. Such arrays can be used for analog dot product, the analog output current being a sum of product depending on the RRAM resistances and the input voltages, according to Kirchoff's law. In this way, less data movements are required since there is no need to move the filter values to the computation unit. However, such architecture is greatly limited by the technology of RRAM, which suffers from device to device variability [23] or sneakpath current when reading [24] and limitations from the architecture such as the high-precision current sensing requirement [25] or noise [26]. In addition, most of recent works mainly focused on the architectural level while physical design and realistic array assumptions were not fully considered.

Manuscript received May 15, 2018; revised August 24, 2018; accepted September 16, 2018. This work was supported in part by the United States–Israel Binational Science Foundation under Grant 2016016, in part by The University of Utah SEED Fund under Grant 10044706, in part by the Technion Computing Engineering Center through the Viterbi Fellowship, and in part by The University of Utah's Florian Solzbacher and Xiaoxin Chen Graduate Fellowship. This paper was recommended by Associate Editor P. K. Meher. *(Corresponding author: Edouard Giacomin.)*

E. Giacomin and P.-E. Gaillardon are with the Laboratory for NanoIntegrated Systems, Department of Electrical and Computer Engineering, The University of Utah, Salt Lake City, UT 84112 USA (e-mail: edouard.giacomin@utah.edu; pierre-emmanuel.gaillardon@utah.edu).

T. Greenberg-Toledo and S. Kvatinsky are with the Viterbi Faculty of Electrical Engineering, Technion—Israel Institute of Technology, Haifa 32000, Israel.

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.2018.2872455

In this paper, we address those challenges by proposing a convolutional engine that performs digital dot products between binary input vectors and binary local weights without relying on an analog sum of currents. By using RRAMs and binary values, our engine can be integrated into BNNs, greatly reducing the overall energy while ensuring a high accuracy. Compared to previous works [13]–[15] and to the best of our knowledge, we propose for the first time to fully digitalize the RRAM-based dot product and show that it is capable of robust operations at the circuit level. We thoroughly validated our convolutional engine at the circuit and architecture levels by using a 28 nm *Fully Depleted Silicon On Insulator* (FDSOI) technology node. The contributions of this work are:

- Our design can tolerate a resistance window as low as 1.09, ensuring reliable operations even under a high RRAM variability ($\sigma/\mu = 25\%$ for a resistance window between both states around 50);
- Our architecture can greatly reduce the energy, up to 2.23× when compared to a conventional *Complementary Metal Oxide Semiconductor* (CMOS)-based *Multiply Accumulate* (MAC) for a standard image processing application;
- Our block can be used to accelerate the inference phase of ISAAC, a state-of-the-art hardware accelerator with in-situ analog arithmetic in crossbars [16], reducing the power consumption by 2.7×.

The rest of this paper is organized as follows: Section II provides some technical background about RRAM technology, binary convolution and related work. Section III presents our proposed binary convolutional block with its array organization as well as the bitcount circuit. Section IV shows experimental results at the circuit and architectural levels. Section V concludes this paper.

## II. Background

In this section, we first present the necessary background about RRAM technology and RRAM dot product. We also introduce some generalities about binary convolution and we briefly review recent works about RRAM-based BNNs.

### A. RRAM Technology

RRAM bitcells are two-terminal devices consisting of metal electrodes and a switching oxide stack [27]. By applying a programming voltage (denoted $V_{prog}$) between the two electrodes, the conductivity of the metal oxide can be changed leading to a switching event of the device between two stable resistance states: a *Low Resistance State* (LRS) and a *High Resistance State* (HRS), allowing them to store a logic binary data 1 or 0 respectively. Applying a positive $V_{prog}$ will induce a switching from HRS to LRS called a *set* process. On the other hand, applying a negative $V_{prog}$ will induce a switching from LRS to HRS called a *reset* process. In addition to the resistive property, a RRAM also introduces a parasitic capacitance $C_p$ between both electrodes [28], as shown in Fig. 1 (a). While $C_p$ will be considered in the simulations, we omit its device representation in the rest of this paper for clarity. Thanks to the filamentary conduction mechanism, the LRS

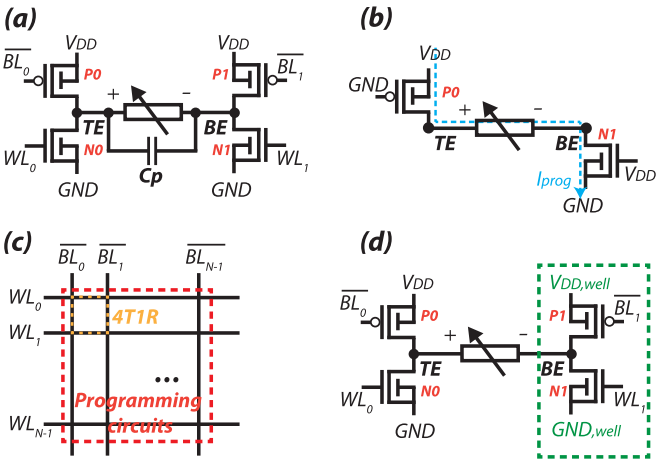



Fig. 1. (a) Schematic of the 4T1R structure; (b) 4T1R structure under a *set* process; (c) Memory bank control lines organization; (d) Deep N-well organization.

can have a wide range of values, allowing RRAM to be used as multilevel cells and to perform dot product operations in an analog way, as it will be explained in Section II.C. However, RRAMs are subjected to different kind of variability (device to device and cycle to cycle [23]) which can impact their resistances values, greatly reducing their ability to act as precise multilevel cells. In addition, a tampered resistance value can lead to erroneous outputs when using RRAMs for computation. Therefore, they need to be integrated in a tolerant and robust design. To precisely control the LRS value, a programming structure is used to efficiently program the RRAMs and provides a programming current denoted $I_{prog}$. In this paper, we consider the *4Transistors 1RRAM* (4T1R) structure, depicted in Fig. 1 (a), since it provides a larger current density and leads to more area efficient designs [29]. The 4T1R programming structure employs two pairs of *nmos* and *pmos* transistors in order to trigger a *set* or a *reset* process. For instance, if transistors $P0$ and $N1$ are turned *on* through signals $\overline{BL_0}$ and $WL_1$, the voltage across the RRAM device is $V_{prog}$ and triggers a *set* process, as shown in Fig. 1 (b). As depicted in Fig. 1 (a), the *nmos* and *pmos* programming transistors are controlled by word line ($WL_i$) or bit line signals ($\overline{BL_i}$) respectively. As such, when using several 4T1R programming structures, a memory bank organization can be used to route the signals, using parallel word lines and bit lines, as illustrated in Fig. 1 (c). In order to provide a large programming voltage range (in this paper, we consider $V_{prog} = 2 * V_{DD}$) without using I/O transistors (which occupy more area and introduce higher capacitance), a deep N-well can be used [28], as illustrated in Fig. 1 (d). In this example, a part of the 4T1R programming structure is driven by a constant voltage domain $V_{DD}$ and $GND$. The other part is placed on a deep N-well and is driven by a switchable voltage domain $V_{DD,well}$ and $GND_{,well}$. As such, the voltage across the RRAM device during a *set* or *reset* operation can reach $V_{prog}$ and $-V_{prog}$ respectively by switching $V_{DD,well}$ and $GND_{,well}$ to the proper values, while the voltage among each transistor terminal is at most $V_{DD}$. This eliminates any breakdown risk or reliability degradation on

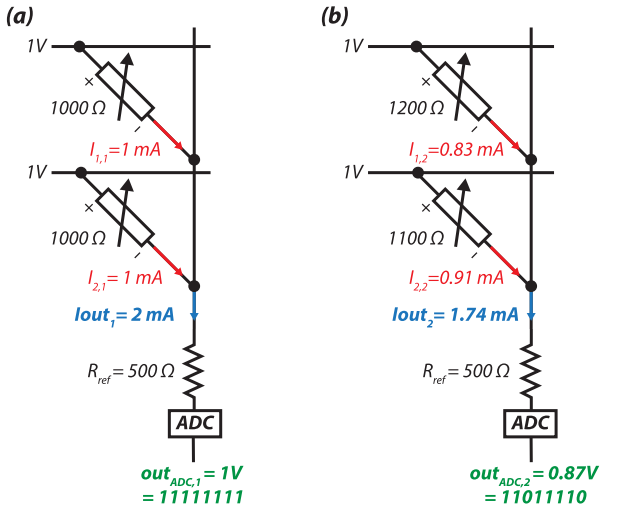



Fig. 2. RRAM-based analog dot product: (a) Ideal; (b) In the presence of variations.

the transistors. More details about the use of deep N-wells and the programming schemes using RRAMs can be found in [28].

### B. Limitations of Analog Dot Product With RRAMs

Besides being used as memory elements, RRAMs can be arranged into crossbar passive arrays (without any access transistors) to perform analog dot products [30], as illustrated in Fig. 2 (a): the output currents ($I_{out_i}$) are the sums of the currents flowing through each RRAM cell and each current is itself the product of the input voltage times the conductance of the RRAM. Due to the high theoretical density of RRAM, this can lead to great benefits in term of area [16], [31]. However, such analog dot product array faces several fundamental limitations: (i) It requires the use of *Analog-to-Digital Converters* (ADCs) and *Digital-to-Analog Converters* (DACs) which are area and power hungry; (ii) The continuous-valued RRAM resistances are hard to control due to process variation [25], especially for passive array when no programming structure is used. We illustrate such kind of situation in Fig. 2 (b) where we assume that the resistance values are not ideal but have suffered from process variation (more than 20% from their nominal value, as reported in literature on real devices [32]). In particular, we assume the resistance values shifted from their nominal value of 1000Ω to 1200Ω and 1100Ω respectively. As shown in Fig. 2 (b) it can alter the final current value ($1.74mA$ instead of $2mA$), potentially leading to incorrect values at the ADC output. In the example, we consider an 8-bit ADC and a 1V reference voltage, as it is the case in the well accepted ISAAC architecture [16]. The first ADC outputs 1V (11111111 in 8-bit binary format) while the second ADC output is 0.87V (11011110) due to RRAM variations, leading to possible erroneous computation (for instance, in the case of a machine learning accelerator such as ISAAC, it has been reported that RRAM variations can cause an error rate distribution of up to 14.5% [33]). These limitations can be partially alleviated when the convolution is performed in a binary way, since the RRAMs only require two distinct resistance states. In that case [13], the summation results in a binary value through a *Comparator Sense Amplifier* (CSA). Nevertheless, CSAs still require a large area and are also subjected to failure due to the offset voltage induced by process variations [34]. To overcome these limitations, we will show how the convolution can be performed in a fully digital way through XNOR and bitcount operations, as explained in Section II-C.

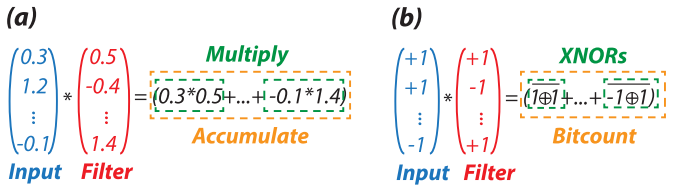



Fig. 3. Convolution between two vectors with: (a) Real values; (b) Binary values (−1 or +1).

### C. XNOR-Based Binary Convolution

A convolution between two real value matrices (a filter and an input matrix) consists of a repeated shift operation which moves the filter over the input and dot product which performs element-wise multiplications between the filter and the input. On Fig. 3 (a), we show a convolution between two vectors (which is a dot product) using real values. The dot product is the sum of products between the elements of the two vectors. As shown in Fig. 3 (b), by constraining the input and the filter matrice values to binary values (−1 or +1), the sum of products is replaced by XNOR and bitcount operations, as explained in [11]. Using binary values can considerably reduce the complexity of the convolution with up to 58× speed-up (in terms of number of the high precision operations) and up to 32× less memory usage [11]. As a result, it can considerably improve the overall performances of CNNs where convolutions account for over 90% of the total number of operations [35]. Note that those benefits are achieved while ensuring a high accuracy (less than ≈10% accuracy reduction compared to the full floating-point implementations [11]). More information about how to digitize a network can be found in [11], [12], and [36].

### D. Related Work

As the computational complexity of CNNs keeps increasing, several approaches have been proposed to optimize their energy efficiency and make them suitable for embedded devices. Recent works [6]–[9] proposed to improve the data-movement flow by increasing the data reuse and avoiding redundant memory accesses, which have a huge contribution in the total energy consumption. Kim *et al.* [10] proposed a methodology to tolerate bit errors resulting from aggressive memory voltage scaling in neural networks and thus, reducing the total energy of the network. In parallel, the use of RRAM devices to further improve the energy efficiency of CNNs has shown great interest in the past few years. Shafiee *et al.* [16] proposed to use RRAM devices to store the weights and perform an analog dot product. However, as stated in Section II.B., the resistance precision of RRAM is limited by the use of ADCs and DACs to interface the analog RRAM array to the digital peripheral circuits. In addition,

those components consume a lot of power and area, mitigating the benefits of such implementation. Furthermore, recent works [13], [14] proposed to perform dot products through RRAMs in a binary way with the use of CSAs or reduced precision ADCs by only using two bit-levels RRAM cells, improving both the energy efficiency and the robustness of their implementations against RRAM process variations. However, they did not study design issue such as the offset voltage of the *Sense Amplifiers* (SAs), potentially leading to operational failure [34]. A recent work [15] proposed a parallel XNOR-RRAM array using CSAs. They showed that by carefully partitioning the RRAM-array, the SA offset issue was alleviated and their proposed architecture was robust against CMOS and RRAM process variations. However, they assumed very optimistic RRAM resistance variation (4.5% where some work reported around 20% of resistance variation [32]) for their study, possibly resulting in an operation failure under realistic assumptions. Moreover, most of the recent works either did not investigate the impact of RRAM process variations at the circuit level [14], [16], [20]–[22] or mainly focused on the architectural level and they lack of proper circuit level evaluations [13], [14], [16]. In addition, some circuit level assumptions were not considered such as the sneakpath current when reading or how to program the RRAM array. This paper intends to fill those gaps. In this work, we show how to perform a dot product in a fully digital manner through XNOR operation without the need to sense the output current. In addition, we provide clear circuit level evaluations for area, energy and robustness under RRAM and CMOS variations of our proposed binary dot product engine.

## III. Proposed Binary Convolutional Engine

In this section, we introduce our implementation of a RRAM-based binary convolution engine. First, we present some generalities about our binary convolutional engine. Then, we describe the working principle of the RRAM-based XNOR cell as well as its array organization. Finally, we present the external bitcount circuit design. Note that the proposed engine is only used for inference.

### A. Generalities

Since a binary dot product can be performed by XNOR and bitcount operations, we develop an RRAM-based XNOR array capable of performing a dot product in a single cycle. A convolution between two matrices resulting in an output matrix in which each element is a dot product, our engine can realize a convolution sequentially where each step consists of a binary dot product. Our array performs in-memory computation: the binary kernel values are stored in the RRAMs as resistance values and XNOR operations are performed between the kernel $K_i$ and the input vector $A$. In that way, our proposed block is capable of performing a fully binary dot product and does not rely on an analog sum of currents. By storing the filter values in the RRAMs, the number of data movements is reduced, greatly decreasing the overall energy consumption as it will be demonstrated in Section IV.

Fig. 4 (a) depicts the general structure of our convolutional engine ($*$ indicates a dot product). As mentioned before,

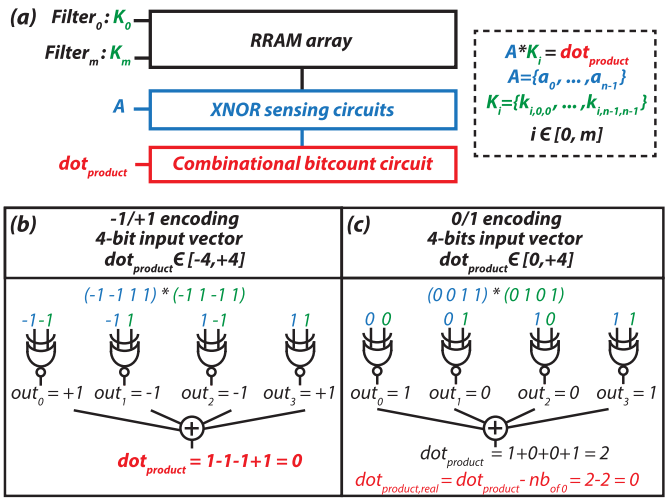



Fig. 4. (a) General structure of the proposed RRAM-based XNOR convolution block engine; Dot product operation with: (b) $-1$ and $+1$ encoding; (c) 0 and 1 encoding.

the binary kernel values (vectors $K_i$), that need permanent storage and that do not need frequent writing, are stored in the RRAMs as binary values in each row. That means that the RRAM array requires $m$ rows if $m$ different binary filters need to be stored in the array. By selecting the appropriate filter (vector $K_i$) and applying the input matrix binary values (vector $A$) as input voltages to the array, the dot product between $A$ and $K_i$ is performed through XNOR and bitcount operations. As explained in Section II-C, in binary convolution, each input and kernel value is either $-1$ or $+1$. In our design, we transpose the $-1/+1$ encoding into 0/1 to allow easy digital implementation. However, when using $-1$ and $+1$ encoding, all the values are summed to obtain the final dot product which will be between $-n$ and $n$ if the input vectors are of size $n$, as shown in Fig. 4 (b). With 0 and 1, the output is necessarily positive and will be between 0 and $n$. Therefore, the bitcount operation is modified through a combinational bitcount circuit (described in Section III-D) to include a negative bias, as depicted in Fig. 4 (c). This negative bias, denoted as $nb_{of\ 0}$ in the figure, is the number of 0 in the dot product since $-1$ is encoded into 0.

### B. RRAM-Based XNOR Cell Working Principle

Based on the circuit topologies presented in [37] and [38], we introduce a RRAM-based XNOR cell. As illustrated in Fig. 5 (a), it is composed of three parts: a precharge circuit that loads both outputs to $V_{DD}$ before the reading phase, a programming circuit to write the appropriate $k$ value to the RRAMs and the read-out circuitry to perform the XNOR operation between $a$ (the input matrix value) and $k$ (the kernel matrix value, stored as the RRAM resistance). In our proposed RRAM-based XNOR bitcell, the bottom part is enclosed in a deep N-well in order to use high programming voltages of $2 * V_{DD}$, as explained in Section II.A. The XNOR bitcell has two modes of operation: a kernel store phase, during which the RRAM cells are programmed with the appropriate kernel value and a computing phase during which the XNOR operation is performed.

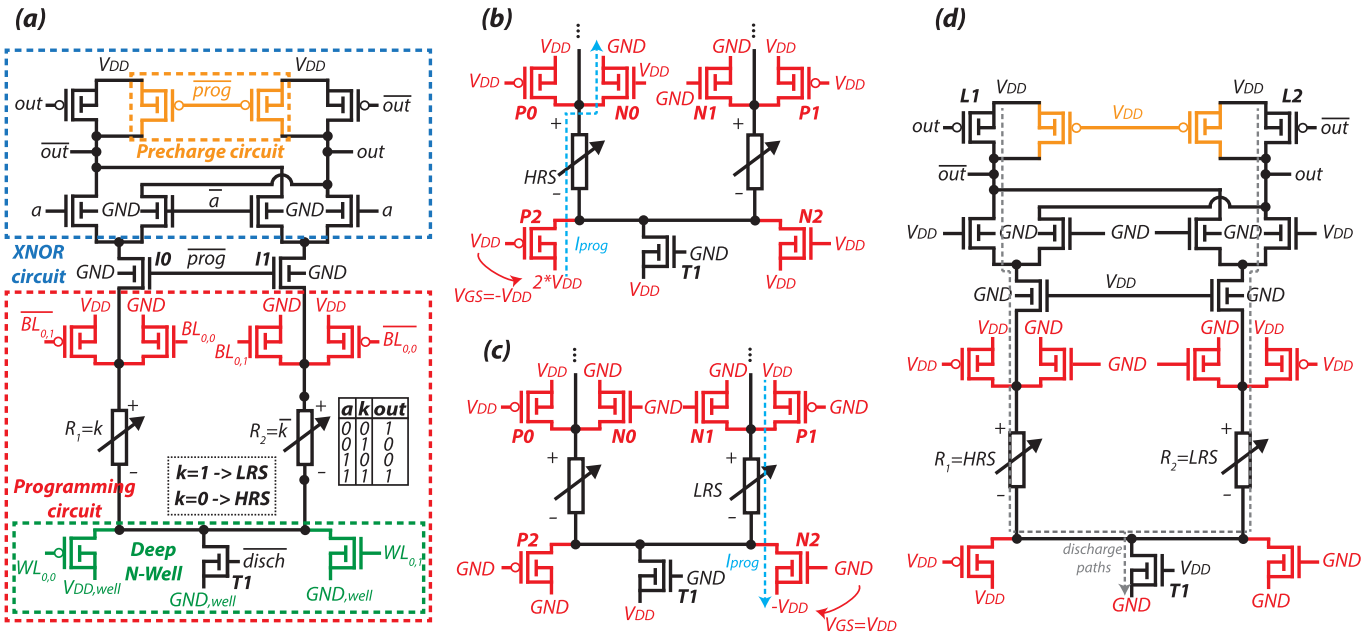



Fig. 5. RRAM-based XNOR cell: (a) general structure; (b) during a *reset* process of $R_1$; (c) during a *set* process of $R_2$; (d) during the computing phase.

*1) Kernel Store Phase:* During the programming phase ($prog = 1$ and $\overline{disch} = 0$), nodes *out* and $\overline{out}$ are precharged to $V_{DD}$ and both RRAMs are isolated from the XNOR circuit through transistors $I0$ and $I1$ and are programmed with a 4T1R programming structure. In addition, signal $\overline{disch}$ is set to $GND$ so that transistor $T1$ is $off$. The value of $k$ will be stored as the RRAM resistances in a complementary fashion. For instance, when $k = 0$, $R_1$ is programmed to HRS and $R_2$ is programmed to LRS. $R_1$ is programmed to HRS by turning *on* transistors $N0$ and $P2$ and turning $off$ transistors $P0$, $P1$, $N1$ and $N2$, as depicted in Fig. 5 (b). During the *reset* process of $R_1$, $V_{DD,well}$ and $GND_{,well}$ are switched to $2 * V_{DD}$ and $V_{DD}$ respectively. As such, the voltage difference across $R_1$ is $-V_{prog} = -2 * V_{DD}$, triggering a *reset* process. In a similar manner, $R_2$ is then programmed to LRS by turning *on* transistors $P1$ and $N2$ and turning $off$ transistors $P0$, $P2$, $N0$ and $N1$, as illustrated in Fig. 5 (c). If we consider $k$=*1*, $R_1$ is programmed to LRS and $R_2$ is programmed to HRS in a similar way.

*2) Inline Kernel Computing Phase:* The computation of an XNOR between an input $a$ and a kernel $k$ stored in the memory is performed as a single memory readout thanks to the XNOR sense amplifiers. The read sequence goes as follows: first, $V_{DD,well}$ and $GND_{,well}$ are switched to $V_{DD}$ and $GND$ respectively. We set $prog = 0$ and $\overline{disch} = 1$ hence nodes *out* and $\overline{out}$ are grounded through the RRAMs and transistor $T1$, as shown in Fig. 5 (d). During this phase, the complementary resistances of the RRAMs modulate the discharge current (the discharge paths are highlighted in gray): the branch with the RRAM in LRS is discharged faster than the branch with the RRAM in HRS. Once *out* or $\overline{out}$ reach the voltage $V_{DD} - V_{T,\text{pmos}}$ ($V_{T,\text{pmos}}$ denotes the threshold voltage of transistors $L1$ and $L2$ in Fig. 5 (d)), it turns *on* the associated *pmos*, pulling up the other node ($\overline{out}$ or *out*) to $V_{DD}$. The resulting output, depending on $a$ and the value stored in $k$ implements an XNOR. Fig. 5 (d) illustrates an example with $a = 1$ and $k = 0$ (so $R_1$ is in HRS and $R_2$ is in LRS). Since $R_2$ is in LRS, the right branch will discharge faster than the left one. Hence, transistor $L1$ will be turned *on* in first, setting node $\overline{out}$ to $V_{DD}$, resulting in a correct XNOR operation. The experimental results section will demonstrate that such structure presents a large tolerance to RRAM variability since the discharge operation phase is successful even for very small HRS/LRS ratio.

### C. RRAM-Based XNOR Array Organization

From the bitcell previously described, it is possible to split it and organize several cells into a matrix. Fig. 6 (a) shows a $m \times l$ RRAM matrix array, where $m$ is the number of the different filters to be stored and $l$ is the number of elements of the output matrix (as an example, storing 10 3×3 filters would require a 10×9 array). From the proposed bitcell structure of Fig. 5 (a), all the XNOR part is shared between all the *4 Transistors 2 RRAMs* structures in each pair of column and is put at the bottom acting as a digital sense amplifier. Similarly, the programming circuits are shared at each column and row to be able to individually address all the RRAMs. Sharing the programming transistors also allows parallel programming. In addition, the sneakpath current issue is alleviated since only one row is selected at a time through signal $En_x$ ($x \in [0, m-1]$). As depicted in Fig. 6 (b), a convolution between an input feature map matrix and a 2D kernel matrix leads to an output feature map matrix with $n^2$ elements. Each 2D kernel is unrolled into a 1D vector so the $n^2$ kernels values are encoded in the $n^2$ pairs of RRAMs (here, $n^2$ is the number of columns in the array of Fig. 6 (a) so $n^2 = l - 1$). Element $o_{i,i}$ is obtained by performing the dot product between the appropriate input matrix values coming

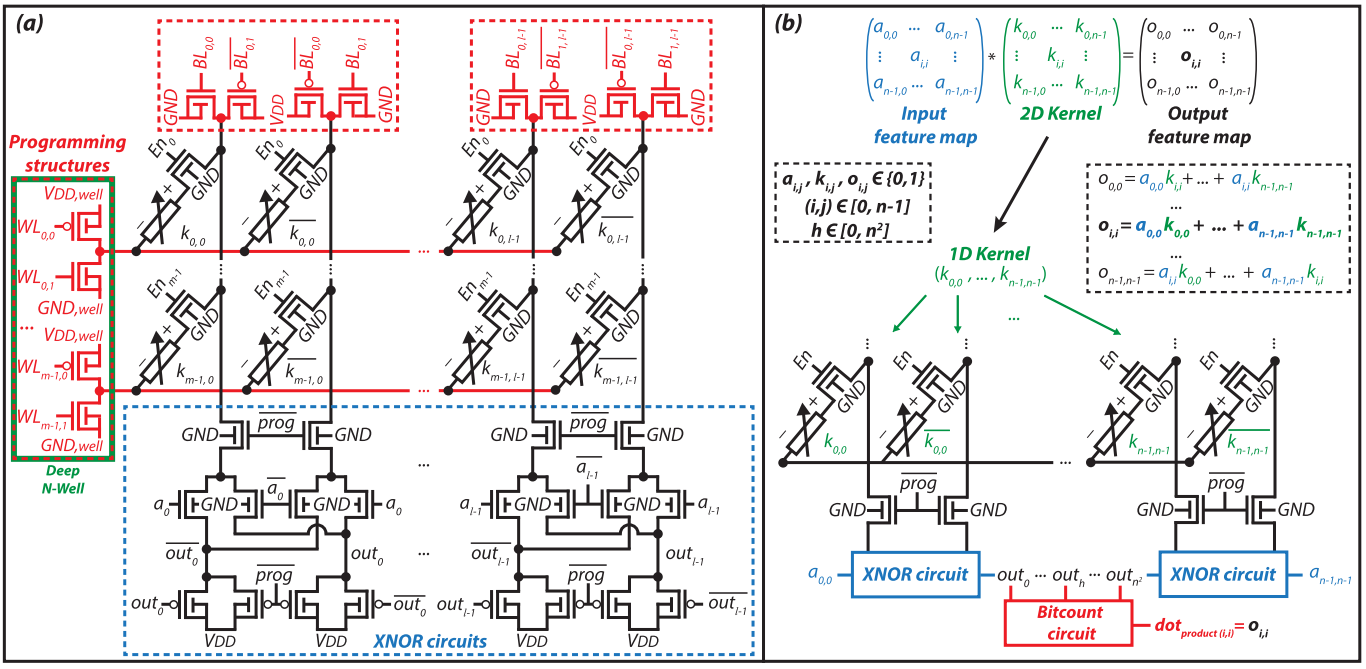
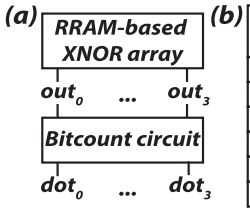



Fig. 6. (a) RRAM-based XNOR $m \times l$ array organization; (b) Binary dot product example ($o_{i,i}$ is computed) between two $n \times n$ matrices.




| Number of 0 in {out$_0$... out$_3$} | | Dot product value {dot$_0$... dot$_3$} | |
|---|---|---|---|
| Decimal | Binary | Decimal | Signed Binary |
| 0 | 000 | 4 | 0100 |
| 1 | 001 | 2 | 0010 |
| 2 | 010 | 0 | 0000 |
| 3 | 011 | -2 | 1110 |
| 4 | 100 | -4 | 1100 |

Fig. 7. (a) General structure of the bitcount circuit; (b) Encoding and final dot product value for a 4-bit input vector.

from the memory $(a_{0,0}, \ldots, a_{n-1,n-1})$ and the kernel values $(k_{0,0}, \ldots, k_{n-1,n-1})$ stored in the RRAMs by selecting the appropriate row through signal $En_x$, as depicted in Fig. 6 (a). Each signals $out_h$ ($h \in [0, n^2]$) are then summed together through an external circuit to perform the bitcount operation, as explained in section III-A. The convolution is therefore performed in $n^2$ steps since the output is a $n^2$-elements matrix.

### D. Combinational Bitcount Circuit

As explained in Section III-A, we need to count the number of 0 to evaluate the final output of the dot product. The bitcount circuit takes the $l$ signals ($out_i$) from the XNOR array as inputs and produces a signed value on $\lfloor \log_2(l) + 2 \rfloor$ bits (+1 for the signed bit), depending on the number of 0. Since the bitcount circuit outputs only depend on the number of 0, so the bitcount circuit inputs, it is fully combinational. Instead of explaining the general behavior of our circuit for $l$ inputs, we provide an example for four inputs (so four signals $out_i$ coming from the RRAM-based XNOR array), as illustrated in Fig. 7 (a). Since the bitcount circuit receives four inputs, the final output is between $-4$ and $+4$. Therefore, four bits are required to represent all the possible values as a signed binary number, as listed in Fig. 7 (b).

## IV. Experimental Results

In this section, we demonstrate the benefits of our RRAM-based XNOR architecture. We first show the energy benefits and robustness of our architecture at the circuit level and then benchmark the performance in terms of energy saving and power consumption reduction at the architectural level by evaluating two cases: (i) A convolution in the context of a fundamental image processing operation: *dilation*; (ii) The integration of our convolutional block in a state-of-the-art RRAM-based hardware accelerator: ISAAC [16].

### A. Circuit Level Evaluation

*1) Experimental Methodology:* Transient simulations have been performed using a 28nm FDSOI design kit. We considered the Stanford RRAM compact model [39] with programming voltages $V_{set} = |V_{reset}| = 0.9V$. The lowest achievable resistance in LRS is $1.7k\Omega$ while the highest resistance in HRS is $23M\Omega$. The parasitic capacitance $C_p$ of a RRAM device is estimated to be $39aF$ by assuming that the RRAMs are sandwiched between $MET1$ - $MET2$ vias in the considered technology. Logic and programming transistors are minimum sized. They have a nominal supply voltage $V_{DD} = 1V$. Delay and power results are extracted from Eldo simulations. In this work, we use a high programming voltage $V_{prog} = 2 * V_{DD}$, as discussed in Section III-B.

*2) RRAM-Based XNOR Operator Transient Simulation:* First, we considered a $128 \times 128$ RRAM-based XNOR array to test its functionality and show that large arrays can be considered. Fig. 8 (a) shows the waveforms of the XNOR output signal ($out_0$) for different input combinations ($a_0$ and $k_0$). A first phase (not shown in the figure) was dedicated to program the RRAMs sequentially (although some parallelism is possible by programming RRAM cells which are column

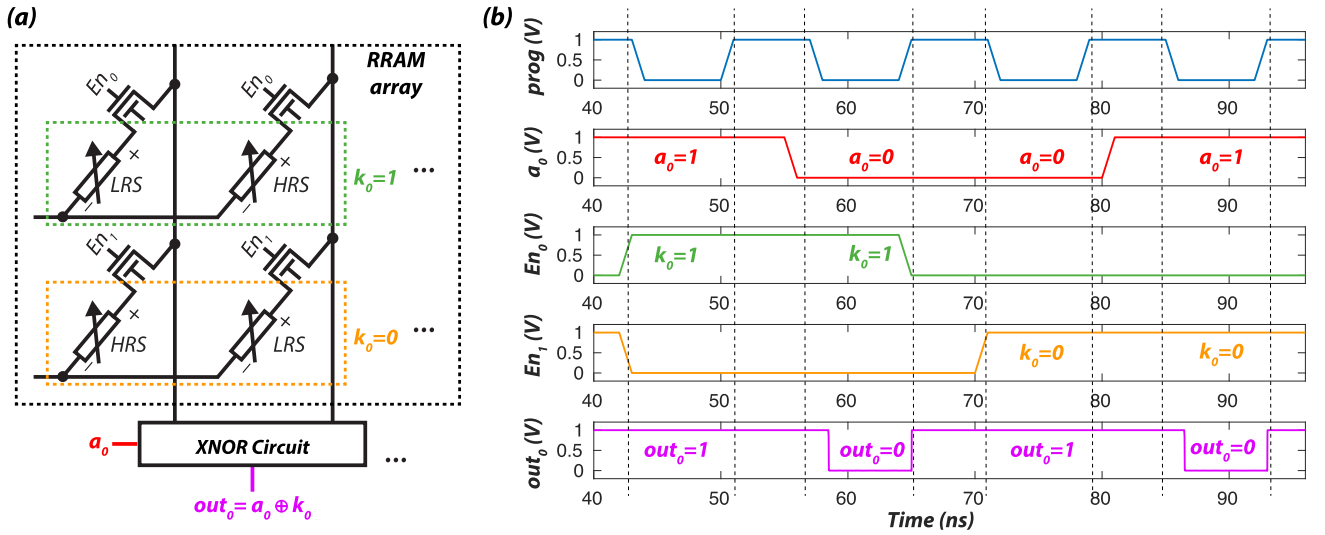



Fig. 8. (a) Simplified schematic of a 2×1 RRAM array (two columns are shown since a single kernel is encoded through two columns), omitting the programming transistors. (b) RRAM-based XNOR 128×128 array functionality waveforms for different XNOR input combinations.

and row independent from each other at the same time). Then, the operating phase is conducted, showing that the RRAM-based XNOR cell operates correctly for the different input cases. To be able to show different values for $k_0$ without reprogramming the associated RRAM cell during simulation (for the clarity of Fig. 8 (b)), we used two enable signals ($En_0$ and $En_1$). As depicted in Fig. 8 (b), when $En_0$ is enabled, the left RRAM is in LRS and the right RRAM is in HRS, leading to an equivalent logic value for $k_0$ of 1. When $En_1$ is enabled, the equivalent logic value of $k_0$ is 0.

*3) Bitcount Circuit:* The bitcount circuit organization is depending on the number of columns in the array. For instance, when considering a 128 × 128 array, the number of 0 is at most 128. That means that the dot product value is ranging from −128 to 128. As a result, the bitcount output is a 9-bit signed number. We described the bitcount circuit behavior in Verilog where the number of 0 is counted through combinational logic and the appropriate signed output value is produced. We then synthesized the considered bitcount circuit with Synopsys Design Compiler. The output netlist is fully combinational and contains 476 cells of various types (AND/OR/INV/AOI, *etc.*).

*4) Energy Benefits:* We then evaluated the energy breakdown of our proposed RRAM-based XNOR array and compared it to a standard CMOS XNOR implementation. Energy repartition for both cases are shown in Fig. 9. At the circuit level, our convolutional block consumes more energy than its CMOS counterpart due to the presence of RRAM devices and additional periphery. However, when taking into account the cost of main memory accesses from [40], the total energy of our block is 1.9× lower than the CMOS implementation since it requires half as much memory accesses.

*5) Robustness Analysis:* We then assessed the circuit robustness of our proposed array, considering CMOS transistor variations (provided by the foundry) and supposed RRAM variability. For sake of the capability of our workstation to run large Monte Carlo simulations, we considered a

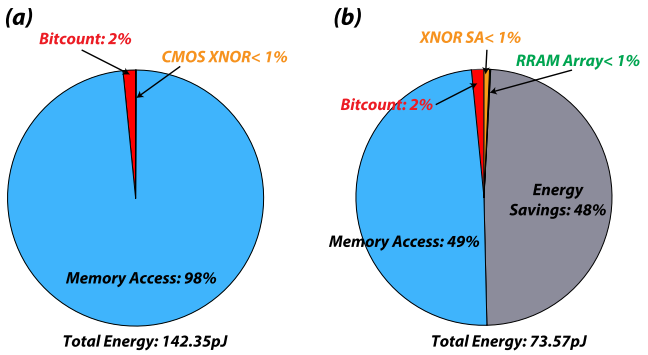



Fig. 9. Energy repartition for: (a) CMOS-based XNOR; (b) RRAM-based XNOR.

25×25 array (sized to store 25 5×5 kernels). We assume that the LRS and HRS follow a Gaussian distribution. The measured nominal values from simulations for LRS and HRS are 284 $k\Omega$ and 14.7$M\Omega$ respectively and are taken as the mean values, denoted by $\mu(LRS)$ and $\mu(HRS)$ respectively. Note that the nominal LRS value is higher than the lowest LRS value achievable by the VerilogA model. This is due to the fact that the programming parameters (programming voltage and current) are not optimal since we used minimum sized programming transistors to save area and the access transistors imply a tension drop at the top electrode of the RRAMs. We study the robustness of our array for different values of the standard deviations $\sigma(LRS)$ and $\sigma(HRS)$ by running Monte Carlo simulations (1000 points for each variation value), as shown in Fig. 10. This results in zero XNOR operation failure on the array for $\sigma(LRS) = \sigma(HRS) \leq 25\%$, demonstrating the robustness of our proposed architecture. This is due to the fact that our structure does not require a high HRS/LRS ratio (1.09 in our case) to behave correctly. We believe that a $(\sigma/\mu)$ ratio of 25% is above the variability of state-of-the-art RRAM technology, as it is reported in [32]. This shows that our circuit can tolerate at 100% the RRAM variability.

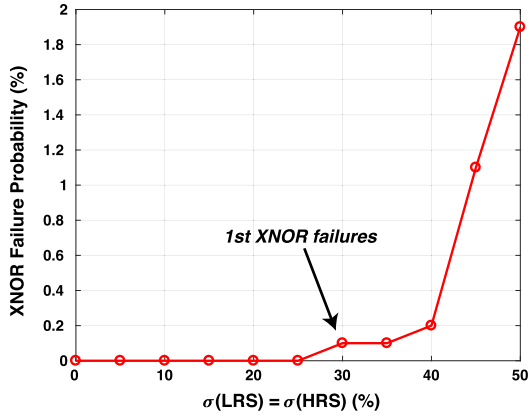



Fig. 10. RRAM-based XNOR failure probability over 1000 Monte Carlo runs for different standard variation on both LRS and HRS.

TABLE I
ENERGY CONSUMPTION (LOGIC COMPUTATION AND MAIN MEMORY ACCESSES) FOR A DILATION BETWEEN AN INPUT IMAGE OF SIZE (512×512) AND DIFFERENT KERNELS FOR THE THREE DIFFERENT BLOCKS

| | Kernel Size | Energy $(mJ)$ | Delay $(s)$ | Energy$^{(1)}$ Overhead | Delay$^{(1)}$ Overhead |
|---|---|---|---|---|---|
| **CMOS MAC** | 3x3 | 7.50 | 0.117 | 2.12× | 2.10× |
| | 5x5 | 21.31 | 0.368 | 2.20× | 2.45× |
| | 7x7 | 42.01 | 0.746 | 2.22× | 2.57× |
| | 11x11 | 104.13 | 1.878 | 2.23× | 2.59× |
| **CMOS XNOR** | 3x3 | 5.69 | 0.063 | 1.60× | 1.12× |
| | 5x5 | 15.67 | 0.169 | 1.61× | 1.13× |
| | 7x7 | 30.64 | 0.335 | 1.62× | 1.15× |
| | 11x11 | 75.56 | 0.821 | 1.62× | 1.14× |
| **RRAM XNOR** | 3x3 | 3.54 | 0.056 | | |
| | 5x5 | 9.70 | 0.150 | | |
| | 7x7 | 18.95 | 0.290 | | |
| | 11x11 | 46.7 | 0.726 | | |

(1) When compared to RRAM XNOR.

### B. Application: Image Processing Dilation

*1) Methodology:* In this section, we evaluated the energy of a fundamental image kernel application using convolution: *dilation*. We used Gem5 [41] for system simulation on a ARMv7-A architecture which is widely used in mobile devices. In our analysis, we consider three cases: (i) A standard 16-bit CMOS MAC unit; (ii) A standard CMOS XNOR circuit; (iii) Our proposed RRAM-based XNOR convolutional block. We studied the estimated cost of a convolution for an input image of size 512×512 pixels using the three different blocks. C programs have been written and fed to Gem5 to emulate the convolution between the input image and the filters in order to get the number of memory accesses between the main memory and the processor. Energy consumption per bit for dual data rate memory accesses are extracted from [40]. Energy consumption values for the logic computations have been extracted from HSPICE simulations [42] at the circuit level for the RRAM XNOR case and the energy for the CMOS XNOR circuit is extracted from Design Compiler, both using the 28nm FDSOI design kit. The energy of the CMOS MAC unit is taken from [43].

*2) Energy Benefits:* Table I presents the energy consumption for the three cases for the input image and dilation filters of different size : 3×3, 5×5, 7×7 and 11×11, which are commonly used in binary image processing applications [44]. The proposed RRAM-based XNOR architecture can save up to 2.23× of energy and the delay is reduced by up to 2.59× when compared to conventional MAC operators since MAC operators are more costly than standard CMOS gates and that weights are already stored in the RRAMs so less memory accesses are required.

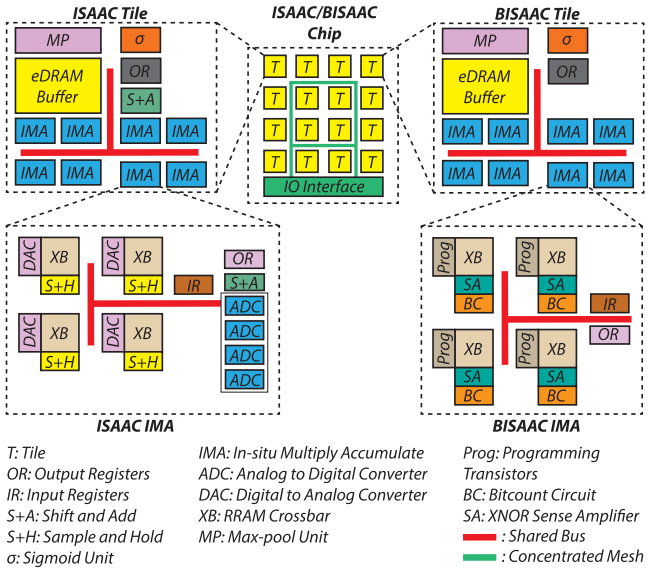
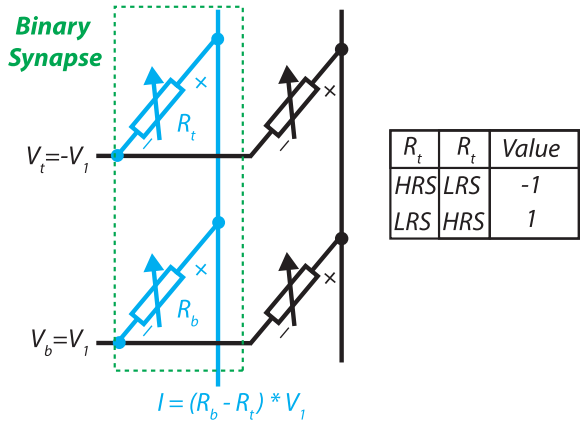



Fig. 11. ISAAC and BISAAC respective hierarchies.

### C. BISAAC: Binary Convolutional Block in a State-of-the-Art Hardware Accelerator

*1) BISAAC Chip Organization and Methodology:* In this section, we evaluated the benefits of our binary convolutional block in the state-of-the-art RRAM-based accelerator ISAAC [16], in its CE configuration. We chose ISAAC as a baseline architecture since it is a well established reference and its seminal paper provides all the necessary circuit level area and energy results for a fair comparison. However, our proposed convolutional block is generic and could be used in many other binary accelerator architectures. In ISAAC, each chip is composed of several tiles, as shown inf Fig. 11 and each tile consists of 12 *In-situ Multiply Accumulate* (IMAs). Each IMA consists of eight *1 Transistor* 1 *RRAM* 128 × 128 crossbars and several peripheral circuits to work with analog signals: ADCs, DACs, *Shift and Add* (S+A), *Sample and Hold* (S+H). However, as discussed in Section II-B, analog dot product functionality is limited by RRAM variability. We showed in Section IV-A.5 that our proposed binary dot product engine can tolerate a high variability on RRAM devices. We therefore introduce BISAAC as a fully binary version of ISAAC by replacing all the analog computations based on RRAM by our proposed engine. Our architecture can bring great robustness against variability as well as energy benefits as it will be demonstrated in Section IV-C.2. For a fair comparison, we first show how ISAAC could support BNNs. Instead of being a multi-level cell, each RRAM cell can represent a single bit and each binary synapse can be represented




| $R_t$ | $R_t$ | Value |
|---|---|---|
| HRS | LRS | -1 |
| LRS | HRS | 1 |

Fig. 12. ISAAC synapse organization for BNN support.

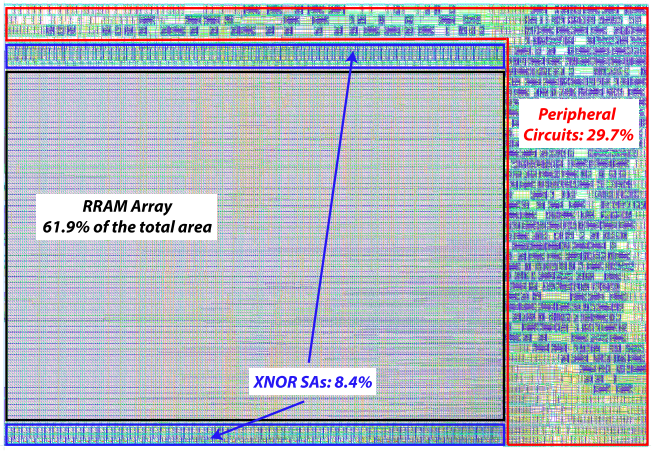



Fig. 13. Layout of a RRAM-based XNOR $128 \times 128$ array and its peripheral circuits with the area repartition of the different blocks.

by two 1-bit cells which will be located in the same column, but in two different rows, as illustrated in Fig. 12. The synapse values are $-1$ or $+1$ depending on the RRAM resistance states $R_t$ and $R_b$. To execute a XNOR operation, the voltages on the columns are set to $V_b = -V_t = V$ and the output is given by the current $I = (R_b - R_t) * V$. As such, a $128 \times 128$ array is still capable of producing 128 binary dot products in a single cycle. In BISAAC, the global tile organization of ISAAC is maintained. The only difference is the IMA organization since they are now using our proposed RRAM-based XNOR array. Consequently, in BISAAC, the ADCs, DACs, S+H and S+A are not required anymore and are replaced by the combinational bitcount circuits and the proposed XNOR SAs to perform binary operations. The power values of the whole RRAM array (and its XNOR SAs) are obtained from HSPICE simulations [42]. Since the array is large, we also considered the presence of output buffers for each column in order to restore the *out* and $\overline{out}$ signals. Compared to ISAAC, we scaled our design from 28nm to 32nm for an apples to apples comparison. For the area estimation, we first draw a full-custom layout of the RRAM array and its associated XNOR SAs on Cadence Virtuoso. The bitcount circuit has been generated with Synopsys Design from Verilog netlists, as explained in Section IV-A.3 and integrated into a Place & Route flow with Cadence Innovus in order to obtain the layout of a whole $128 \times 128$ RRAM-based XNOR array, as presented in the next subsection.

*2) BISAAC Energy and Area Breakdown:* Fig. 13 shows the area repartition of the different blocks of the array after the Place & Route step. Even if each column feed a XNOR sense amplifier, the total XNOR area is only 8.4% of the total area since those cells are relatively small (10 transistors per XNOR), when compared to conventional ADCs used in ISAAC (where they were sharing ADCs among IMAs to save power and area). In addition, we placed the XNOR SAs at the top and the bottom of the array, to ensure a small array area while respecting the pitch matching between the columns and the XNOR SAs. In total, the periphery (bitcount circuit, output buffers and XNOR SAs) occupies 38.1% of the total area, which is relatively low since we used single sense ending. Table II shows the area and power value for ISAAC and BISAAC. In our proposed architecture, since each column requires two RRAM cells and two access transistors (instead of one RRAM cell and one access transistor in the case of ISAAC), we consider that the RRAM array parts of the whole architecture consumes twice the area when compared to ISAAC. However, this not does lead to an area overhead (BISAAC is still 0.7% smaller than ISAAC) since our architecture does not require large ADCs and DACs. In addition, the total power consumption of ISAAC is 65.8W while BISAAC only consumes 24.4W, leading to a power reduction of 2.7×. Again, this is mainly due to the fact that the power hungry ADCs and DACs are not required in our design.

*3) BISAAC Performances and Parallelism Opportunities:* While our proposed convolutional block is sequential, it still achieves the same level of throughput than ISAAC. In ISAAC, 128 dot products can be performed in a single cycle (100 *ns*) due to the parallelism opportunities brought by analog RRAM arrays. However, the overall speed is limited by the ADC sampling rate (1.28 GHz) since each dot product has to be processed through the same ADC sequentially. In BISAAC, the speed is limited by the crossbar array reading time, which is about 771 *ps* for a $128 \times 128$ array from SPICE simulation. As a result, in 100 ns, BISAAC is able to produce 129 dot products which is roughly in the same order than ISAAC. In addition, by a smart mapping of the kernels between different crossbars and IMAs, a level of parallelism can be achieved. A way to reduce the delay of a convolution and increase the throughput can be to replicate the weight in another IMA, as proposed in ISAAC. In that way, two IMAs can proceed different rows from the same image input at the same time, leading to a two time faster convolution. If half of the IMAs of the chip are not used, we can extend this and replicate all the weights, roughly increasing the overall throughput by a factor of two. Finally, the parallelism can be further increased by splitting the RRAM array into several sub-arrays where each sub-array has its own bitcount circuit, as depicted in Fig. 14. For instance, if we consider an array having 128 columns, 14 kernels of size $3 \times 3$ can be unfolded in a single column, meaning that 14 dot products can be processed during the same cycle.

TABLE II
ISAAC-CE [16] AND BISAAC AREA AND POWER COMPARISON

| **ISAAC-CE [16]** | | | | **BISAAC** | | | |
|---|---|---|---|---|---|---|---|
| **Component** | **Number** | **Power** | **Area $mm^2$** | **Component** | **Number** | **Power** | **Area $mm^2$** |
| Tile periphery(1) | | 40.9 mW | 0.215 | Tile periphery(1) | | 40.9 mW | 0.215 |
| Hyper Tr(2) | | 10.4 W | 22.88 | Hyper Tr(2) | | 10.4 W | 22.88 |
| **IMA properties (12 IMAs per tile) 1 IMA = ADC + DAC + S+H + S+A + IR + OR + RRAM array** | | | | **IMA properties (12 IMAs per tile) 1 IMA = BC + IR + OR + RRAM array + XNOR SA + output buffers** | | | |
| ADC | 8 | 16 mW | 0.0096 | Bitcount circuit (BC) | 8 | 3.96 $\mu$W | 0.0061 |
| DAC | 8×128 | 4 mW | 0.00017 | | | | |
| S+H | 8×128 | 10 $\mu$W | 0.00004 | | | | |
| S+A | 4 | 0.2 mW | 0.00024 | | | | |
| IR | 1 | 1.24 mW | 0.0021 | IR | 1 | 1.24 mW | 0.0021 |
| OR | 1 | 0.23 mW | 0.00077 | OR | 1 | 0.23 mW | 0.00077 |
| RRAM array | 8 | 2.4 mW | 0.0002 | RRAM array + XNOR SAs + output buffers | 8 | 2.0 mW(3) | 0.00387(3) |
| **IMA total** | 12 | **289 mW** | **0.157** | **IMA total** | 12 | **42.5 mW** | **0.154** |
| **1 Tile total (12 IMAs + periph.)** | | **330 mW** | **0.372** | **1 Tile total (12 IMAs + periph.)** | | **83.4 mW** | **0.369** |
| **168 Tile total** | | **55.4 W** | **62.5** | **168 Tile total** | | **14 W** | **61.9** |
| **Chip total (tiles + hyper tr)** | | **65.8 W** | **85.4** | **Chip total (tiles + hyper tr)** | | **24.4 W** | **84.8** |

(1) Periphery: eDRAM buffers, EDRAM-to-IMA buses, routers, sigmoids, S+A, MaxPool and OR [16].
(2) Hyper Transport serial link. More details about this component can be found in [16].
(3) Considering the RRAM arrays as well as the XNOR SAs and the output buffers. For the RRAM array, each column requires twice the number of RRAM cells and access transistors when compared to ISAAC so the area is doubled.

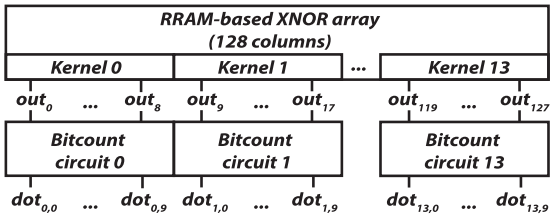



Fig. 14. Parallelism possibilities by mapping 14 kernels of size 3×3 into a single row having 128 columns.

## V. CONCLUSION

In this paper, we proposed a fully binary RRAM-based convolutional engine that performs digital dot product operations, without relying on an analog sum of currents. Electrical simulations at the 28nm technology node showed that our design can tolerate a high RRAM variability ($\sigma/\mu$ =25% for a HRS/LRS ratio around 50) without any failure, showing the robustness of our architecture against RRAM and CMOS process variations. The low tolerable HRS/LRS ratio (1.09) opens the opportunity for using other memory technologies such as Spin-Transfer Torque RAM. Architectural evaluations showed that the energy can be reduced by 2.2× compared to conventional MAC operators for a binary image dilation. In addition, our convolutional block can be used in BNNs, which can ensure a high accuracy (within 10% reduction compared to the full precision networks) by being integrated into a state-of-the-art RRAM-based hardware accelerator such as ISAAC. In that case, we showed that the overall power consumption can be reduced by 2.7× compared to its conventional analog structure.

## ACKNOWLEDGMENTS

The authors would like to thank professor Rajeev Balasubramonian from the University of Utah for the fruitful discussion.

## REFERENCES

[1] S. H. Chen, "Comparison of delta-type discrete singular convolution kernels for anti-noise edge detection," in *Proc. Int. Symp. Comput., Consum. Control*, Jun. 2014, pp. 1229–1232.
[2] C. J. Pye and J. A. Bangham, "A fast algorithm for morphological erosion and dilation," in *Proc. 8th Eur. Signal Process. Conf. (EUSIPCO)*, Sep. 1996, pp. 1–4.
[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 25, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Red Hook, NY, USA: Curran Associates, 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
[4] T. Yoshioka, S. Karita, and T. Nakatani, "Far-field speech recognition using CNN-DNN-HMM with convolution in time," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Apr. 2015, pp. 4360–4364.
[5] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *IEEE ISSCC Dig. Tech. Papers.*, Feb. 2014, pp. 10–14.
[6] L. Du *et al.*, "A reconfigurable streaming deep convolutional neural network accelerator for Internet of Things," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 1, pp. 198–208, Jan. 2018.
[7] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.
[8] J. Jo, S. Kim, and I.-C. Park, "Energy-efficient convolution architecture based on rescheduled dataflow," *IEEE Trans. Circuits Syst. I, Reg. Papers*, to be published.
[9] J. Sim, J.-S. Park, M. Kim, D. Bae, Y. Choi, and L.-S. Kim, "A 1.42TOPS/W deep convolutional neural network recognition processor for intelligent IoE systems," in *IEEE ISSCC Dig. Tech. Papers.*, Jan./Feb. 2016, pp. 264–265.

[10] S. Kim, P. Howe, T. Moreau, A. Alaghi, L. Ceze, and V. S. Sathe, "Energy-efficient neural network acceleration in the presence of bit-level memory errors," *IEEE Trans. Circuits Syst. I, Reg. Papers*, to be published.

[11] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," *CoRR*, vol. abs/1603.05279, Aug. 2016. [Online]. Available: http://arxiv.org/abs/1603.05279

[12] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to $+1$ or $-1$," *CoRR*, vol. abs/1602.02830, Mar. 2016. [Online]. Available: http://arxiv.org/abs/1602.02830

[13] L. Ni, Z. Liu, H. Yu, and R. V. Joshi, "An energy-efficient digital ReRAM-crossbar-based CNN with bitwise parallelism," *IEEE J. Explor. Solid-State Computat. Devices Circuits*, vol. 3, pp. 37–46, 2017.

[14] T. Tang, L. Xia, B. Li, Y. Wang, and H. Yang, "Binary convolutional neural network on RRAM," in *Proc. 22nd Asia South Pacific Design Automat. Conf. (ASP-DAC)*, Jan. 2017, pp. 782–787.

[15] X. Sun, S. Yin, X. Peng, R. Liu, J.-S. Seo, and S. Yu, "XNOR-RRAM: A scalable and parallel resistive synaptic architecture for binary neural networks," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2018, pp. 1423–1428.

[16] A. Shafiee *et al.*, "ISAAC: A convolutional neural network accelerator with *in-situ* analog arithmetic in crossbars," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 14–26.

[17] M. Hu *et al.*, "Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication," in *Proc. Design Autom. Conf. (DAC)*, Jun. 2016, pp. 1–6.

[18] D. Soudry, D. Di Castro, A. Gal, A. Kolodny, and S. Kvatinsky, "Memristor-based multilayer neural networks with Online gradient descent training," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 10, pp. 2408–2421, Oct. 2015.

[19] X. Liu *et al.*, "RENO: A high-efficient reconfigurable neuromorphic computing accelerator design," in *Proc. 52nd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2015, pp. 1–6.

[20] L. Xia *et al.*, "Switched by input: Power efficient structure for RRAM-based convolutional neural network," in *Proc. 53rd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2016, pp. 1–6.

[21] X. Chen, J. Jiang, J. Zhu, and C.-Y. Tsui, "A high-throughput and energy-efficient RRAM-based convolutional neural network using data encoding and dynamic quantization," in *Proc. 23rd Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2018, pp. 123–128.

[22] P. Chi *et al.*, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 27–39.

[23] S. Yu, X. Guan, and H.-S. P. Wong, "On the switching parameter variation of metal oxide RRAM—Part II: Model corroboration and device design strategy," *IEEE Trans. Electron Devices*, vol. 59, no. 4, pp. 1183–1188, Apr. 2012.

[24] A. Levisse, B. Giraud, J. P. Noël, M. Moreau, and J. M. Portal, "Sneakpath compensation circuit for programming and read operations in RRAM-based crosspoint architectures," in *Proc. 15th Non-Volatile Memory Technol. Symp. (NVMTS)*, Oct. 2015, pp. 1–4.

[25] L. Xia *et al.*, "Technological exploration of RRAM crossbar array for matrix-vector multiplication," *J. Comput. Sci. Technol.*, vol. 31, no. 1, pp. 3–19, Jan. 2016, doi: 10.1007/s11390-016-1608-8.

[26] S. Ambrogio, S. Balatti, V. McCaffrey, D. Wang, and D. Ielmini, "Impact of low-frequency noise on read distributions of resistive switching memory (RRAM)," in *Proc. IEEE Int. Electron Devices Meeting*, Dec. 2014, pp. 14.4.1–14.4.4.

[27] H.-S. P. Wong *et al.*, "Metal-oxide RRAM," *Proc. IEEE*, vol. 100, no. 6, pp. 1951–1970, Jun. 2012.

[28] X. Tang, E. Giacomin, G. D. Micheli, and P.-E. Gaillardon, "Circuit designs of high-performance and low-power RRAM-based multiplexers based on 4T(ransistor)1R(RAM) programming structure," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 64, no. 5, pp. 1173–1186, May 2017.

[29] X. Tang, G. Kim, P.-E. Gaillardon, and G. De Micheli, "A study on the programming structures for RRAM-based FPGA architectures," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 63, no. 4, pp. 503–516, Apr. 2016.

[30] J. Liang and H.-S. P. Wong, "Cross-point memory array without cell selectors-device characteristics and data storage pattern dependencies," *IEEE Trans. Electron Devices*, vol. 57, no. 10, pp. 2531–2538, Oct. 2010.

[31] E. Rosenthal, S. Greshnikov, D. Soudry, and S. Kvatinsky, "A fully analog memristor-based neural network with online gradient training," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2016, pp. 1394–1397.

[32] A. Chen and M.-R. Lin, "Variability of resistive switching memories and its impact on crossbar array performance," in *Proc. Int. Rel. Phys. Symp.*, Apr. 2011, pp. MY.7.1–MY.7.4.

[33] B. Feinberg, S. Wang, and E. Ipek, "Making memristive neural network accelerators reliable," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2018, pp. 52–65.

[34] I. Agbo *et al.*, "Quantification of sense amplifier offset voltage degradation due to zero-and run-time variability," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2016, pp. 725–730.

[35] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," in *Artificial Neural Networks and Machine Learning*, S. Wermter *et al.*, Eds. Cham, Switzerland: Springer, 2014, pp. 281–290.

[36] D. Soudry, I. Hubara, and R. Meir, "Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights," in *Proc. 27th Int. Conf. Neural Inf. Process. Syst. (NIPS)*, vol. 1. Cambridge, MA, USA: MIT Press, 2014, pp. 963–971. [Online]. Available: http://dl.acm.org/citation.cfm?id=2968826.2968934

[37] W. Zhao *et al.*, "Synchronous non-volatile logic gate design based on resistive switching memories," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 61, no. 2, pp. 443–454, Feb. 2014.

[38] S. Matsunaga *et al.*, "MTJ-based nonvolatile logic-in-memory circuit, future prospects and issues," in *Proc. Design, Autom. Test Eur. Conf. Exhib.*, Apr. 2009, pp. 433–435.

[39] Z. Jiang, S. Yu, Y. Wu, J. H. Engel, X. Guan, and H. S. P. Wong, "Verilog-a compact model for oxide-based resistive random access memory (RRAM)," in *Proc. Int. Conf. Simulation Semiconductor Process. Devices (SISPAD)*, Sep. 2014, pp. 41–44.

[40] K. T. Malladi, F. A. Nothaft, K. Periyathambi, B. C. Lee, C. Kozyrakis, and M. Horowitz, "Towards energy-proportional datacenter memory with mobile DRAM," in *Proc. 39th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2012, pp. 37–48.

[41] N. Binkert *et al.*, "The gem5 simulator," *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, 2011.

[42] *HSPICE user guide: Basic simulation and analysis*, Synopsys, Mountain View, CA, USA, 2018.

[43] H. Reyserhove, N. Reynders, and W. Dehaene, "Ultra-low voltage datapath blocks in 28nm UTBB FD-SOI," in *Proc. IEEE Asian Solid-State Circuits Conf. (A-SSCC)*, Nov. 2014, pp. 49–52.

[44] C. Torres-Huitzil, "Fast hardware architecture for grey-level image morphology with flat structuring elements," *IET Image Process.*, vol. 8, no. 2, pp. 112–121, Feb. 2014.


**Edouard Giacomin** received the M.Sc. degree in electrical and computer engineering from the École Supérieure de Chimie Physique Électronique de Lyon, Lyon, France, in 2016. He is currently pursuing the Ph.D. degree in electrical engineering with The University of Utah, UT, USA. In 2018, he was a Visiting Research Intern with imec, Leuven, Belgium. His research interests include emerging technologies such as resistive random-access memory, 3-D nanofabric architectures, and very large-scale integration design.

**Tzofnat Greenberg-Toledo** received the B.Sc. degree in electrical engineering from Technion in 2015. She is currently pursuing a master's degree with the Andrew and Erna Viterbi Faculty of Electrical Engineering, Technion—Israel Institute of Technology. From 2014 to 2016, she was a Logic Design Engineer with Intel Corp. Her research interests are computer architecture and accelerators for deep neural networks with the use of memristors.

**Shahar Kvatinsky** received the B.Sc. degree in computer engineering and applied physics and the M.B.A. degree from The Hebrew University of Jerusalem in 2009 and 2010, respectively, and the Ph.D. degree in electrical engineering from the Technion—Israel Institute of Technology in 2014. From 2006 to 2009, he was a Circuit Designer with Intel. He was a Post-Doctoral Research Fellow with Stanford University from 2014 to 2015. He is currently an Assistant Professor with the Andrew and Erna Viterbi Faculty of Electrical Engineering, Technion—Israel Institute of Technology. His current research is focused on circuits and architectures with emerging memory technologies and design of energy efficient architectures. He was a recipient of the 2015 IEEE Guillemin-Cauer Best Paper Award, the 2015 Best Paper of Computer Architecture Letters, the Viterbi Fellowship, the Jacobs Fellowship, the ERC Starting Grant, the 2017 Pazy Memorial Award, the 2014 and 2017 Hershel Rich Technion Innovation Awards, the 2013 Sanford Kaplan Prize for Creative Management in High Tech, the 2010 Benin Prize, and six Technion excellence teaching awards. He is an Editor of the *Microelectronics Journal*.

**Pierre-Emmanuel Gaillardon** (S'10–M'11–SM'16) received the Degree in electrical engineering from CPE-Lyon, France, in 2008, the M.Sc. degree in electrical engineering from INSA, Lyon, France, in 2008, and the Ph.D. degree in electrical engineering from CEA-LETI, Grenoble, France, and the University of Lyon, France, in 2011. He was a Research Associate with the Laboratory of Integrated Systems (Prof. De Micheli), Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland, and a Visiting Research Associate with Stanford University, Palo Alto, CA, USA. He was a Research Assistant with CEA-LETI. He is currently an Assistant Professor with the Electrical and Computer Engineering Department, The University of Utah, Salt Lake City, UT, USA. He is also an Adjunct Assistant Professor with the School of Computing, The University of Utah, where he leads the Laboratory for NanoIntegrated Systems. His research activities and interests are currently focused on the development of novel computing systems exploiting emerging device technologies and novel EDA techniques. He was a recipient of the C-Innov 2011 Best Thesis Award, the Nanoarch 2012 Best Paper Award, the BSF 2017 Prof. Pazy Memorial Research Award, the 2017 NSF CAREER Award, and the 2018 IEEE CEDA Pederson Award. He is an Associate Editor of the IEEE TRANSACTIONS ON NANOTECHNOLOGY. He has been serving as a TPC member for many conferences, including the DATE 2015–2019, the DAC 2016–2018, and the Nanoarch 2012–2017, and is a reviewer for several journals and funding agencies. He served as a Topic Co-Chair "Emerging Technologies for Future Memories" for the DATE 2017–2019.