BCIM: Efficient Implementation of Binary Neural Network Based on Computation in Memory

Mahdi Zahedi, Taha Shahroodi, Carlos Escuin, Georgi Gaydadjiev, *Senior Member, IEEE*, Stephan Wong, *Senior Member, IEEE*, Said Hamdioui, *Senior Member, IEEE*

***Abstract*—Applications of Binary Neural Networks (BNNs) are promising for embedded systems with hard constraints on energy and computing power. Contrary to conventional neural networks using floating-point datatypes, BNNs use binarized weights and activations to reduce memory and computation requirements. Memristors, emerging non-volatile memory devices, show great potential as a target implementation platform for BNNs by integrating storage and compute units. However, the efficiency of this hardware highly depends on how the network is mapped and executed on these devices. In this paper, we propose an efficient implementation of XNOR-based BNN to maximize parallelization. In this implementation, costly analog-to-digital converters are replaced with sense amplifiers with custom reference(s) to generate activation values. Besides, a novel mapping is introduced to minimize the overhead of data communication between convolution layers mapped to different memristor crossbars. This comes with extensive analytical and simulation-based analysis to evaluate the implication of different design choices considering the accuracy of the network. The results show that our approach achieves up to $5\times$ energy-saving and $100\times$ improvement in latency compared to baselines.**

***Index Terms*—Computation-in-memory, Memristor, Binary neural network, energy-efficient accelerator**

I. Introduction

NEURAL Networks (NNs) are leveraged in a variety of applications [1]–[3]. With the growth of the network size for advanced applications, the implementation of NNs has become challenging, considering hardware limitations (e.g., BERT has around 110 million parameters [4]). Binary Neural networks (BNNs), where the weights and activation values are binarized (-1,+1), receive more attention from researchers due to their high model compression rate and simplified computations [5]. This is appealing for edge devices where there are hard constraints on memory capacity, computing resources, and energy budget. Although the computations are simplified, further improvement in the efficiency of BNN implementations relies on reducing the cost of data transfer between memory and computing units (memory wall). Computation-in-Memory (CIM) and the unique characteristics of emerging non-volatile memories (memristors) [6]–[8] are promising candidates to deliver the next level of energy-efficiency implementation of BNNs. Hence, there is a need to design energy-efficient accelerators leveraging the notion of CIM to enable large-size networks for edge devices.

Memristor crossbar structures are tailored for vector-matrix multiplication (VMM) operation. As a result, a wide range of applications such as graph processing [9], [10], bioinformatics [11]–[13], image processing [14], [15], and security [16], [17] are promising to be accelerated by CIM-based hardware accelerators [18], [19]. CIM-based accelerators utilizing these memories not only reduce the overhead of data transfer, but also enhance the performance of VMM operation as a key kernel in BNNs. However, using memristors to operate on signed numbers (-1,+1) in BNN is challenging. From this perspective, existing works can be classified into hardware or algorithmic solutions. As a hardware solution, positive and negative values can be mapped to different memristors [20]–[22]. Other approaches are considering one- [23] or two-column reference memristors [24] while converting the weights and activations to unsigned representation. In general, these approaches require more devices, increase design complexity, and reduce the energy/performance efficiency of the system. As an algorithmic solution, a signed VMM can be converted to XNOR operations [25] where the operands are unsigned (0,1). In this category, an accelerator is designed in which memristors are also used as an activation function [26]. This induces endurance, energy, and performance issues due to excessive memristor programming. To ensure the accuracy of XNOR operations against device variation, a new memristor crossbar structure based on differential sensing is proposed [27]. However, XNOR operations are forced to be performed sequentially due to the sensing mechanism. All these overheads drive researchers to explore new mappings and implementations of BNNs to enhance their efficiency further for edge devices.

This work advances the state-of-the-art by proposing an efficient implementation of BNNs. The proposed mapping of operands for XNOR operations to the crossbar allows simultaneous crossbar row activation. This maximizes resource utilization on the crossbar and enhances performance. Moreover, we mimic the functionality of an Analog-to-Digital Converter (ADC) and the following digital processing, initially needed for this mapping, by only a Sense Amplifier (SA) with an adjusted reference. Furthermore, we minimize data communication between layers by proposing a novel mapping of the weights and activation values into the crossbar and its

M.Zahedi, T.Shahroodi, S.Wong, and S.Hamdioui are with the Department of Quantum and Computer Engineering, Delft University of Technology, Delft, The Netherlands. Email: {M.Z.Zahedi, T.Shahroodi, J.S.S.M.Wong, S.Hamdioui}@tudelft.nl

Georgi Gaydadjiev is with Delft University of Technology and Imperial College London. Email: g.n.gaydadjiev@tudelft.nl

Carlos Escuin is with the Universidad de Zaragoza, Spain. Email: escuin@unizar.es

Color versions of one or more figures in this article are available at https://doi.org/10.xxxx/TETC.2023.xxxx. Digital Object Identifier 10.xxxx/TETC.2023.xxxx

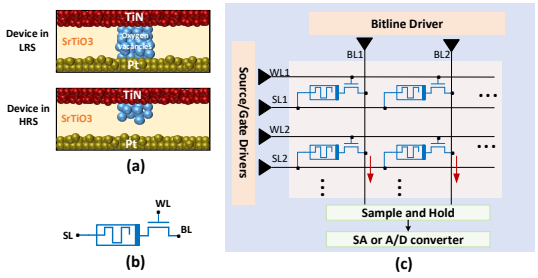

Fig. 1: (a) ReRAM memristor device behavior (b) 1T1R memristor cell (c) CIM tile encompassing crossbar and peripheries.

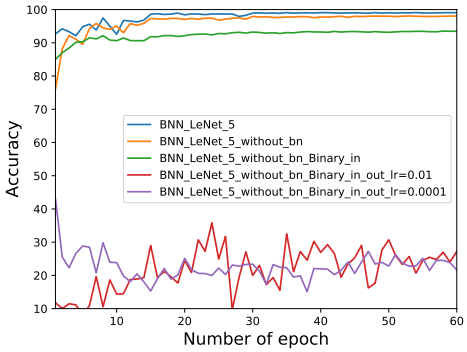

Fig. 2: Accuracy of binarized LeNet5 network and the impact of input/output layer binarization as well batch normalization (bn) on accuracy loss using different learning rates (lr).

input buffer. We investigate the efficiency of our approach on different network structures in terms of accuracy, energy, and performance by developing our PyTorch-based simulation platform [28]. The platform can mimic the behavior of the crossbar and allows for more characteristics and non-idealities to be integrated and explored for different networks. Our approach achieves close to $5\times$ energy-saving and $100\times$ improvement in latency compared to the state-of-the-art computation-in-memory designs at the cost of up to 4% accuracy loss. This paper presents the following main contributions:

- An energy-efficient and highly parallel implementation of XNOR-based BNNs where the functionality of ADC and the required digital processing after that are modeled by a SA with an adjusted reference;
- An efficient mapping of the weights and activation values to improve data utilization and minimize the communication between network layers;
- An extensive analytical and simulation-based analysis where the proposed implementation behaves as an approximation to comprehend the implication of SA reference values on the accuracy of the design.

The paper is organized as follows. Section II provides background on memristor devices and binary neural networks. Existing accelerators for BNN are explained in Section III. We discuss our proposal design in Section IV. In Section V, we perform analytical analyses to elaborate more on the implications of the sensing scheme on accuracy. Section VI evaluates the design, while Section VII concludes the paper.

II. Preliminary

In this section, we first provide background on memristor devices and the operations supported in a crossbar array, and then, we briefly explain the basics of binary neural networks.

A. Memristor devices

Despite charge-based memories, memristor devices hold data as resistance levels. The data can be presented as a binary value utilizing a low resistive state (LRS) and a high resistive state (HRS). Among different memristor technologies, Figure 1(a) illustrates Resistive Random-Access Memory (ReRAM) devices [29] consisting of a metal-insulator-metal stack; the bipolar device is set and reset by changing the polarity of the programming voltage (e.g., 2V) to form or dissolve the conducting filament. To read the device without disturbance, a small voltage (e.g., 0.2V) is applied, and the current (voltage) through (across) the device should be sensed while programming the device requires higher voltage/current and longer latency. Figure 1(b) shows a schematic representation of the 1T1R memristor-based structure. This is a fundamental block for constructing a CIM tile encompassing memristors in crossbar structure and peripheries, as shown in Figure 1(c), where drivers are employed to drive Select-line (SL), Word-line (WL), and Bit-line(BL). The analog output of the crossbar is captured and converted to the digital domain using a sense amplifier (SA) or A/D converter (ADC). The main computational operations that can be performed on the crossbar include addition, logical operations, and Matrix-Matrix Multiplication (MMM). Besides the capabilities of co-locating computation and storage together, huge parallelism can be achieved within a single memory array (crossbar and its periphery) as well as at the inter-array level. These are the main drivers that attract researchers to exploit this concept for different state-of-the-art applications [30].

B. Fundamentals of Binary Neural Networks

Designing larger networks and the ability to train them with advanced algorithms were the main drivers to enable neural networks for complex applications. However, implementing these networks in embedded platforms with limited storage and computation units is challenging, specifically in consideration of strict energy/performance constraints. Despite conventional neural networks with high precision datatypes, in BNNs, weights and activations are binarized to make the network extremely compact. Equation 1 shows a simple binarization rule that can be applied to both activations (input tensors) and weights where B_ω and B_I are the binarized weights and input tensors, respectively. The binarization not only saves on storage usage, but also reduces the expensive multiply-accumulate operations to simple additions.

$$B_\omega = \begin{cases} +1 & if\ \omega \geq 0 \\ -1 & if\ \omega < 0 \end{cases} B_I = \begin{cases} +1 & if\ I \geq 0 \\ -1 & if\ I < 0 \end{cases} \quad (1)$$

Although binarization enhances the system's efficiency in terms of memory usage, energy, and performance, it usually

comes at the cost of accuracy loss compared to its high-precision counterpart. Therefore, using proper methods and algorithms to preserve the accuracy of the network as high as possible is essential. Each iteration of training a network can be divided into three steps: forward pass, backward propagation, and parameter update. The weights during the forward pass and backward propagation are binarized. However, we need to use high-precision weights during parameter updates. Since parameter changes obtained by gradient descent are tiny, binarization ignores these changes and the network cannot be trained [25], [31]. In addition, binarizing the input and output layers usually results in a huge accuracy loss. Figure 2 depicts the accuracy of the binarized LeNet5 network trained for the MNIST dataset. This clearly shows the impact of binarizing the input and output layers as well as batch normalization (bn) on the accuracy of the network. Batch normalization normalizes the contribution of layers' input. This helps to stabilize the learning processes during network training.

III. Memristor-based accelerators for BNN

To implement BNNs, besides using traditional systems (CPU, GPU, and FPGA) [32]–[34], computation-in-memory (CIM) accelerators based on emerging non-volatile memories (memristors) draw the attention of researchers. Memristor crossbar arrays are tailored to perform analog VMM with higher energy efficiency compared to their digital counterpart (CPU/GPU) [35]. Memristor devices usually can alternate between a few resistance levels (e.g., two levels) while more levels lead to reliability, stability, and accuracy degradation. Hence, BNN-based applications where the main kernel (VMM) is binarized are the promising targets to be implemented using memristor devices. A small-scale demonstration of a BNN on memristor devices is presented in [36] with focusing mainly on device variation and its implication on accuracy. A new methodology is proposed in [37] to make the design more tolerant against device variation to be able to activate more word-lines and perform more computation at the same time. Based on Equation 1, BNNs require signed representation, but negative numbers cannot be directly stored in memristors. Accordingly, existing BNN accelerators can fit in two categories based on how they address the problem.

- Hardware solutions: To deal with signed numbers, both weights and activation values can be represented as two vectors only holding absolute numbers; one holds positive and one holds negative values [20]. The two vectors created for both the weights and activation values are programmed to the corresponding memristors and sent to the input ports of a crossbar (select-line), respectively. Subsequently, the four possible partial results are computed and summed up in an analog manner. This requires a high number of memristor devices which translates to low area and energy efficiency. In addition, since the proposed approach requires input current in both directions, the complexity of input drivers is increased. A similar approach is mapping positive and negative weights into different crossbars [21], [22]. In these works, ADC is exploited to compute the partial result when a BNN layer size is larger than the crossbar size. Then, the partial result from different crossbars is accumulated and given to an activation function. However, using ADCs imposes significant energy and area overhead on the system. Another solution is using one- [23] or two-column reference memristors [24] while the weights and activations are presented as $\{0,1\}$. In this design, the current flowing through the reference column(s) has to be mirrored equal to the number of columns in the crossbar. This increases the design complexity and energy consumption of the system. In addition, when a layer size cannot fit into a crossbar, it gets critical to have a flexible referencing scheme to avoid accuracy loss. We discuss this more in Section V.

- Algorithmic solutions: Binary multiply and accumulate operation can be replaced by the following sequence of operations: **XNOR, popcount,** and **post processing** [25]. As a result, the weights and activations for BNN can be presented as unsigned $\{0,1\}$ values. This makes the implementation of BNNs on memristor crossbars simpler. Memristor-based content-addressable memory (CAM) structure can be used to implement binary XNOR operation and, in turn, BNNs [26]. In this design, the activation function is implemented by a memristor where its state determines the input value for the next layer. However, this suffers from an extremely high number of device programming, which causes challenges in terms of reliability, performance, and energy. An XNOR-based robust design to device imperfections is proposed using a differential sensing mechanism [27]. Due to the structure of the crossbar and the mapping of the weights, this design cannot exploit maximum parallelism in producing output values for each layer of BNN. This work is closest to our design and is considered a baseline. We elaborate more in the following section.

In conclusion and considering the limitations and challenges of existing works, a novel highly-parallel and energy-efficient BNN design is needed.

IV. Methodology

In this section, we first explain the principles behind XNOR-based BNNs. This is used for both the implementation of BCIM and the baseline. Second, we discuss data mapping and execution of BNNs in the baseline [27]. Third, we present the new mapping and execution of BNNs in BCIM and compare it with the baseline. Fourth, we explain how the crossbar's input buffer, holding activation values, is managed to minimize data transfer between crossbars implementing the BNN layers.

A. Multiply-accumulate based on XNOR operation

The multiply-accumulate operation between two signed binarized vectors can be replaced by the sequence of 1) XNOR, 2) popcount, and 3) post-processing (Shift and Subtract) operations [25]. To achieve that, first, both vectors are converted from signed to unsigned, where '-1' is replaced by '0'. This is helpful considering memristor devices since it simplifies

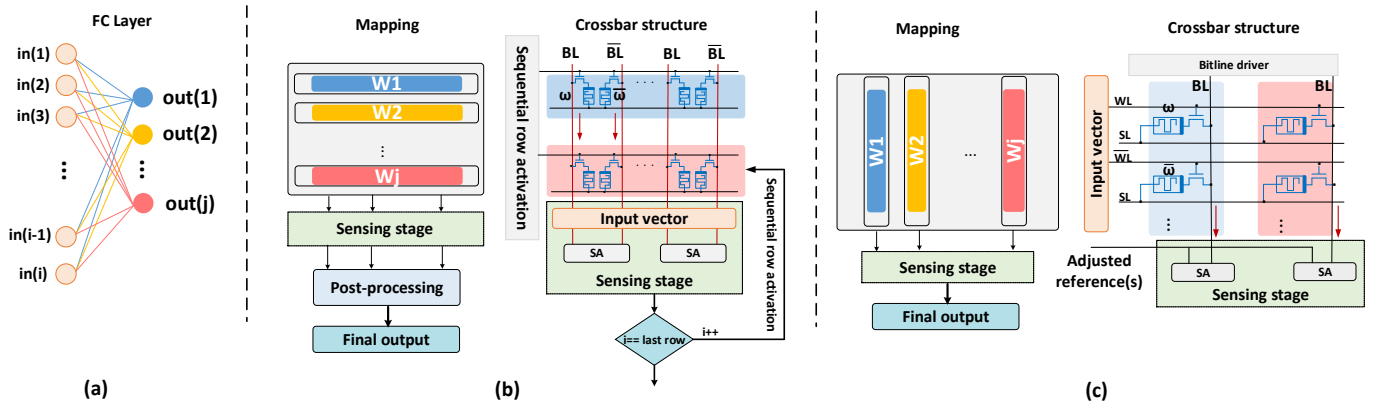

Fig. 3: (a) An illustration of a fully connected layer (b) BNN implementation using differential sensing and sequential XNOR operation [27] (c) proposed design where massive XNOR operations are performed in parallel.

the mapping of weights to the crossbar without concern for negative values. Second, by applying Equation 2, the final value is obtained where *A'* is the unsigned representation of vector *A*. Popcount() returns the number of ones in a bitstream, and *'vector size'* is the length of the two vectors.

$$A * B = 2 * Popcount(A^{'} \odot B^{'}) - vector\ size \quad (2)$$

In the following, an example is provided to have better clarification. The result of the multiply-accumulate operation between vectors A and B from the traditional approach is:

$$A = [1, -1, -1, 1]\ B = [-1, 1, 1, 1] \Rightarrow A * B = -2$$

In the new approach, vectors $A^{'}$ and $B^{'}$ are created by converting A and B from signed to unsigned representation. First, we perform the XNOR operation between $A^{'}$ and $B^{'}$. Second, the result is given to the *Popcount* function. Third, the final processing, including Shift and Subtraction, is performed on the output of the Popcount function.

$$A^{'} = [1, 0, 0, 1]\ B^{'} = [0, 1, 1, 1] \Rightarrow A^{'} \odot B^{'} = [0, 0, 0, 1]$$
$$A*B = 2*Popcount(A^{'} \odot B^{'}) - vector\ size = 2*1 - 4 = -2$$

By applying the above method for BNNs, one vector can be considered as an activation vector ($A^{'}$) while another vector ($B^{'}$) holds the weights. The result is an activation value for the next layer. The process of generating the activation value for the next layer can be expressed as:

$$out_m = Sign(2 * \sum_{k=1}^{I} (in_k \odot \omega_{k,m}) - vector\ size) \quad (3)$$

where in_k represents the k^{th} activation value for the current layer; $\omega_{k,m}$ is the weight connecting the k^{th} activation value to the m^{th} output; and I is equal to the number of activation values of the current layer. The operator Σ performs as the Popcount function. Using this algorithmic solution to avoid representing negative data can reduce the number of memristor devices $2\times$ compared to a hardware solution [20]

In the following, we provide an example of a fully connected layer to explain how the baseline as well as BCIM map and execute BNNs based on the above methodology.

B. State-of-the-art XNOR-based BNN

Figure 3(b) illustrates how a fully connected layer is mapped to a crossbar based on the approach proposed in [27]. The binary weights (ω) and their complements ($\overline{\omega}$), associated with each output channel (indicated by different colors), are programmed into one row of the crossbar. In order to compute the result for one output channel, first, its corresponding row is read. Second, the XNOR operation is performed with the (analog) value on the bit-lines of the crossbar and the input activation vector (orange box in Figure 3(b)). This operation is done within the sensing stage by modifying the circuit of Sense Amplifiers (SAs). In this design, the complementary value for both weights and input vector is required to be able to perform the XNOR operation in the sensing stage. Finally, the output is given to the digital periphery to perform Popcount, Shift, and Subtract operations.

C. Proposed XNOR-based BNN implementation

Figure 3(c) depicts the mapping of the weights and the crossbar structure in BCIM. All the weights (ω) and their complementary values ($\overline{\omega}$), corresponding to each output channel, are programmed in one column of the crossbar. We provide the input activation vector as well their complements to the word-lines ($WL, \overline{WL}$) of the crossbar. Therefore, two memristors are allocated for each weight (ω and $\overline{\omega}$) and two world-lines for each activation value (WL and $\overline{WL}$). Hence, XNOR operation between one element of the activation vector and the weight vector is computed as $in_k \odot \omega_{k,m} = in_k.\omega_{k,m} + \overline{in_k}.\overline{\omega_{k,m}}$ where in_k and $\overline{in_k}$ are the activation values provided to the WL and $\overline{WL}$, respectively. It should be noted that the summation between the two terms of the above formula is implemented with analog addition on the bit-line.

The vector resulting from XNOR operations on all the pair elements of activation and weights vectors should be passed through the Popcount function. This is indicated in Equation 3 by the Σ operator. Since each element contributes to the current flowing to the same bit-line, the analog sum of contributions represents the output of Popcount in the analog domain. In the naive approach, this analog value can be translated to the digital domain by using an Analog-to-Digital Converter (ADC). Then, we perform other operations (Shift and

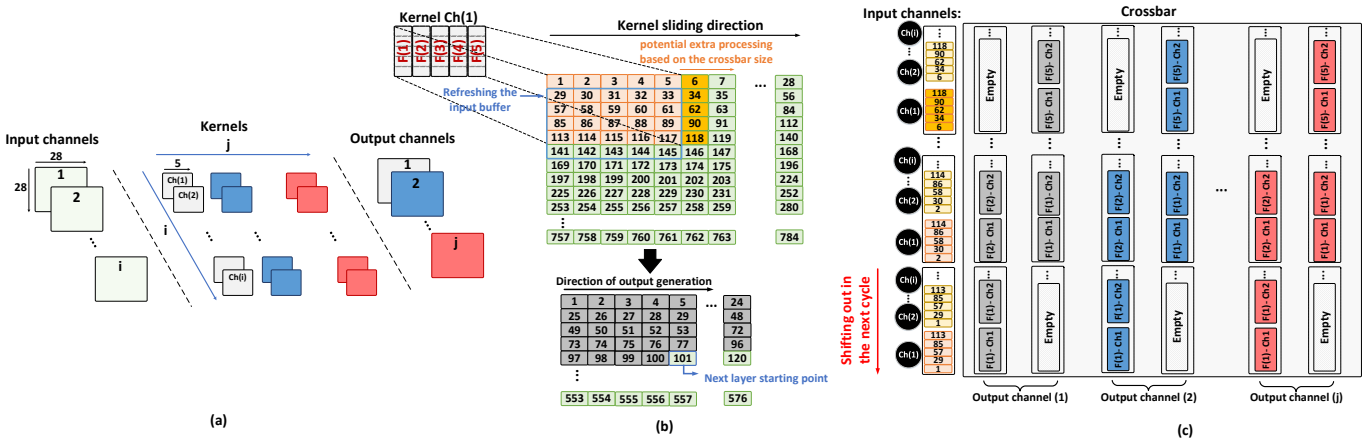

Fig. 4: (a) Example of a CNN layer (b) details of a convolution operation with 5×5 and 28×28 kernel and input size (c) mapping of the activation values to the input buffer and kernels to the crossbar based on the proposed approach to minimize data transmission between layers by only streaming the newly computed activation values into the input buffer.

Subtraction) in the periphery of the crossbar. However, ADC is a power and area-hungry component [38]. Consequently, using this component not only reduces the energy efficiency of the design, but also has to be time multiplexed between several bit-lines, which in turn reduces the performance. However, according to Equation 3, the output of Popcount can be directly compared with a reference noted in Equation 4 to obtain the final output. Hence, the bit-line's analog output can be given to a SA with a customized reference to generate the output value. This also eliminates the remaining processing (Shift and Subtraction) in the digital periphery.

$$SA\ reference = vector\ size/2 \tag{4}$$

Based on this approach, we first maximized the number of parallel output activation values that can be computed for a BNN layer. All the bit-lines can be activated in parallel to compute the result for several output channels. Second, to avoid reducing the performance and energy efficiency by utilizing a high-resolution ADC, a simple analog SA with a customized referencing value is deployed. This not only performs the sign operation, but also omits extra digital processing in the periphery, thereby achieving considerable energy and performance improvement. Third, in this method, when vectors cannot fit into one column of the crossbar, they have to be broken and mapped to several columns. This may lead to approximate computing. In Section V, we scrutinize this interesting scenario analytically.

D. Efficient data movement

Data movement between the BNN layers may influence the performance and energy of the system [38], but is often overlooked by the existing works. In this subsection, we focus on how the data should be transferred from one convolutional layer to the next one to minimize the number of transactions and the size of a buffer placed between layers. This approach can be utilized for both, binary and non-binary datatypes.

Figure 4(a) depicts an example of a convolution layer where the kernel matrix is convolved into the *"i"* input channels to generate data for the *"j"* output channels. In this example, the input size for each channel and the kernel size are 28×28 and 5×5, respectively. Figure 4(b) illustrates the details of the convolution operation where each kernel slides on a corresponding input channel to produce the partial result. The kernels are programmed to the crossbar while the data of input channels corresponding to the current operating window (highlighted by light orange) are buffered and sent to the word-lines of the crossbar. When the operating window slides, the data has to be sent and reorganized in the buffer to be matched to the weights of the kernel programmed into the crossbar. However, bringing the whole data again for the following operating window is not an efficient way since most of it already exists in the input buffer of the crossbar from the previous operating window.

To provide better data utilization and reduce the number of transactions, Figure 4(c) demonstrates an efficient mapping of kernels in the crossbar as well as activation value in the input buffer. In this approach, the kernels and the input data within the operating window are sliced into columns. The same columns for different input channels are packed together and placed in the input buffer. The next columns are stacked on top of each other as highlighted by the light orange color in Figure 4(c). The kernels are also treated the same way. By doing that, when the operating window slides to the right (assuming stride is one), the left-most columns for all the input channels are shifted out and new data corresponding to the right-most columns are streamed into the buffer. There is no need to change the mapping of the kernels in the crossbar and they always reside in front of the right inputs. When the operating window reaches the last columns, it has to be shifted down and start from the most left column again. Therefore, the input buffer is refreshed and filled with data highlighted by the blue window in 4(b). As a result, maximum data is utilized when the operating window slides while the input buffer can be implemented as simply as possible.

In order to maximize the performance, we can exploit parallelization and pipelining. In case the crossbar dimension

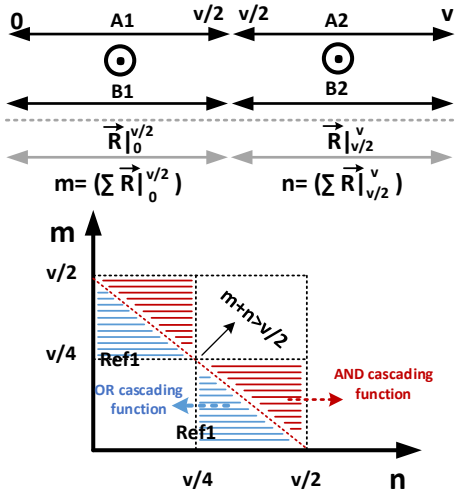

Fig. 5: Illustration of the regions where logical *AND* and *OR* cascading functions inject inaccuracy into the network.

is large enough, the computation for the current and next operating windows can be performed in parallel. As illustrated in Figure 4(b) and 4(c), an extra column (highlighted by bright orange) required for the next operating window is placed into the input buffer of the crossbar. Besides, we have to consider another column in the crossbar to be able to generate the value for both operating windows simultaneously. It has to be taken into account that this extra input set should not contribute to the computation of the current window. Therefore, the memristors located in the first column and in front of this extra input set should be programmed to logic value '0'. It is worth mentioning that the kernels for other output channels are programmed to different columns of the crossbar to maximize parallelization. However, in case the crossbar has a lower number of columns, we need to deploy more crossbars to avoid an excessive number of reprogrammings. Besides parallelization, the same pipelining approach presented in [38] can be applied in this work. Depending on the kernel size of the next layer in the network, when enough elements are produced for the output channels of the current layer, the operation can be started for the next layer.

V. Intra-Layer Accuracy Analysis

In Section IV, the proposed implementation was presented where a single SA can generate the activation value for the next BNN layer (see Figure 3). However, if the weights that are supposed to be in a single column of a crossbar cannot fit into it, they have to be split and mapped to more columns. In other words, if there are not enough memristors in a column of a crossbar to store a kernel (e.g., the blue kernel in Figure 3(c)), this kernel has to be broken into several parts each mapped to different columns. Therefore, the final activation value has to be calculated from the intermediate activation values obtained from different sets of columns. This is where inaccuracy is injected into the network with a particular probability distribution. In the following, the ideal situation is formulated where the crossbar size is equal to or greater than the vector size. $\overrightarrow{A}$ and $\overrightarrow{B}$ are the two input binary vectors, $\overrightarrow{R}$ is the result of XNOR operation between the two input vectors, and $\Sigma(\overrightarrow{R})$ produces the output of Popcount function on the binary vector $\overrightarrow{R}$.

Vector size = ν, *Crossbar size* = C, and $C \geq \nu$

input 1: $\overrightarrow{A}$, input 2: $\overrightarrow{B}$

$$\overrightarrow{R} = \overrightarrow{A} \odot \overrightarrow{B}$$

$$out_{golden}(\overrightarrow{R}) = \begin{cases} 1 & \text{if} \quad \Sigma(\overrightarrow{R}) > \nu/2 \\ 0 & \text{otherwise} \end{cases}$$

In case the crossbar size is not big enough, the formulation is changed as presented below. As an example, we assume the crossbar size is half of the vector size. Therefore, each vector has to be split into two parts and mapped to two columns of the crossbar.

Vector size = ν, *Crossbar size*: $C = \nu/2$

input 1: $\overrightarrow{A}|_0^{\nu/2}, \overrightarrow{A}|_{\nu/2}^{\nu}$ where $\overrightarrow{A} = [\overrightarrow{A}|_0^{\nu/2}, \overrightarrow{A}|_{\nu/2}^{\nu}]$

input 2: $\overrightarrow{B}|_0^{\nu/2}, \overrightarrow{B}|_{\nu/2}^{\nu}$ where $\overrightarrow{B} = [\overrightarrow{B}|_0^{\nu/2}, \overrightarrow{B}|_{\nu/2}^{\nu}]$

$$\overrightarrow{R}|_0^{\nu/2} = \overrightarrow{A}|_0^{\nu/2} \odot \overrightarrow{B}|_0^{\nu/2} \qquad \overrightarrow{R}|_{\nu/2}^{\nu} = \overrightarrow{A}|_{\nu/2}^{\nu} \odot \overrightarrow{B}|_{\nu/2}^{\nu}$$

$$out_{p1}(\overrightarrow{R}|_0^{\nu/2}) = \begin{cases} 1 & \text{if} \quad \Sigma(\overrightarrow{R}|_0^{\nu/2}) > (\nu/2)/2 \\ 0 & \text{otherwise} \end{cases}$$

$$out_{p2}(\overrightarrow{R}|_{\nu/2}^{\nu}) = \begin{cases} 1 & \text{if} \quad \Sigma(\overrightarrow{R}|_{\nu/2}^{\nu}) > (\nu/2)/2 \\ 0 & \text{otherwise} \end{cases}$$

Since we mapped the vector into two columns, two intermediate activation values (out_{p1}, out_{p2}) are obtained. The final value depends on the *cascading function*, which receives intermediate activation values (out_{p1}, out_{p2}) as input and produces the final activation value. This function can be a simple logical *AND* or *OR* function. The following is an example of the AND ($\wedge$) cascading function.

$$out(\overrightarrow{R}|_{\nu/2}^{\nu}, \overrightarrow{R}|_0^{\nu/2}) = out_{p2}(\overrightarrow{R}|_{\nu/2}^{\nu}) \wedge out_{p1}(\overrightarrow{R}|_0^{\nu/2})$$

In the case of logical *AND* as an example, the following conditions show the scenarios where the output of the cascading function differs from the golden output. This is also illustrated in Figure 5. The y and x axes are the output of Popcount (Σ) obtained from the result of the first ($\overrightarrow{R}|_{\nu/2}^{\nu}$) and second parts ($\overrightarrow{R}|_0^{\nu/2}$) of the output vector. The red and blue regions indicate inaccurate results by the *AND* and *OR* functions.

$out(\overrightarrow{R}|_0^{\nu/2}, \overrightarrow{R}|_{\nu/2}^{\nu}) \neq out_{golden}(\overrightarrow{R})$ if:

$$\begin{cases} \Sigma(\overrightarrow{R}|_0^{\nu/2}) + \Sigma(\overrightarrow{R}|_{\nu/2}^{\nu}) = \Sigma(\overrightarrow{R}|_0^{\nu}) > \nu/2 \\ \Sigma(\overrightarrow{R}|_0^{\nu/2}) < \nu/4 \quad \vee \quad \Sigma(\overrightarrow{R}|_{\nu/2}^{\nu}) < \nu/4 \end{cases}$$

According to the aforementioned conditions, the output of AND cascading function does not generate the expected results only if 1) the final activation value is expected to be '1' ($\Sigma(\overrightarrow{R}|_0^{\nu}) > \nu/2$) and 2) one of the intermediate activation

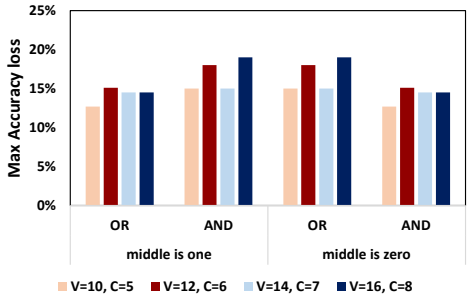

Fig. 6: Maximum accuracy loss simulated for all possible input vectors for different vector sizes (V), crossbar size (C), and cascading functions.

values regarding the partial output vectors ($\overrightarrow{R}|_0^{\nu/2}$ or $\overrightarrow{R}|_{\nu/2}^{\nu}$) is '0'. Hence, the final result, which is generated using logical AND between the output of the two intermediate activation values from $\overrightarrow{R}|_0^{\nu/2}$ and $\overrightarrow{R}|_{\nu/2}^{\nu}$ vectors would be '0'. We illustrate these two conditions in Figure 5. The region above the $m+n=\nu/2$ line satisfies the first condition. In this region, the data points that fall into the two triangles, highlighted in red, meet the second condition. It should be noted that the condition where the expected final activation value is '0', but both partial activation values would be '1' never happens.

The combinations of input vectors for a given (m, n) in these regions (blue for OR and red for AND cascading function in Figure 5) is calculated based on Equation 5. m and n are the outputs of the popcount function for the two partial output vectors resulting from XNOR operations. Accordingly, Equation 6 calculates all the possible combinations of input vectors that fall into the 'Solution Set'. The solution set for the AND cascading function is highlighted in red in Figure 5. According to these two equations, Figure 6 depicts the maximum accuracy loss for two cascading functions considering two boundary conditions. This accuracy loss is defined as the percentage of vectors out of all possible combinations where the cascading function fails to produce the correct result. The boundary condition determines the output of SA in case the data is the same as the reference. This is done by generating all the combinations of input sets to verify the Equation 5. We observe that the accuracy loss does not have considerable changes over vector sizes as the relative area associated with inaccurate region remains the same (Figure 5). It should be noted that this accuracy loss in Figure 6 should not be confused with the accuracy of an entire BNN.

$$N(m,n) = ({}_mC_{\nu/2} * 2^m * 2^{\nu/2-m}) * \\ ({}_nC_{\nu/2} * 2^n * 2^{\nu/2-n}) = 2^\nu * ({}_mC_{\nu/2} * {}_nC_{\nu/2}) \quad (5)$$

$$TN_{(AND)} = \sum_{(m,n)\in Solution\ Set} N(m,n) \quad (6)$$

In the aforementioned example (see Figure 6), the activation value for each partial output vector ($\overrightarrow{R}|_0^{\nu/2}$ and $\overrightarrow{R}|_{\nu/2}^{\nu}$) are obtained by only employing one reference (i.e., $\nu/4$). In order to reduce the accuracy loss, more references can be considered. This leads to more intermediate results, which provide us with more information as well as the flexibility to have advanced cascading functions. However, we should take into account that adding references increases the hardware complexity of SA. Next, we investigate a scenario where SAs have two references.

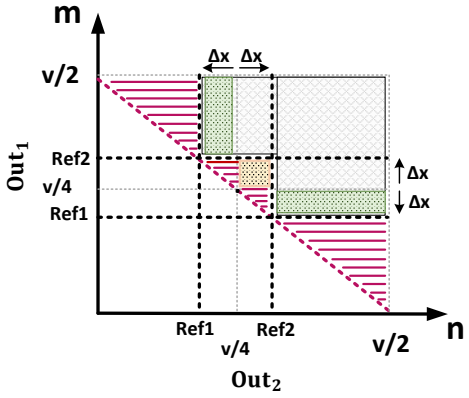

Fig. 7: Illustration of the scenario where there are two references for each partial output vector. The value of references (or Δx) should be defined in a way that the cascading function covers more area above $m+n=\nu/2$ line.

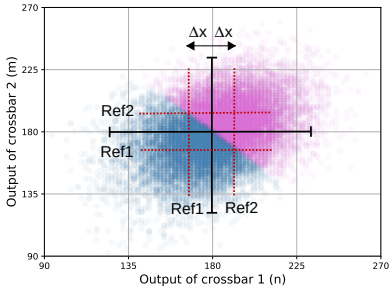

Fig. 8: Data points for two partial output vectors obtained from one layer of CNN-1 network using MNIST dataset. The size of each vector ($\nu/2$) is 360. The figure clearly shows a non-uniform distribution of data points.

Two references: Figure 7 illustrates the scenario where SAs have two references. This figure provides insight into the impact of references and their value on accuracy. Similar to Figure 5, m and n are the outputs of the popcount

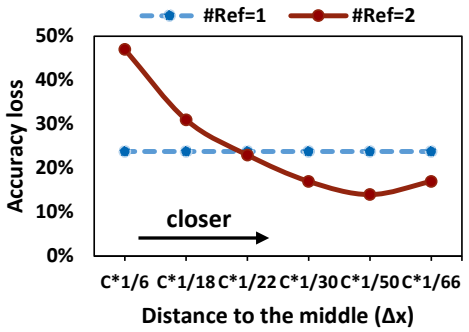

Fig. 9: Accuracy loss based on the value of Δx presented relative to the crossbar size (C).

function for the two partial output vectors resulting from XNOR operations. The goal is to determine where to put the references (or determine Δx) to minimize the accuracy loss (or minimize highlighted red region in Figure 7). We assume the references are placed symmetrically around the center point to simplify the analysis. As mentioned before, we need a cascading function to receive intermediate activation values (SA_{out1}, SA_{out2}), generated by the SAs, and produce the final activation value. In the following, we describe one example of a cascading function.

F= **[**$(SA_{out2} > Ref2) \wedge (SA_{out1} > Ref1)$**]** $\vee$ **[**$(SA_{out2} > Ref1) \wedge (SA_{out1} > Ref2)$**]**

According to the above cascading function, the final activation value would be 1 only if one of the outputs is higher than Ref_2 while the other one is higher than Ref_1. In Figure 7, the area covered by this function is highlighted in gray and green. As we can see, this function cannot cover the entire area above the $m+n = \nu/2$ line. The uncovered part is highlighted in red. Compared to the scenario where we have only one reference (see Figure 5), some new regions are covered (highlighted in green) while one region is left out (highlighted in yellow). Depending on the value of Δx, the size of these regions changes. If the data points were distributed uniformly, we could easily obtain the optimum Δx where the area covered by this cascading function is maximum. However, according to Equation 5 (or Figure 8), the data is not uniformly distributed. Hence, simulation can help to find the optimum Δx. Figure 9 shows the accuracy of this cascading function over different Δx. In this figure, the value of Δx is presented relative to the crossbar size ('C') which is equal to $\nu/2$. According to this figure, using two references can lead to better accuracy than one reference (dashed blue line) if we can find the optimum values for the references. In addition, it is worth clarifying that accuracy results in Figure 9 are not the final BNN accuracy.

It should be noted that when we have two references, putting one reference in the center does not make sense since the area covered by the second reference would be either a subset or superset of the first reference. However, this is not the case when we have three references. In the following, we elaborate on the scenario where we have three references. Finally, we should consider that the data sets provided to any neural network should follow the same probability distribution as the training data set. This should be taken into account in any neural network, including BCIM. Hence, if a new dataset has a different distribution, not only the network has to be retrained, but also the analysis for finding the optimal references should be performed again. As an example, Figure 8 shows the data points for the MNIST dataset. Of course, if a new dataset has a different distribution, the network should be retrained, the data points would be different, and the references should be adjusted.

Three references: Figure 10 presents an example where three references (Ref0, Ref1, Ref2) are considered to generate an activation value. In the ideal scenario where there is no need to split the vectors, the reference, obtained from Equation 4, is equal to $\nu/2$. However, for the scenario where we have to split the vector into two parts, the reference for the two

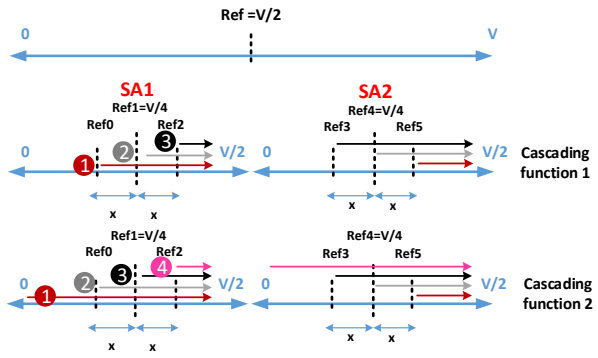

Fig. 10: Illustration of two cascading functions where two auxiliary references are added to the main reference.

partial output vectors, based on the same equation, should be $\nu/4$. This is called primary reference and indicated by "Ref1" in Figure 10. In this example, next to the primary reference, we utilize two more -auxiliary references- to improve accuracy. Next, we investigate the implication of the number of auxiliary references as well as their actual values on accuracy loss.

Considering the aforementioned scenario where three references are employed, we can produce three intermediate values for each of the partial output vectors. Hence, the final activation value should be decided based on these six binary values. In the following, we describe two possible cascading functions to produce the final activation value. They are also illustrated in Figure 10.

Prime implicant of cascading function 1:
F1= **[**$(SA_{out2} > Ref5) \wedge (SA_{out1} > Ref0)$**]** $\vee$ **[**$(SA_{out2} > Ref4) \wedge (SA_{out1} > Ref1)$**]** $\vee$ **[**$(SA_{out2} > Ref3) \wedge (SA_{out1} > Ref2)$**]**

Prime implicant of cascading function 2:
F2= $[(SA_{out2} > Ref5)] \vee [(SA_{out1} > Ref2)] \vee [(SA_{out2} > Ref3) \wedge (SA_{out1} > Ref1)] \vee [(SA_{out2} > Ref4) \wedge (SA_{out1} > Ref0)]$

The numbers in Figure 10 indicate the different conditions where the cascading function produces logic 1 as the final activation value. As an example, the first cascading function comprises three conditions, where meeting each can set the final activation value to 1. Each number in Figure 10 illustrates one term in the prime implicant of cascading function 1. These are based on the fact that the summation of two Popcount functions (Σ) obtained from two output vectors should be greater than half of the original vector size (Equation 4). This function always sets the activation value to one accurately (true positive), but it sometimes misses to set it to one (false negative). Considering that, the second cascading function makes the conditions more relaxed. The probability of accuracy loss for these two functions is computed as follows.

$P_{Loss}(F1) =$
$\mathbf{P}([SA_{out2} > Ref5 \wedge SA_{out1} < Ref0] \wedge [SA_{out2} + SA_{out1} > \nu/2]) + \mathbf{P}([SA_{out2} < Ref3 \wedge SA_{out1} > Ref2] \wedge [SA_{out2} + SA_{out1} > \nu/2]) + \mathbf{P}([Ref4 < SA_{out2} <$

TABLE I: Typologies of the BNNs and their software accuracy

Name	Topology	Dataset	Accuracy
LeNet-5	5x5,6 - 2x2 Pool - 5x5,16 - 2x2 Pool - FC(120) - FC(84) - FC(10)	MNIST	98%
CNN-1	5x5,5 - 2x2 Pool - FC(720) - FC(70) - FC(10)	MNIST	97%
CNN-2	7x7,10 - 2x2 Pool - FC(1210) - FC(1210) - FC(10)	MNIST	98%
MLP-S	FC(784) - FC(500) - FC(250) - FC(10)	MNIST	97%
MLP-M	FC(784) - FC(1000) - FC(500) - FC(250) - FC(10)	MNIST	98.2%
MLP-L	FC(784) - FC(1500) - FC(1000) - FC(500) - FC(10)	MNIST	98.4%
AlexNet	11x11,96 - 3x3 Pool/2 - 5x5,256 - 3x3 Pool/2 - 3x3, 384 - 3x3,384 - 3x3,256 - 3x3 Pool/2 - 3x3 AvgPool/2 - FC(4096) - FC(4096) - FC(10)	(enlarged) CIFAR-10	80%

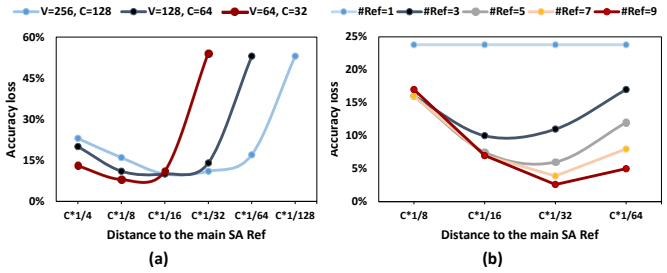

Fig. 11: (a) Accuracy loss based on the distance of two auxiliary references to the main reference (b) effect of the number of auxiliary references on accuracy.

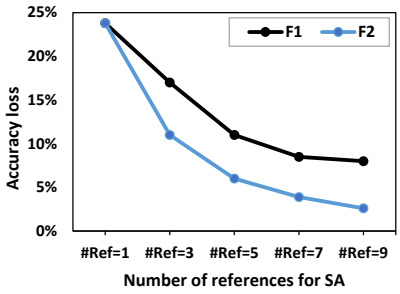

Fig. 12: Impact of the two cascading functions illustrated in Figure 10 on accuracy loss.

$Ref5] \wedge [Ref0 < SA_{out1} < Ref1] \wedge [SA_{out1} + SA_{out2} > \nu/2]) + \mathbf{P}([Ref3 < SA_{out2} < Ref4] \wedge [Ref1 < SA_{out1} < Ref2] \wedge [SA_{out1} + SA_{out2} > \nu/2])$

$P_{Loss}(F2) =$
$\mathbf{P}([SA_{out2} > Ref5] \wedge [SA_{out2} + SA_{out1} < \nu/2]) + \mathbf{P}([SA_{out1} > Ref2] \wedge [SA_{out2} + SA_{out1} < \nu/2]) + \mathbf{P}([Ref4 > SA_{out2} > Ref3] \wedge [Ref2 > SA_{out1} > Ref1] \wedge [SA_{out2} + SA_{out1} < \nu/2]) + \mathbf{P}([Ref5 > SA_{out2} > Ref4] \wedge [Ref1 > SA_{out1} > Ref0] \wedge [SA_{out2} + SA_{out1} < \nu/2])$

An important parameter that has a remarkable impact on the accuracy loss is the distance of auxiliary references to the main reference (*"x"* in Figure 10). This is quite dependent on the distribution of data. Hence, the designer can analyze the network and, based on that, find the proper value for the references where the accuracy loss is minimized. Figure 11(a) demonstrates the impact of this parameter for cascading function 2 assuming a normal distribution. This is presented for different crossbar sizes ("*c*"). The distance to the main reference is shown relative to the crossbar size. The figure indicates the importance of the values for the references and how considerably they can change the accuracy loss. Another important parameter is the number of references. The implication on accuracy can be comprehended from Figure 11(b). It is observed that by adding more references, an improvement in accuracy is reduced while more complexity is added to the hardware. Finally, the impact of the cascading functions on accuracy is evaluated in Figure 12 over a different number of references. The same two methods presented in Figure 10 are also used for the situation where we have more than three references. The figure indicates that choosing a proper function can help the accuracy of the system remarkably.

VI. Evaluation

A. Simulation setup

Our simulation results are obtained by creating our PyTorch-based platform [28]. This platform is able to evaluate the accuracy, energy, and latency of different networks containing binarized and non-binarized layers. The software is written in a modular way to flexibly change network structure as well as different circuit-level parameters. The system runs at a clock frequency of 1GHz. The width of the databus transferring data between the crossbars is 32 bits. This is required for communication between layers. Based on the 32nm technology node, transferring data to store it in an input buffer consumes 5mW [39], [40]. The energy and latency number of the "Shift and Add" unit required for non-binarized layers taken from [40]. In all the simulations, the crossbar size is 512×512 [41]. We use an analytical model based on a small ReRAM memristor prototype and extend the memory to the required size. The LRS and HRS for the memristors are 5k and 1G, respectively. The read voltage is 0.2V and the latency of the crossbar to charge the bit-lines is considered to be 10 ns. The model is acquired from the results of the EU project MNEMOSENE [42].

The specification of the sensing mechanism is taken from [43]. The energy per ADC read is 12 pJ, and its latency is 3 ns. Besides, the energy of SA is assumed to be 10 fJ with 1 ns latency. In case our SA needs more references (e.g., 3 references), its energy and latency get increased linearly by the number of references [44]. A maximum of 3 references for a SA are considered in the simulations. The energy and latency numbers are parameterized in the simulation platform and can be changed based on different circuit designs.

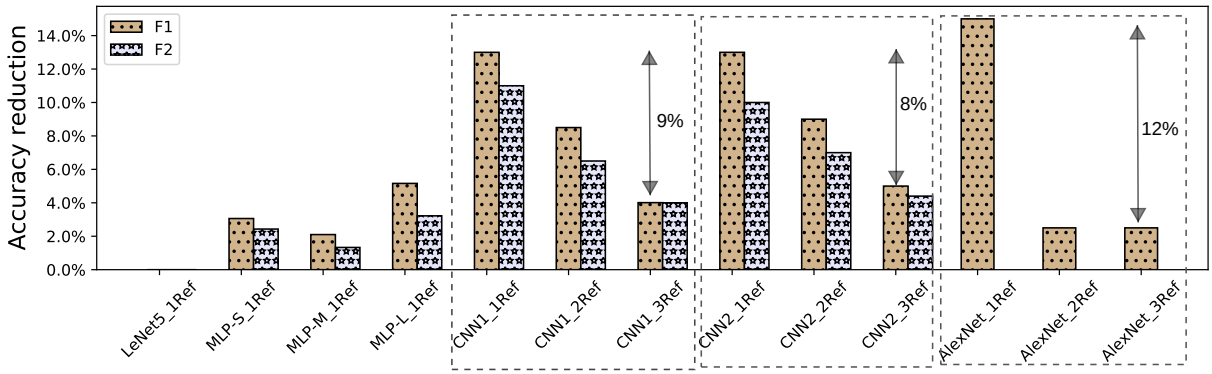

Fig. 13: Accuracy reduction for different network structures due to the crossbar size limitation and breaking the vectors over more crossbars. F1 and F2 are two cascading functions. For SAs with one or two references, cascading functions are 'AND' and 'OR'. For SAs with three references, cascading functions are described in Figure 10.

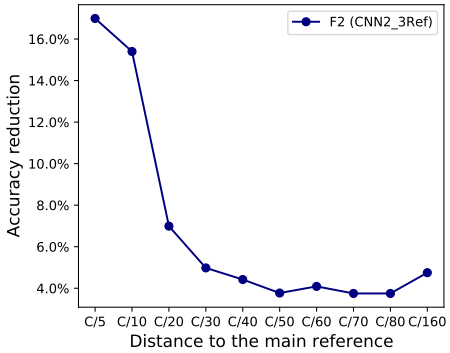

Fig. 14: Impact of auxiliary references and their distance from the main reference on accuracy loss. The simulation is performed for the CNN2 network employing cascading function F2 which is described in Figure 10.

Our benchmark (MlBench) comprises 7 BNNs for machine learning applications. The structure of each network is listed in Table I. LeNet-5, CNN-1, CNN-2, and AlexNet are convolutional networks, and MLP-S/M/L are multilayer perceptrons (MLPs) with different network scales [45]. We use MNIST and CIFAR-10 datasets to evaluate our networks. The input images for CIFAR-10 are enlarged to 256×256 required for AlexNet.

We compare our design with a recent work published in one of the leading journals in this field [27]. For this work, we instantiate the digital post-processing units (popcount) for every 16 columns of the crossbar instead of sequentially operating over all the columns (see Figure 3(a)). This diminishes the latency overhead of digital processing for the baseline. Besides, the second baseline is called "Exact computing", where we use ADC instead of SA to avoid any accuracy loss at the cost of energy and performance reduction. As explained before, since the crossbar dimensions are often smaller than the layer of a network, each crossbar is responsible for generating a partial result. Then, this partial result has to be aggregated and passed to the next layer. However, since the proposed technique uses SA, these partial results are turned into binary values generated by SAs. Hence, some of the information is lost, and it causes accuracy loss (see Section V). However, in case we deploy ADC instead of SA, this aforementioned source of accuracy loss is avoided.

B. Result and discussion

In the following, first, we present the total accuracy loss for different networks. Second, we evaluate the design in terms of energy and performance and elaborate more on our observations.

Accuracy analysis

Figure 13 depicts the accuracy loss using our proposed approach compared to the software implementation. The figure presents the results for the benchmarks considering different cascading functions (see Figure 10). Depending on the size of the layers in a network, we can see whether we have an accuracy loss or not.

1) Small-size network layers (LeNet5): Since the size of layers in the LeNet-5 network is within the range of crossbar size, no accuracy loss is observed. To clarify more, we can consider one of the fully connected layers -FC(84)- where there are 84 output activations. This layer receives 120 inputs. Hence, the number of memristor devices we need for a single column in order to accommodate this vector is 120×2 (see Figure 3(c)). Therefore, considering the crossbar size, there is no need to break the vector and reduce the accuracy.

2) Medium- and large-size network layers: The rest of the networks used in our simulations have larger layers to fit in one crossbar. Therefore, each layer has to be broken and mapped into several crossbars. This is where inaccuracy is injected into the network. Figure 13 shows that accuracy reduction using only SA with 1 reference is up to 14%. This accuracy loss is much less for MLP networks. Considering CNNs (CNN1, CNN2, and AlexNet), adding two more references to the SA can improve the accuracy by up to 12%. This means that only using three references in the SA can provide a decent accuracy loss of around 2%. It is worth mentioning that in case we want to perform computation "precisely", an ADC should be used in the design. One can interpret an 8-bit ADC as a SA with 256 reference levels. Therefore, Figure 13 shows that with a smart

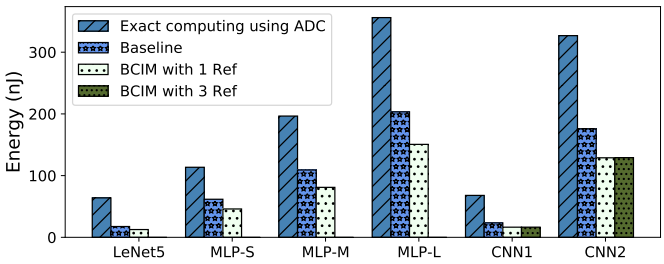

Fig. 15: Energy consumption of BCIM compared to the baseline as well the design performing precise computing using ADC.

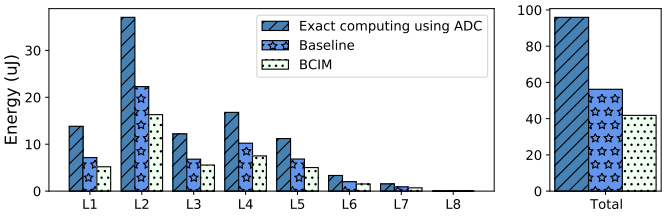

Fig. 16: Energy breakdown as well as total energy consumption of AlexNet for BCIM, baseline, and exact computing designs.

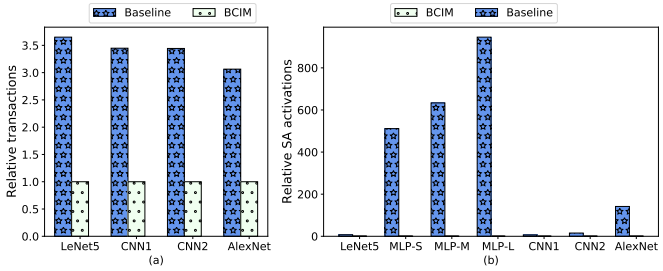

Fig. 17: (a) Number of transactions between crossbars and (b) number of SA activation for entire networks (normalized to BCIM).

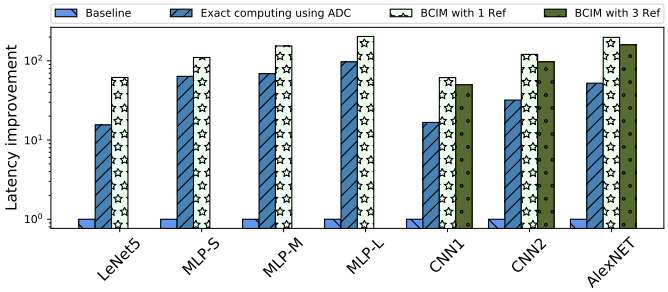

Fig. 18: Latency improvement of BCIM compared to the baseline and exact computing design.

selection of three references out of those 256 references, only 2% accuracy loss can be observed. In the case of AlexNet, since the size of layers is extremely large, layers are broken and mapped into more crossbars. Therefore, more options are available on how to perform the cascading function. The detail of the cascading function used for AlexNet can be found in [28].

Besides the number of references, another important parameter that can have a remarkable impact on accuracy is the actual value of references. Figure 14 depicts the implication of positioning the references on the accuracy loss. The simulation is performed for the CNN2 network with three references by changing the distance of auxiliary references to the main reference ("x" in Figure 10). The distance is relative to the crossbar size ("C"). Placing the references far from or too close to each other reduces their efficiency in eliminating the cases where inaccurate activation values are generated. Therefore, the designer should find the optimal value for the references by profiling the network.

Energy analysis

Figure 15 presents the energy numbers of different networks for the classification of one input image. The figure shows the energy number for 1) the baseline, 2) BCIM, and 3) the design where we want to do exact computing using ADC. The result indicates BCIM can achieve around 40% energy improvement compared to the baseline. Besides, BCIM can reduce the energy 5× compared to the design where exact computing is performed. In addition, we show the energy breakdown of AlexNet as well as its total energy in Figure 16. It is clear that energy consumption has a strong correlation with layer size and the amount of computation that has to be done in a layer. In addition, although the last layer is not binarized, its contribution to the total energy is very limited. This is due to the size of this layer compared to other layers (impact of output binarization on accuracy shown in Figure 2).

Comparing BCIM with the baseline in terms of energy, there are two factors that contribute to this energy improvement. First, the number of transactions required between the layers (or crossbars) is 3 times less in BCIM compared to the baseline. We present this relative comparison in Figure 17(a). This improvement is due to the mapping of activations and weights into the input buffer and the crossbar, respectively (see Figure 4). Second, the number of times SA is activated in BCIM is less than the baseline. We show this in Figure 17(b). It is worth mentioning that in both designs, the major contributor to the energy is the crossbar rather than the periphery or the data transfer between the crossbar. Hence, this is the reason BCIM archives marginal improvement in terms of energy compared to the baseline. In the future, if the energy consumption of memristor devices improves, then these two factors will play a major role in the energy.

In case we aim for exact computing using ADC, the total energy consumption significantly increases. The major contributor to this rise in energy is the costly ADC component. As mentioned in the Simulation setup section, ADC consumes 12 pJ per conversion. Considering the number of conversions required per crossbar activation, this imposes significantly more energy consumption on the system than a SA with around 10 fJ energy per sensing.

Latency analysis

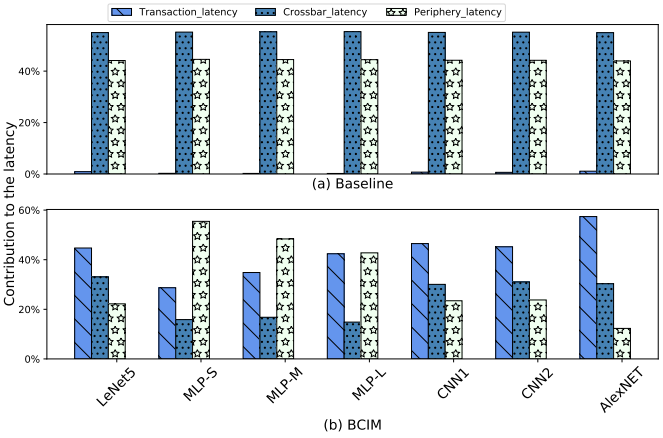

Fig. 19: Contribution of the different parts of the design to the total latency for (a) Baseline and (b) BCIM.

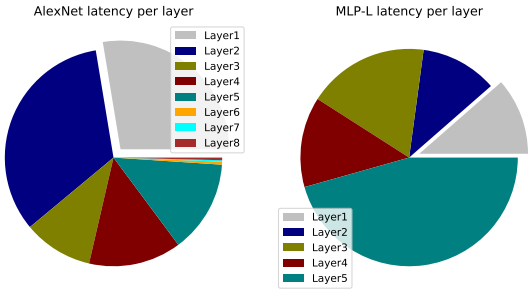

Fig. 20: Latency breakdown for different layers of AlexNet (left) and MLP-L (right)

Figure 18 shows the relative latency improvement for our different networks normalized to the baseline. BCIM achieves up to 100× improvement compared to the baseline. Besides, compared to the design where we perform exact computing, BCIM improves the latency by more than 3×.

There are two major factors involved in BCIM latency improvement compared to the baseline. We explain them in the following.

1) Mapping of weights into the crossbar: As illustrated in Figure 3(b), due to the way weights are mapped to the crossbar as well as the way computation is performed in the baseline, the output activation values are produced sequentially. This means at each time step, one row of the crossbar associated with one output value is activated. However, because of the weight mapping and the way we performed XNOR operations, the output activation values are produced in parallel (see Figure 3(c)). This improves the total latency of the network considerably. Figure 19(a) presents the contribution of different parts of the design for the baseline to the total latency. As we can see, the crossbar has a major contribution. However, in BCIM, by changing the mapping of the weights, the contribution of the crossbar was reduced significantly. We can observe this in Figure 19(b). In addition to the crossbar, the total latency of the periphery is high in the baseline, as we can see in Figure 19(b). This is because every time a crossbar row is activated, the sensing and all the digital peripheries should be conducted.

2) Mapping of activation value into the input buffer: Figure 19(b) shows that by changing the weight mapping and reducing the contribution of crossbar and periphery in the total latency, data communication becomes dominant in most cases. Hence, it is critical to have an optimized mapping of activation value into the input buffer in order to minimize the data communication overhead. As we can see in Figure 19(b), even after the optimization of the input buffer, the communication overhead is still dominant.

BCIM improves the latency by more than 3× compared to the exact computing design (see Figure 18). First, ADC imposes more latency in the periphery compared to SA in order to perform a conversion. Second, due to the high area consumption of ADC compared to a SA, more bit-lines in the crossbar should be shared by an ADC. Hence, this also increases the latency of this design.

Figure 19 presents the contribution of three major latency consumers (transaction, crossbar, and periphery) for (a) the baselines, (b) and BCIM. Considering the baseline, crossbars contribute more to the latency than other contributors. This is due to the large amount of sequential computation that has to be performed in the crossbar. However, in the case of BCIM, thanks to the efficient mapping and implementation, the crossbar is not the major contributor anymore. However, this contribution is higher for convolutional networks compared to MLPs. In a convolution layer, a kernel (mapped to a crossbar) has to slide over the entire input data in order to produce output activation values. This means more computation is performed on each crossbar with one input set than in an MLP network. Figure 20 shows the latency breakdown per layer for AlexNet. The figure again demonstrates that the convolution layers have more contributions than fully connected layers within a network. This means that if we have to satisfy higher performance requirements in our design, more resources should be dedicated to these layers. Considering Figure 20(b), where we provide the breakdown of the MLP-L network, the last layer has the major contribution to the latency since this layer is not binarized, and ADCs have to be employed. Using ADC considerably increases the contribution of peripheral circuits in latency. This is the reason that the periphery contributes more to the latency in our MLP networks (see Figure 19(b)).

VII. Conclusion and Future Directions

This paper proposed a novel in-memory memristor-based design that substantially improves both the latency and energy efficiency of BNN networks. The proposed XNOR-based BNN design replaces the ADC and digital post-processing functionality with a SA with adjusted reference(s) while maximizing parallelization and resource utilization in the design using a novel mapping of weights and activation values in the crossbar and its input buffer. The impact of SA references on the accuracy has been evaluated at the crossbar and network levels. The design was also evaluated in terms of energy and latency for different networks and datasets. This work is able to improve energy and latency up to 5× and 100× compared

to the baselines with marginal accuracy loss. In our future work, we will evaluate the design for larger and more complex networks and datasets to comprehend the impact of inaccuracy injected into intermediate layers on the overall accuracy of the networks.

REFERENCES

[1] D. W. Otter, J. R. Medina, and J. K. Kalita, "A survey of the usages of deep learning for natural language processing," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 2, pp. 604–624, 2020.

[2] A. Dhillon and G. K. Verma, "Convolutional neural network: a review of models, methodologies and applications to object detection," *Progress in Artificial Intelligence*, vol. 9, no. 2, pp. 85–112, 2020.

[3] W. Wang, Y. Yang, X. Wang, W. Wang, and J. Li, "Development of convolutional neural network and its application in image classification: a survey," *Optical Engineering*, vol. 58, no. 4, p. 040901, 2019.

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of Deep Bidirectional Transformers for Language Understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[5] C. Yuan and S. S. Agaian, "A comprehensive review of binary neural network," *arXiv preprint arXiv:2110.06804*, 2021.

[6] O. Golonzka, U. Arslan, P. Bai, M. Bohr, O. Baykan, Y. Chang, A. Chaudhari, A. Chen, J. Clarke, C. Connor *et al.*, "Non-volatile RRAM embedded into 22FFL FinFET technology," in *2019 Symposium on VLSI Technology*. IEEE, 2019, pp. T230–T231.

[7] R. Dittmann, S. Menzel, and R. Waser, "Nanoionic memristive phenomena in metal oxides: the valence change mechanism," *Advances in Physics*, vol. 70, no. 2, pp. 155–349, 2021.

[8] M. Zahedi, M. A. Lebdeh, C. Bengel, D. Wouters, S. Menzel, M. Le Gallo, A. Sebastian, S. Wong, and S. Hamdioui, "Mnemosene: Tile architecture and simulator for memristor-based computation-in-memory," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 18, no. 3, pp. 1–24, 2022.

[9] L. Song, Y. Zhuo, X. Qian, H. Li, and Y. Chen, "GraphR: Accelerating graph processing using ReRAM," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 531–543.

[10] M. Zahedi, G. Custers, T. Shahroodi, G. Gaydadjiev, S. Wong, and S. Hamdioui, "SparseMEM: Energy-efficient Design for In-memory Sparse-based Graph Processing," in *2023 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2023, pp. 1–6.

[11] T. Shahroodi, M. Zahedi, A. Singh, S. Wong, and S. Hamdioui, "Krakenonmem: a memristor-augmented hw/sw framework for taxonomic profiling," in *Proceedings of the 36th ACM International Conference on Supercomputing*, 2022, pp. 1–14.

[12] F. Zokaee *et al.*, "Finder: Accelerating fm-index-based exact pattern matching in genomic sequences through reram technology," in *2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 2019, pp. 284–295.

[13] T. Shahroodi, M. Zahedi, C. Firtina, M. Alser, S. Wong, O. Mutlu, and S. Hamdioui, "Demeter: A fast and energy-efficient food profiler using hyperdimensional computing in memory," *IEEE Access*, vol. 10, pp. 82 493–82 510, 2022.

[14] G. Karunaratne, M. Le Gallo, M. Hersche, G. Cherubini, L. Benini, A. Sebastian, and A. Rahimi, "Energy efficient in-memory hyperdimensional encoding for spatio-temporal signal processing," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 5, pp. 1725–1729, 2021.

[15] M. Le Gallo *et al.*, "Compressed sensing with approximate message passing using in-memory computing," *IEEE Transactions on Electron Devices*, vol. 65, no. 10, pp. 4304–4312, 2018.

[16] B. Cambou *et al.*, "Cryptography with analog scheme using memristors," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 16, no. 4, pp. 1–30, 2020.

[17] M. Masoumi, "Novel Hybrid CMOS/Memristor Implementation of the AES Algorithm Robust Against Differential Power Analysis Attack," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 7, pp. 1314–1318, 2020.

[18] M. Zahedi, T. Shahroodi, G. Custers, A. Singh, S. Wong, and S. Hamdioui, "System design for computation-in-memory: From primitive to complex functions," in *2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE, 2022, pp. 1–6.

[19] J. D. Ferreira, G. Falcao, J. Gómez-Luna, M. Alser, L. Orosa, M. Sadrosadati, J. S. Kim, G. F. Oliveira, T. Shahroodi, A. Nori *et al.*, "pluto: Enabling massively parallel computation in dram via lookup tables," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 900–919.

[20] J. Chen, S. Wen, K. Shi, and Y. Yang, "Highly parallelized memristive binary neural network," *Neural Networks*, vol. 144, pp. 565–572, 2021.

[21] T. Tang, L. Xia, B. Li, Y. Wang, and H. Yang, "Binary convolutional neural network on rram," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2017, pp. 782–787.

[22] L. Huang, J. Diao, H. Nie, W. Wang, Z. Li, Q. Li, and H. Liu, "Memristor based binary convolutional neural network architecture with configurable neurons," *Frontiers in neuroscience*, vol. 15, p. 328, 2021.

[23] Y.-F. Qin, R. Kuang, X.-D. Huang, Y. Li, J. Chen, and X.-S. Miao, "Design of high robustness bnn inference accelerator based on binary memristors," *IEEE Transactions on Electron Devices*, vol. 67, no. 8, pp. 3435–3441, 2020.

[24] Y. Zhao, Y. Wang, R. Wang, Y. Rong, and X. Jiang, "A highly robust binary neural network inference accelerator based on binary memristors," *Electronics*, vol. 10, no. 21, p. 2600, 2021.

[25] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European conference on computer vision*. Springer, 2016, pp. 525–542.

[26] Y. Halawani, B. Mohammad, M. A. Lebdeh, M. Al-Qutayri, and S. F. Al-Sarawi, "Reram-based in-memory computing for search engine and neural network applications," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 388–397, 2019.

[27] T. Hirtzlin, M. Bocquet, B. Penkovsky, J.-O. Klein, E. Nowak, E. Vianello, J.-M. Portal, and D. Querlioz, "Digital biologically plausible implementation of binarized neural networks with differential hafnium oxide resistive memory arrays," *Frontiers in neuroscience*, vol. 13, p. 1383, 2020.

[28] "BCIM simulation platform." [Online]. Available: https://github.com/mahdi-zahedi/BCIM-Binary-Neural-Network-using-Computation-in-Memory-.git

[29] K. Fleck, U. Böttger, R. Waser, N. Aslam, S. Hoffmann-Eifert, and S. Menzel, "Energy dissipation during pulsed switching of strontium-titanate based resistive switching memory devices," in *2016 46th European Solid-State Device Research Conference (ESSDERC)*. IEEE, 2016, pp. 160–163.

[30] S. Hamdioui, H. A. Du Nguyen, M. Taouil, A. Sebastian, M. L. Gallo, S. Pande, S. Schaafsma, F. Catthoor, S. Das, F. G. Redondo, G. Karunaratne, A. Rahimi, and L. Benini, "Applications of computation-in-memory architectures based on memristive devices," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2019, pp. 486–491.

[31] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," *Advances in neural information processing systems*, vol. 28, 2015.

[32] S. Liang, S. Yin, L. Liu, W. Luk, and S. Wei, "Fp-bnn: Binarized neural network on fpga," *Neurocomputing*, vol. 275, pp. 1072–1086, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231217315655

[33] C. Fu, S. Zhu, H. Su, C.-E. Lee, and J. Zhao, "Towards fast and energy-efficient binarized neural network inference on fpga," *arXiv preprint arXiv:1810.02068*, 2018.

[34] H. Yang, M. Fritzsche, C. Bartz, and C. Meinel, "Bmxnet: An open-source binary neural network implementation based on mxnet," in *Proceedings of the 25th ACM international conference on Multimedia*, 2017, pp. 1209–1212.

[35] M. Hu, C. E. Graves, C. Li, Y. Li, N. Ge, E. Montgomery, N. Davila, H. Jiang, R. S. Williams, J. J. Yang *et al.*, "Memristor-based analog computation and neural network classification with a dot product engine," *Advanced Materials*, vol. 30, no. 9, p. 1705914, 2018.

[36] Y. Kim, W. H. Jeong, S. B. Tran, H. C. Woo, J. Kim, C. S. Hwang, K.-S. Min, and B. J. Choi, "Memristor crossbar array for binarized neural networks," *AIP Advances*, vol. 9, no. 4, p. 045131, 2019.

[37] D. Ahn, H. Oh, H. Kim, Y. Kim, and J.-J. Kim, "Maximizing parallel activation of word-lines in mram-based binary neural network accelerators," *IEEE Access*, vol. 9, pp. 141 961–141 969, 2021.

[38] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[39] A. Ankit, I. E. Hajj, S. R. Chalamalasetti, G. Ndu, M. Foltin, R. S. Williams, P. Faraboschi, W.-m. W. Hwu, J. P. Strachan, K. Roy *et al.*,

"Puma: A programmable ultra-efficient memristor-based accelerator for machine learning inference," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 715–731.

[40] M. Zahedi, R. van Duijnen, S. Wong, and S. Hamdioui, "Tile Architecture and Hardware Implementation for Computation-in-Memory," in *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2021, pp. 108–113.

[41] P. Narayanan, S. Ambrogio, A. Okazaki, K. Hosokawa, H. Tsai, A. Nomura, T. Yasuda, C. Mackin, S. Lewis, A. Friz *et al.*, "Fully on-chip mac at 14nm enabled by accurate row-wise programming of pcm-based weights and parallel vector-transport in duration-format," in *2021 Symposium on VLSI Technology*. IEEE, 2021, pp. 1–2.

[42] "Mnemosene project," http://www.mnemosene.eu, accessed: 2010-09-30.

[43] G. Karunaratne, M. Le Gallo, G. Cherubini, L. Benini, A. Rahimi, and A. Sebastian, "In-memory hyperdimensional computing," *Nature Electronics*, vol. 3, no. 6, pp. 327–337, 2020.

[44] L. Xie, H. A. Du Nguyen, J. Yu, A. Kaichouhi, M. Taouil, M. AlFailakawi, and S. Hamdioui, "Scouting logic: A novel memristor-based logic design for resistive computing," in *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2017, pp. 176–181.

[45] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A Novel Processing-in-memory Architecture for Neural Network Computation in ReRAM-based Main Memory," in *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3. IEEE Press, 2016, pp. 27–39.

Mahdi Zahedi received his M.Sc. degree in electrical engineering from the University of Tehran, Iran, 2018. He is currently pursuing his Ph.D. degree in electrical engineering at Delft University of Technology, Delft, The Netherlands. His current research interests include computer architecture and system-level HW/SW co-design.

Taha Shahroodi received a B.Sc. degree in computer engineering from the Sharif University of Technology (SUT), Tehran, Iran, in 2018, and an M.Sc. degree in computer science from the ETH Zürich, Zürich, Switzerland, in 2020. He is currently pursuing his Ph.D. at TU Delft, The Netherlands. His current research interests include bioinformatics, computer architecture, and hardware/software co-design.

Carlos Escuin Carlos Escuin received his B.Sc. degree in Computer Science from the Universidad de Zaragoza, Spain, and the M.Sc. degree in High-Performance Computing from the Universitat Politècnica de Catalunya, Spain, in 2016 and 2018, respectively. He is currently pursuing his Ph.D. degree in Computer Science at the Universidad de Zaragoza, Spain. His current research interests include computer architecture, memory hierarchy, cache memories, non-volatile memories, and computing in memory.

Georgi Gaydadjiev Georgi Gaydadjiev received his M.Sc. (1996) in Electrical Engineering and Ph.D. (2007) in Computer Engineering from Delft University of Technology. He is currently a full professor in Computer System Architectures at TU Delft, the Chair of Innovative Computer Architectures at the University of Groningen, and an honorary visiting professor of the Department of Computing at Imperial College. He holds three patents, co-authored close to 200 refereed scientific conferences and journal articles, and received three best paper awards. His research interests include: Computer (System) Architecture and Micro-architecture, Reconfigurable and Heterogeneous Computing, Distributed and Advanced Memory Systems, Design tools and Methodologies, Low-Power Architectures, Architectural Support for Compilers and Runtime Systems.

Stephan Wong is currently an associate professor at the Delft University of Technology, The Netherlands. He obtained his Ph.D. from the same university in December 2002. His Ph.D. thesis entitled "Microcoded Reconfigurable Embedded Processor" describes the MOLEN polymorphic processor, organization, and (micro-)architecture. His research interests include Reconfigurable Computing, Distributed Collaborative Computing, High-Performance Computing, Embedded Systems, and Hardware/Software Co-Design. He is also an IEEE Senior Member.

Said Hamdioui (Senior Member, IEEE) (http://www.ce.ewi.tudelft.nl/hamdioui/) is currently Chair Professor on Dependable and Emerging Computer Technologies and Head of the Computer Engineering Laboratory (CE-Lab) of the Delft University of Technology, the Netherlands. He received the MSEE and PhD degrees (both with honors) from TUDelft. Prior to joining TUDelft as a professor, Hamdioui worked for Intel (Californa, USA), Philips Semiconductors R&D (Crolles, France) and Philips/ NXP Semiconductors (Nijmegen, The Netherlands). His research focuses on two domains: Dependable CMOS nano-computing (including Reliability, Testability, Hardware Security) and emerging technologies and computing paradigms (including 3D stacked ICs, memristors for logic and storage, and in-memory-computing). Hamdioui owns two patents, has published one book and contributed to other two, and had co-authored over 250 conference and journal papers. He has consulted for many semiconductor companies in the area of memory testing.