Chapter

Study of RRAM-Based Binarized Neural Networks Inference Accelerators Using an RRAM Physics-Based Compact Model

Tommaso Zanotti, Paolo Pavan and Francesco Maria Puglisi

Abstract

In-memory computing hardware accelerators for binarized neural networks based on resistive RAM (RRAM) memory technologies represent a promising solution for enabling the execution of deep neural network algorithms on resource-constrained devices at the edge of the network. However, the intrinsic stochasticity and nonidealities of RRAM devices can easily lead to unreliable circuit operations if not appropriately considered during the design phase. In this chapter, analysis and design methodologies enabled by RRAM physics-based compact models of LIM and mixed-signal BNN inference accelerators are discussed. As a use case example, the UNIMORE RRAM physics-based compact model calibrated on an RRAM technology from the literature, is used to determine the performance vs. reliability trade-offs of different in-memory computing accelerators: i) a logic-in-memory accelerator based on the material implication logic, ii) a mixed-signal BNN accelerator, and iii) a hybrid accelerator enabling both computing paradigms on the same array. Finally, the performance of the three accelerators on a BNN inference task is compared and benchmarked with the state of the art.

Keywords: RRAM, logic-in-memory, binarized neural networks, compact modeling, in-memory computing

1. Introduction

The increasing adoption of devices for the internet of things (IoT) combined with the diffusion of data-driven computing approaches, such as artificial neural networks [1], is promoting innovation in multiple sectors. Thus, to sustain the diffusion of these technologies, a constant research effort is being directed to the development of ultra-low power computing devices and architectures that will enable the execution of complex computations directly on low-power devices at the edge of the network, reducing the burden on the cloud computing infrastructure. In-memory computing (IMC) architectures based on new nanoelectronics devices, are considered a key enabling technology. In these architectures, computations are executed directly inside or in the proximity of the memory array, removing the main source of inefficiencies, i.e., the von Neumann bottleneck (VNB), which is caused by the time and energy hungry data transfer between the memory and the processing unit. Among novel nonvolatile memory devices, resistive random access memory (RRAM) technologies are one of the most mature and are considered as the frontrunners for enabling the diffused adoption of IMC accelerators, thanks to their low-cost, back end of line compatibility, high-density, high switching speed, and relatively low programming voltages. Still, RRAM technologies present several nonideal effects (e.g., cycle-to-cycle (C2C) and device-to-device variations, and random telegraph noise (RTN)) which can negatively impact the reliability of RRAM-based circuits and limit the number of bits that can be reliably stored in a single device.

Thus, to address these nonideal effects appropriate design methodologies need to be followed, and IMC accelerators in which RRAM devices are used as binary elements preferred. Among these IMC accelerators, logic-in-memory (LIM) [2–5] and mixed-signal binarized neural networks (BNN) inference accelerators [6–8] are promising solutions for resource-constrained edge devices. LIM accelerators enable the in-memory computation of logic operations. BNNs [9] are neural networks in which neurons' weights and activations are encoded using a single bit, and 2D RRAM arrays can be used to compute vector-matrix multiplication (VMM) operations (i.e., the most executed computation in neural networks) in the analog domain, achieving high energy efficiency and performance.

In this chapter, analysis and design methodologies enabled by an RRAM physics-based compact model, are discussed and applied to LIM and mixed-signal BNN inference accelerators. As a use case example, the UNIMORE RRAM physics-based compact model [10, 11] calibrated on an RRAM technology from the literature [12], is used to extract performance vs. reliability trade-offs of different IMC accelerators: (i) a LIM accelerator based on the material implication logic, (ii) a mixed-signal BNN accelerator, and (iii) a hybrid accelerator enabling the coexistence of both computing paradigms on the same array. Finally, the performance of the three accelerators on a BNN inference task are compared and benchmarked with a state-of-the-art solution.

2. Resistive RAM memory technologies and compact modeling

RRAM technologies are considered a promising nonvolatile memory technology for next-generation embedded systems applications, thanks to their low fabrications cost, small feature size, back end of line compatibility, and performance. Specifically, RRAMs typically provide fast switching speed (i.e., < 10 ns [13]), low programming energy (i.e., < 1pJ [13]), long retention (i.e., > 10 years [13]), high endurance (i.e., 10^6–10^{12} [13]), and a relatively large memory window (i.e., > 10^2 [13]) when used as binary memory elements. These devices can be electrically switched between nonvolatile a low and a high resistive state (LRS and HRS, respectively) that can be used to encode a logic 1 and a logic 0, respectively. Depending on the materials used for their fabrication, different switching mechanisms can occur [14, 15]. In this chapter, bipolar metal-oxide-based RRAM cells that exhibit filamentary switching are considered, but the methods described apply also to other resistive switching memory technologies.

A bipolar metal-oxide-based RRAM is programmed into a LRS by biasing it with a sufficiently large positive voltage, which causes the bond breakage and drift toward the top electrode (TE) of oxygen ions in the metal oxide layer, resulting in the formation of an oxygen deficient conductive filament (CF) assisting ohmic-like drift conduction [14, 16]. During a device set, a current compliance (I_C) must be provided to prevent dielectric breakdown. The device is reset into a HRS by biasing it with a sufficiently negative voltage (i.e., V_{RESET}) which induces drift and recombination of oxygen ions with oxygen vacancies in the dielectric which partially dissolve the CF. The current conduction in HRS is associated with trap-assisted tunneling in the dielectric [14, 16]. Due to the physical mechanisms behind the device's operation, these devices present intrinsic nonidealities. Specifically, programming C2C variations of the CF morphology and the dielectric composition [17], result in stochastically distributed resistive states which can influence the available memory window and circuit reliability. Also, random telegraph noise (RTN), which is caused by the effect of interstitial oxygen ions and vacancies [11, 18], is typically observed in low voltage reads.

2.1 The UNIMORE RRAM physic-based compact model

When designing novel RRAM-based circuits, RRAM devices nonideal effects and complex switching mechanism, result in non-trivial design constraints that need appropriate analysis tools to be studied. Thus, compact models of RRAM devices have been developed to enable circuit simulations [19]. Different compact models typically adopt different approximations, and two main categories can be identified, i.e., general purpose and physics-based compact models. The formers, typically adopt simpler equations that enable to simulate larger circuits, however at the expense of lower accuracy when simulating the device operation outside the operating range considered for parameter calibration, and a lack of a clear mapping between model and technology related parameters. Conversely, in physics-based compact models, parameters and equations are linked to the device physics, enabling to reproduce the device characteristic in multiple operating conditions [11] and simplifying the parameter extraction [20]. Thus, as a first step when studying the performance and reliability tradeoffs of novel RRAM-based circuits, physics-based compact models should be employed [21].

For the analysis and use-case examples reported in this chapter, the UNIMORE RRAM physics-based compact model available in [10, 11], is used. Other physics-based compact models could also be used to perform the analysis [21–23], however for other compact models, clear parameter extraction procedures are currently not available [20]. The UNIMORE RRAM compact model is implemented in Verilog-A and reproduces the effects of self-heating, C2C variability, multi-level RTN, and is completed by an automated parameter extraction procedure [20]. The RRAM device approximated by the compact model is shown in **Figure 1a** and **b** for a device in LRS and HRS, respectively. The compact model considers a CF and a dielectric barrier whose thickness x is modulated by means of a system of differential equations, which take into account thermal effects, the field-driven oxygen ions drift and recombination during the device reset, and the field-accelerated bond breaking during the device set [10, 11]. Internally, the compact model includes two sub modules that reproduce the effects of C2C variability and RTN, as shown in **Figure 1b**. A detailed description of the modeling of C2C variability and RTN is available in [11, 18]. The compact model was calibrated in [11] on 4 RRAM metal-oxide-based bipolar RRAM technologies. In the rest of the chapter, data and results of simulations are reported considering the model calibrated on the *TiN/HfO$_x$/AlO$_x$/Pt* RRAM technology from [12], which has an I_C of 100 μA. The calibrated compact model can well reproduce the experimental data in different operating conditions, as shown in **Figure 1c** and **d**, the

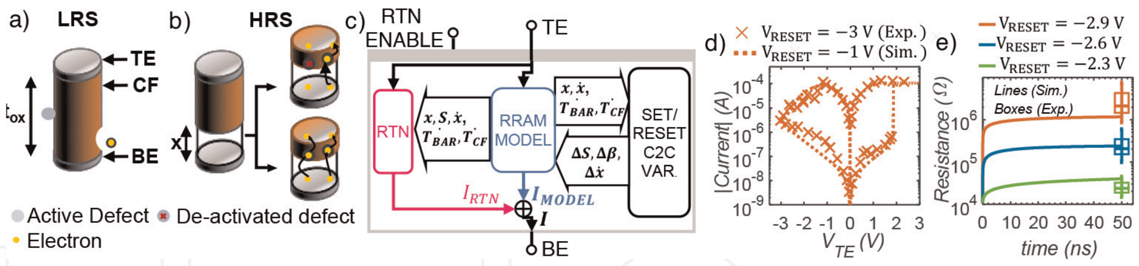


Figure 1.
Representation of an RRAM device as approximated in the compact model for a device in (a) LRS and (b) HRS, respectively. The defects causing RTN are also shown. (b) Functional block diagram of the compact model. The internal models for the device nonidealities are shown. Experimental and simulated, (c) quasi-static IV, and (d) pulsed reset characteristics of a TiN/HfOx/AlOx/Pt device. Data from [12].

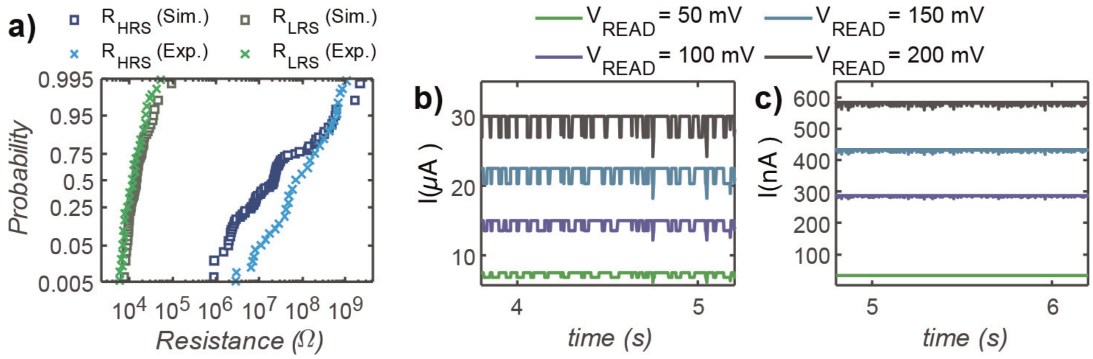


Figure 2.
Experimental and simulated cycle-to-cycle variability at different reset voltages. (b) and (c) simulated RTN at different read voltages for a device in (b) LRS, and (c) HRS.

experimental probability distributions of the resistive states, see **Figure 2a**, and multi-level RTN signals, as shown in **Figure 2b** and **c**. Simulations are performed with Cadence Virtuoso®.

2.2 Using compact models to study RRAM-based IMC architectures

As discussed in the following sections, IMC accelerators can be quite complex, and composed of a large number of RRAM devices and components. As the size of the accelerators increases, circuit simulation can quickly require excessive computing resources that are often not available. A possible approach is to study the problem at different abstraction layers. Specifically, the three steps approach shown in **Figure 3** can be followed. The first step of this methodology requires the electrical characterization of novel emerging nonvolatile memory devices, and the application of parameter extraction strategies to calibrate compact models on the specific technology. In the second step, the compact model is used to perform circuit simulations of smaller portion of the entire architecture. Specifically, the functionality of the core of operations of the in-memory computing framework of interest is first simulated using the compact model without including nonideal effects. The results of these simulations can provide indications regarding valid circuit operating points and the existing design tradeoffs. Parametric simulations including the device nonideal effects are then used to identify appropriate operating points that satisfy the desired reliability vs. performance tradeoffs, and to estimate the performance in the worst-case (WC in **Figure 3**) corners of the circuit. Finally, the results from the previous analysis can be used to estimate the performance of a more complex design based on the core

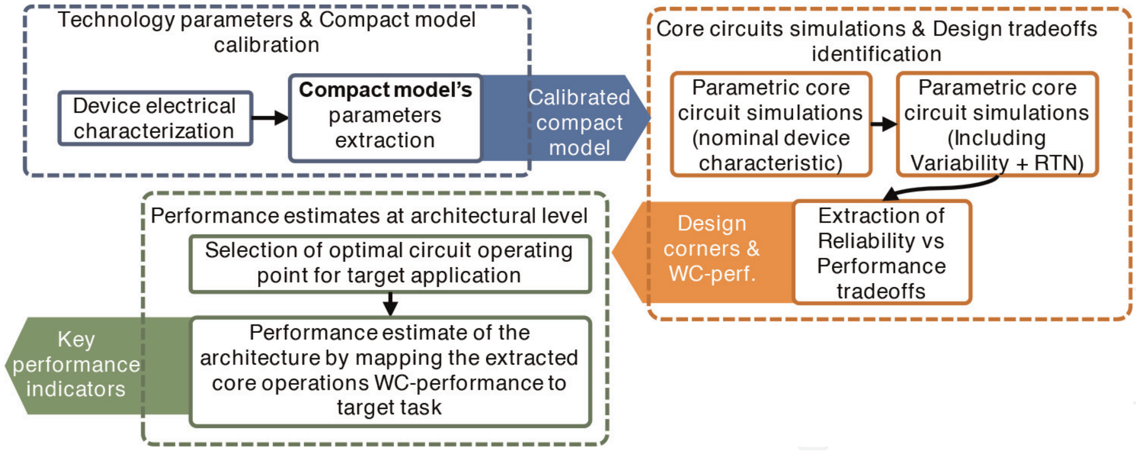

Technology parameters & Compact model calibration
Device electrical characterization
Compact model's parameters extraction
Calibrated compact model
Core circuits simulations & Design tradeoffs identification
Parametric core circuit simulations (nominal device characteristic)
Parametric core circuit simulations (Including Variability + RTN)
Extraction of Reliability vs Performance tradeoffs
Design corners & WC-perf.
Performance estimates at architectural level
Selection of optimal circuit operating point for target application
Performance estimate of the architecture by mapping the extracted core operations WC-performance to target task
Key performance indicators

Figure 3.
Flow-chart of the methodology enabled by the RRAM physics-based compact model that can be used to design reliable RRAM-based in-memory computing architectures and to estimate their performance.

operations studied with the compact model. Specifically, the worst-case performance estimates for the core operations are mapped to each operation executed at the architectural level. Although this step introduces several approximations, the use of the worst-case performance for each operation results in an underestimation of the energy efficiency therefore providing a sufficient headroom to account, to first order, for additional energy overheads possibly existing in the complete architecture.

A use case example of this methodology is discussed in the following sections of this chapter. In Section 3, the compact model is used to study the performance and reliability of the core operations of a LIM architecture, while in Section 5 the results of analysis enabled by the compact model are used to estimate the performance, at a higher abstraction level, of an inference accelerator based on BNNs.

3. Logic-in-memory with RRAM devices

In recent years, several IMC hardware accelerators based on resistive memory technologies have been proposed in the literature as a solution for improving computation efficiency, especially for data intensive tasks. Among these accelerators, LIM circuits enable the computation of logic operations on arrays of RRAM devices. Two different approaches can be commonly distinguished, i.e., stateful and non-stateful computing paradigms. Stateful LIM frameworks include the material implication [4, 24] and the memristor-aided logics (MAGIC) [2]. In these approaches, RRAM devices are treated at the same time as computing and storing elements. When performing an operation, inputs are stored as the resistive state of RRAM devices, and by using specific voltage pulses, the resistive state of an output device is either changed or preserved depending on the input combination. Non-stateful LIM approaches such as the scouting logic [25] and the smart material implication logic (SIMPLY) [26], instead, exploit the peripheral circuit to perform part of the computations. Inputs are encoded in the nonvolatile resistance of RRAM devices which are read in parallel with small voltages (i.e., ≈100 mV). The voltage at a specific circuit node changes depending on the input combination, and by sensing this voltage it is possible to implement different logic operations.

In the following subsections, the compact model is used to study the performance and reliability of a stateful and a non-stateful LIM frameworks based on the material implication logic. In addition, in Section 3.3 the performance and characteristics of different LIM frameworks are compared considering the computation of a 1-bit full addition operation as a benchmark.

3.1 RRAM-based material implication logic

The material implication logic is type of stateful logic that is functionally complete [27] and is based on two core operations, the IMPLY and the FALSE. These two operations were demonstrated in [4] to be easily implemented with RRAM devices and a circuit such as the one reported in **Figure 4a**, which consists of RRAM devices, that store the inputs and outputs of IMPLY and FALSE operations and act as computing elements, a control logic and analog tri-state drivers that deliver appropriate voltages to RRAM devices, and a resistor R_G. The IMPLY is a two inputs one output operation and its truth table is reported in **Figure 4b**. To execute an IMPLY operation, the inputs are encoded in the resistive state of devices P and Q. Then, the control logic delivers two voltage pulses with amplitudes V_{COND} and V_{SET}, on P and Q, respectively. The voltage pulse amplitudes are dimensioned so that the state of device P never changes, while the state of device Q changes according to the truth table in **Figure 4b**. The FALSE operation is a 1 input 1 output logic operation which always results in logic 0 output and is implemented by delivering a negative voltage V_{FALSE} to an RRAM device.

However, the use of high voltage pulses to conditionally program the output device in the IMPLY operation, leads to intrinsic reliability challenges that need to be studied with accurate compact models, including the effects of temperature and variability. Using the UNIMORE RRAM physics-based compact model, simulations of the IMPLY operation performed in [5] for different pairs of V_{SET} and V_{COND} voltages show that only a very narrow design space exists. As shown in **Figure 4c**, voltage variation of tens of mV, easily introduced by voltage drivers and line parasitic resistances, can lead to a malfunction of the circuit [5]. Also, the use of high voltage pulses induces the problem of logic state degradation [3, 5, 26] on devices that should preserve a HRS after the operation execution. These are devices P and Q in the first and third cases of the truth table, respectively. In these cases, although the voltage drop across these RRAM devices cannot fully switch a device, it can induce a drop of their resistance. Repeatedly executing IMPLY operations on these devices eventually leads to a bit

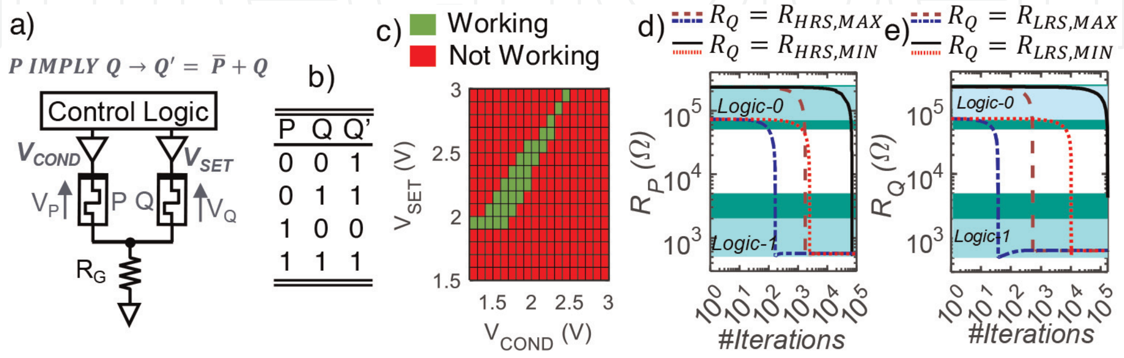


Figure 4.
(a) RRAM-based material implication logic core gate. (b) IMPLY operation truth table. Q' is the state of Q after the operation execution. (c) Map of the V_{SET} and V_{COND} pairs leading to a correct IMPLY gate operation. The effect of C2C variations is considered. Data from [5]. d), and e) effect of the resistive state degradation on device P and Q, due to the repeated execution of IMPLY operations on the same input device when both inputs are zero and when P = 1 and Q = 0, respectively.

corruption. This effect cannot be prevented completely, due to existence of opposite requirements on V_{COND} for the two cases of the truth table. The number of cycles before a corruption depends on the V_{SET} and V_{COND} pair, but also on the initial device resistance, as shown in **Figure 4d** and **e**.

Thus, although promising, the analysis performed with the compact model highlighted that this stateful LIM framework is intrinsically unreliable and other approaches should be preferred.

3.2 The smart material implication logic

The SIMPLY LIM framework was proposed in [26] as a more reliable and efficient solution to implement the material implication logic on RRAM devices. The circuit used to implement an IMPLY operation is similar to the one used in the stateful approach with the addition of a comparator which is feedback to the control logic, see **Figure 5a**. Since, during an IMPLY operation, the state of the output device Q changes only when both P and Q are in HRS, in the SIMPLY framework the operation is split into a read step, which detects this input configuration, followed by a conditional programming step, as sketched in **Figure 5b**. During a read step small (i.e., ≈ 100 mV) voltage pulses are delivered to the input RRAM devices. As a result, the voltage across R_G (i.e., V_N) when both inputs are in HRS is lower than in all the other cases. Thus, a sufficient read margin exists to discriminate the first case of the truth table from all the others by using a comparator with an appropriate threshold V_{TH}. In the conditional programming step, the control logic delivers V_{SET} to Q only when the comparator detects the first case of the truth table, while it keeps the analog drivers in high impedance otherwise. Since only a sufficient read margin needs to be ensured, this approach is intrinsically more reliable than the stateful one. In fact, circuit simulations performed in [26, 28, 29] with the compact model, including the effect of variability and RTN showed that a sufficiently large read margin is easily obtained by tuning the read voltage and V_{FALSE} (i.e., higher HRS leads to larger memory windows). Also, simulations confirmed that by delivering only small read voltages to the RRAM devices, the effect of variability is virtually solved. In addition, V_{COND} is no more

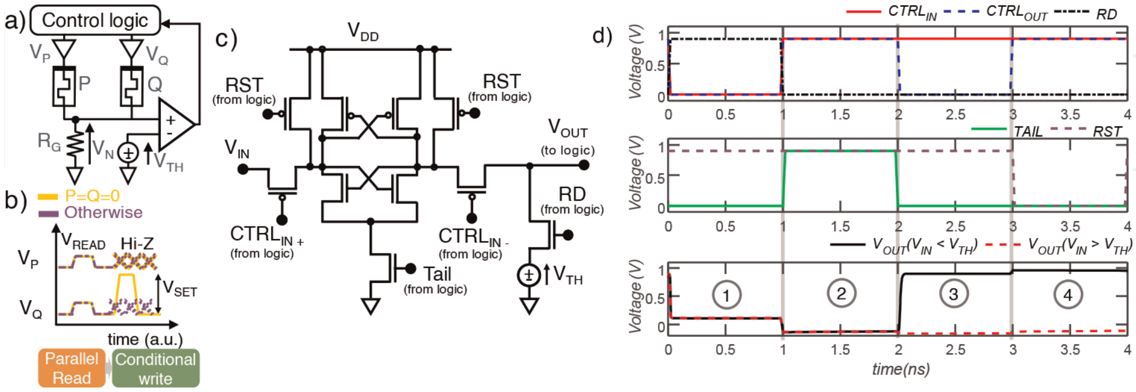


Figure 5.
a) SIMPLY core logic gate. b) Signals used to perform an IMPLY operation in the SIMPLY framework. When the input configuration P = Q = 0 *is detected,* V_{SET} *is delivered to* Q*. In all the other cases the voltage drivers are kept in high impedance (hi-Z). c) Sense amplifier circuit from [26] used as comparator. d) Control (i.e.,* $CTRL_{IN}$, $CTRL_{OUT}$, RD, TAIL, *and* RST*) and output signals (i.e.,* V_{OUT}*) of the sense amplifier circuit in c). A complete comparison is completed in four steps: 1) the internal nodes of the VSA are pre-charged with the input and threshold voltage (i.e.,* V_{TH}*); 2) the comparison is performed by turning on the* TAIL *transistor; 3) the result of the comparison is passed to the output node; 4) the control logic delivers the conditional* V_{SET} *pulse and resets the internal nodes of the VSA.*

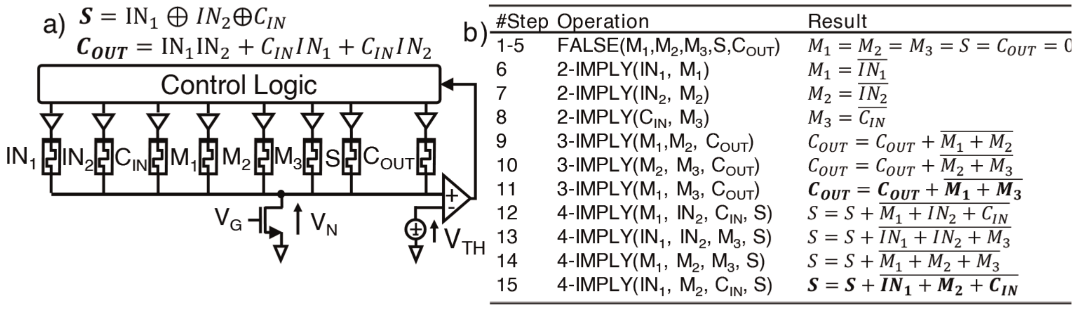
required, and since V_{SET} is applied only when P and Q are zero, in three out of four cases of the truth table the energy consumption is greatly reduced with respect to the stateful implementation, as discussed in [28]. Additional energy can be saved using the same approach on the FALSE operation, thus splitting the operation into a read step followed by a conditional V_{FALSE} pulse. When the device is already in the HRS, delivering V_{FALSE} to the device would waste additional energy.

Also, the multi-input IMPLY operation was proposed in [30] to speed up computations, and its feasibility up to four inputs operations was demonstrated by means of circuit simulations on SIMPLY-based architectures in [31]. In the SIMPLY framework the multi-input IMPLY operation (hereafter called n-IMPLY, where n indicates the number of inputs) is executed in the same way as the two inputs IMPLY operation, with the only difference that all the input devices are read in parallel. V_{SET} is then delivered to the output device only when all the inputs are in HRS. The logic operation is equivalent to $Q' = Q + \overline{P + S + \ \ldots}$, where the output is written on Q while P and S are other input devices. Although the read margin at the input of the comparator decreases for increasing number of inputs, the same voltage threshold V_{TH} can be used. The SIMPLY architecture enabling the execution of n-IMPLY operations is referred to as n-SIMPLY in this chapter.

To correctly evaluate the performance of SIMPLY-based architectures it is important to also evaluate the performance of the comparator. The latter was implemented as a voltage sense amplifier (VSA) designed in [26] with a 45 nm technology from [32]. The circuit of the VSA and the timing of the control signals are reported in **Figure 5c** and **d**. The proposed design consumes less than 8 fJ per comparison when V_{DD} is 2 V and the temperature is in the range 0 to 85°C.

3.2.1 Full adder implementation on a n-SIMPLY-based architecture

To implement more complex logic functions, sequences of IMPLY and FALSE operations and a sufficient number of devices are required. To highlight the remarkable energy efficiency of the n-SIMPLY architecture here we consider the implementation of a full adder (FA). As shown in **Figure 6a**, to implement a 1-bit FA an array with at least 8 RRAM devices is needed. These devices are used to store the inputs (i.e., IN_1, IN_2, and C_{IN}), the outputs (i.e., S, C_{OUT}), and partial results (i.e., M_1, M_2, and M_3) of the operation. To compute the result of the 1-bit addition, the 15 computing steps of n-IMPLY and FALSE operations reported in **Figure 6b** need to be executed sequentially [31]. The circuit in **Figure 6a** was simulated in [31] using the compact model and considering a clock frequency of 500 MHz, to estimate its performance. As reported in [31] the computation of a 1-bit FA consumes a worst-case


a)
$\boldsymbol{S} = \mathrm{IN}_1 \oplus IN_2 \oplus C_{IN}$
$\boldsymbol{C_{OUT}} = \mathrm{IN}_1\mathrm{IN}_2 + C_{IN}IN_1 + C_{IN}IN_2$
Control Logic
IN_1
IN_2
C_{IN}
M_1
M_2
M_3
S
C_{OUT}
+
-
V_G
V_N
V_{TH}
b)

#Step	Operation	Result
1-5	FALSE(M_1,M_2,M_3,S,C_{OUT})	$M_1 = M_2 = M_3 = S = C_{OUT} = 0$
6	2-IMPLY(IN_1, M_1)	$M_1 = \overline{IN_1}$
7	2-IMPLY(IN_2, M_2)	$M_2 = \overline{IN_2}$
8	2-IMPLY(C_{IN}, M_3)	$M_3 = \overline{C_{IN}}$
9	3-IMPLY(M_1,M_2, C_{OUT})	$C_{OUT} = C_{OUT} + \overline{M_1 + M_2}$
10	3-IMPLY(M_2, M_3, C_{OUT})	$C_{OUT} = C_{OUT} + \overline{M_2 + M_3}$
11	3-IMPLY(M_1, M_3, C_{OUT})	$\boldsymbol{C_{OUT} = C_{OUT} + \overline{M_1 + M_3}}$
12	4-IMPLY(M_1, IN_2, C_{IN}, S)	$S = S + \overline{M_1 + IN_2 + C_{IN}}$
13	4-IMPLY(IN_1, IN_2, M_3, S)	$S = S + \overline{IN_1 + IN_2 + M_3}$
14	4-IMPLY(M_1, M_2, M_3, S)	$S = S + \overline{M_1 + M_2 + M_3}$
15	4-IMPLY(IN_1, M_2, C_{IN}, S)	$\boldsymbol{S = S + \overline{IN_1 + M_2 + C_{IN}}}$

Figure 6.
a) Implementation of a 1-bit FA on the n-SIMPLY architecture. b) Sequence of computing steps from [31] required to compute the output for a 1-bit FA when exploiting the n-SIMPLY framework.

energy of 4.2 pJ and it is computed in 60 ns. Although these numbers are orders of magnitude higher than those achievable with full CMOS implementations, n-SIMPLY-, and LIM-based architectures in general, provide considerable advantages when data intensive tasks are considered. Thus, in [31] the parallel execution on conventional CMOS circuits and on the n-SIMPLY architecture of 512 32-bit FA operations was compared. When including the VNB overhead, the n-SIMPLY architecture was demonstrated to achieve a $> 10^6$ energy delay product (EDP) improvement with respect to the CMOS implementation.

3.3 Comparison with other RRAM-based logic circuits

Although in this chapter material implication-based LIM frameworks are analyzed as a use case example, other approaches have been studied in the literature. Here follows a description and an analysis of the main characteristics and performance of the most common LIM frameworks in which RRAM devices are used both as computing and as memory elements, and hence can enable the implementation of non-von Neumann computing architectures.

3.3.1 Memristor ratioed logic

Although the Memristor Ratioed Logic (MRL) [33, 34] enables to fabricate logic gates with smaller footprints with respect to conventional CMOS logic circuits, it is not included in the analysis because in this framework RRAM devices are used only as computing elements, while inputs are encoded as voltages. Ali et al. demonstrated in [34] a possible realization on a crossbar array of a MRL-based FA design, however in this approach the computation parallelism is limited, and retrieving the input data would still incur in the VNB overhead.

3.3.2 Memristor-aided logic

The MAGIC LIM framework is similar to the stateful RRAM-based material implication logic one described in Section 3.1. The core logic operations in the MAGIC framework are the NOR and the NOT operations that can be implemented with RRAM devices using the circuit shown in **Figure 7a**. Differently from the stateful material implication logic gate, the resistor R_G is replaced with an RRAM device (i.e., *O* in **Figure 7a**) which stores the result at the end of an operation execution. Thus, before each operation *O* is set into LRS. The inputs are stored on the devices P and Q (see **Figure 7a**), and by delivering a negative voltage pulse with amplitude V_0 to their TE, the resistive state of the device *O* changes depending on the resistive state of the inputs. As in the stateful material implication logic implementation, the conditional programming of an RRAM device reduces the circuit reliability, increasing the BER. As discussed in [35, 36], RRAM-based MAGIC implementations are affected by the small design space, and by logic state degradation, the latter affecting both the input and output devices. Also, as discussed in [35], the circuit reliability is strongly influenced by the RRAM technology characteristics. Indeed, larger $|V_{SET}/V_{RESET}|$and R_{OFF}/R_{ON} ratios can potentially improve the circuit reliability, provided that the effect of C2C variability is considered in the design phase. This further underlines the importance of accurate compact models for the implementation of device-circuit co-optimization strategies.

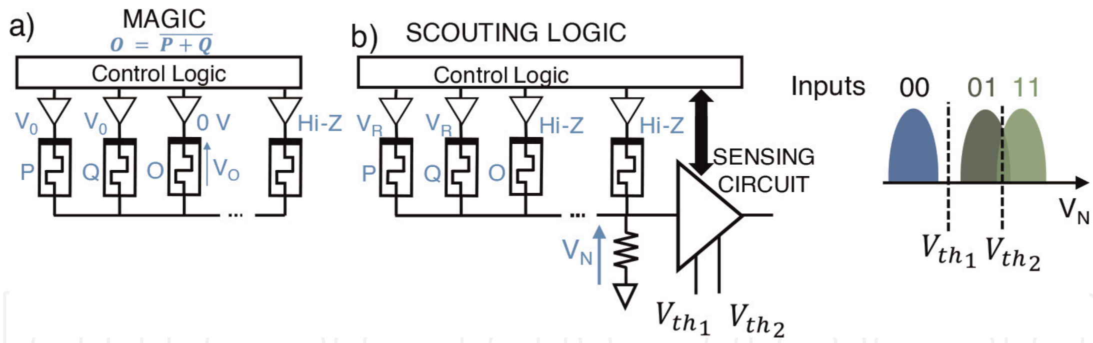


Figure 7.
a) NOR operation as implemented in the MAGIC LIM framework on a linear RRAM array. The resistive state of the output device O *changes depending on the input configuration. b) Sketch of the scouting logic LIM framework and associated reliability issues. Similarly to SIMPLY, two devices are read in parallel and the voltage (V_N) at the input of a sensing circuit changes depending on the input configuration. The sensing circuit tries to detect how many devices are in LRS, but as shown in the sketched distribution of* V_N *for different input combinations, while a sufficient read margin exists to detect the* 00 *combination, the* 01 *and the* 11 V_N *distributions are partially overlapped, thus leading to high BER.*

3.3.3 Scouting logic

The approach used in scouting logic is similar to the one employed in the SIMPLY framework. As shown in **Figure 7b**, also in this case RRAM devices encode the input bits of logic operations, and their state is read in parallel using current or voltage sense amplifiers. The equivalent parallel resistance of the input RRAM devices changes depending on the number of inputs in LRS, leading to different distributions at the input of the sense amplifier [25]. By using different thresholds, the sense amplifier can implement different logic operations. Specifically, by discriminating the 11 input combination from the others using the sense amplifier and V_{TH2} (see **Figure 7b**), the control logic in the array periphery can compute the AND and NAND logic operations. Conversely, the OR and the NOR operations can be implemented by discriminating the 00 input combination from the others using the V_{TH1} threshold (see **Figure 7b**). Also, in this framework the XOR and XNOR operations could be potentially implemented in only two read steps which discriminate the 01 input combination from the others. However, although as discussed in Section 3.2 for the SIMPLY framework, the 00 input combination can be reliably distinguished from the others, the distributions at the input of the sense amplifier for the 01 and 11 input combinations overlap due to the effects of variability and RTN, leading to very high bit error rates and a low circuit reliability when computing AND, NAND, XOR, or XNOR operations. Thus, in this framework only OR and NOR operations can be reliably computed [37]. Also, differently from SIMPLY, in which the output device is also an input of IMPLY operations, in scouting logic if more complex operations are implemented, partial results need to be stored on an additional device. Using the same device to accumulate multiple partial results (i.e., to OR together the partial results) may cause a set voltage pulse to be applied to a device already in LRS, thus causing high energy consumption.

3.3.4 Enhanced scouting logic

To improve the reliability of the scouting logic, Yu et al. introduced in [37] an enhanced version which is based on 2T1R memory arrays. As sketched in **Figure 8a** and **b**, the transistors in series with the RRAM devices can be used to either read the

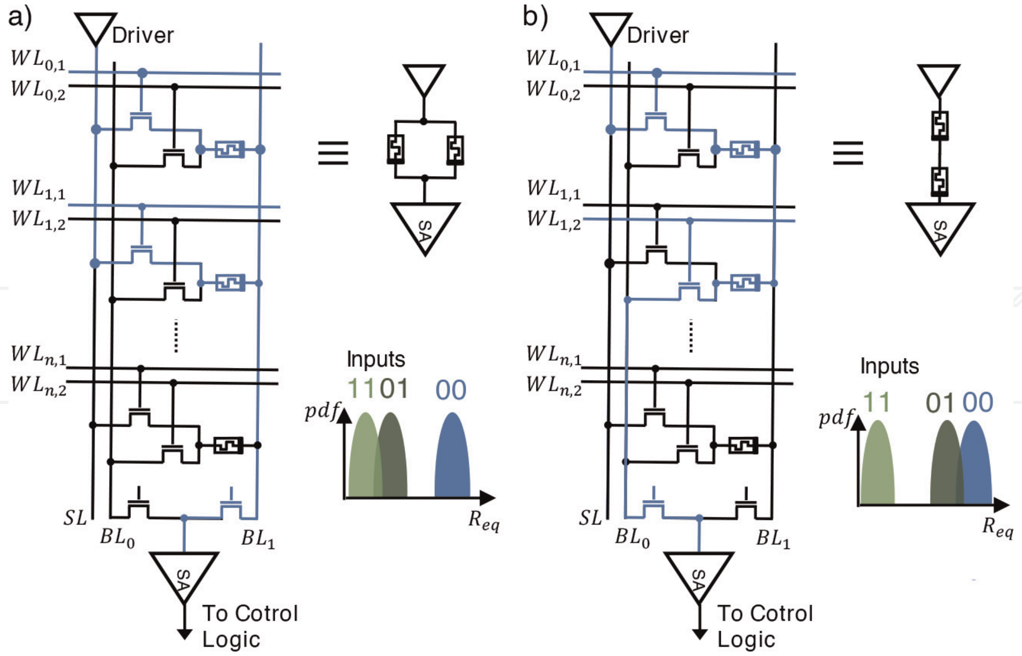


Figure 8.
Operation of the enhanced scouting logic implementation on 2T1R arrays from [37]. a) Execution of an OR/NOR operation. The WL and BL are controlled so that the parallel resistance between the input devices is connected to the sense amplifier. b) Execution of an AND/NAND operation. The WL and BL are controlled so that the series resistance between the input devices is connected to the sense amplifier.

equivalent parallel or series resistance between the two RRAM devices storing the input bits. As in the scouting logic, when reading the resistive state of the two devices in parallel the 00 input combination can be easily distinguished from the others to compute the OR or NOR operations. Instead, by reading the equivalent series resistance, the 11 input combination can be distinguished from the others thanks to a sufficiently large read margin, which enables the correct implementation of AND and NAND operations. Thus, by trading an increased chip area for reliability, in this framework both OR and AND operations can be reliably computed. Compared to the SIMPLY framework this approach can potentially reduce the number of computing steps required to compute more complex operations, however at the cost of a larger chip area that is required to accommodate the additional bit line and selector transistor, and a more complex sense amplifier circuit.

3.3.5 Performance comparison on a 1-bit FA implementation

The performance of LIM frameworks can be compared considering different metrics. As shown in **Table 1**, these include both metrics that can be directly associated with specific LIM frameworks (i.e., number of devices and of computing steps used to implement a specific logic function, and the feasibility in crossbar arrays), and other metrics that also depend on design choices and the specific technology employed to fabricate or to simulate the circuits, making the comparison between different LIM solutions non-trivial. To identify key differences between different LIM approaches the execution of a 1-bit FA operation is considered, since it represents a common simple benchmark for LIM frameworks.

Considering the stateful material implication-based implementations (i.e., “Stateful Mat. Imp.” in **Table 1**) different approaches can be followed to optimize the number of required devices or the number of computing steps. As reported in the

Author(s)	Type of LIM (Sim/ Exp)	Physics-Based Model	Number of Devices	Feasible in RRAM Array	Number of Elementary Steps	Delay	Energy	Circuit parameters**	Retains input values	Endurance before Refresh*
Talati et al. [38]	MAGIC (Sim)	NO	5–19	YES	13–15	≈ 17–19.5 ns (Reported)	≈3–6pJ (Reported)	V_0 = 1 V V_{SET} = 2 V V_{RESET} = -1 V f_{op} = 0.77GHz $\overline{R_{LRS}} = 1k\Omega$ $\overline{R_{HRS}}$ =300kΩ	NR (Tech. & Design Dependent)	NR
Yu [37]	Enhanced scouting logic	NR	9 RRAM	YES	17	68 ns (Estimated at 0.5 GHz)	NR	V_{SET} = NR V_{RESET} = NR V_{READ}=NR f_{op} = 0.33GHz $\overline{R_{LRS}} = 30k\Omega$ $\overline{R_{HRS}}$ =16.6 MΩ R_G = 160kΩ	YES	NR (Expected to be high)
Siemon et al. [39]	n-bits Stateful Mat. Imp. (Sim)	YES	8 RRAM	YES	19	3.61 µs (estimated)	202 pJ (estimated)	V_{SET} = 1.45 V V_{COND} = 1.24 V f_{op} = 1 MHz $\overline{R_{LRS}}$ =NR $\overline{R_{HRS}}$ =NR R_G = 15kΩ	NO	NR
Lehtonen et al. [27]	Stateful Mat. Imp. (Sim)	NO	8 RRAM	YES	136	NR	NR	NR	YES	NR
Kvatinsky et al. [3]	Stateful Mat. Imp. (Sim)	NO	9 RRAM	YES	23	9.1 µs (estimated)	NR	V_{SET} = 1 V V_{COND} = 0.5 V f_{op} = 2.5 MHz $\overline{R_{LRS}}$ =1kΩ $\overline{R_{HRS}} = 100k\Omega$ R_G = 10kΩ	NO	≈ 300 up to 10^5 (trades with energy)
Kvatinsky et al. [3]	Stateful Mat. Imp. (Sim)	NO	6 RRAM	YES	29	11.5 µs (estimated)	NR	V_{SET} = 1 V V_{COND} = 0.5 V f_{op} = 2.5 MHz $\overline{R_{LRS}}$ =1kΩ	NO	≈ 300 up to 10^5 (trades with energy)

Author(s)	Type of LIM (Sim/ Exp)	Physics-Based Model	Number of Devices	Feasible in RRAM Array	Number of Elementary Steps	Delay	Energy	Circuit parameters**	Retains input values	Endurance before Refresh*
								$\overline{R_{HRS}} = 100\text{k}\Omega$ R_G = 10kΩ		
Cheng et al. [40]	Stateful Mat. Imp. (Exp)	NA	8 RRAM	YES	27	54 μs (reported)	19.5 pJ (reported)	V_{SET} = -2 V V_{COND} = -1 V V_{RESET} = 1.5 V f_{op} = 200 kHz $\overline{R_{LRS}} = 3\text{k}\Omega$ $\overline{R_{HRS}} = 300\text{k}\Omega$ R_G = 20kΩ	NO	NR
Zanotti et al. [26]	Stateful Mat. Imp. (Sim)	YES	9 RRAM	YES	43	345 ns (Reported)	6.4 nJ (reported)	V_{SET} = 2.15 V V_{COND} = 1.7 V_{RESET} = -1.45 V f_{op} = 0.05GHz $\overline{R_{LRS}} = 2.8\text{k}\Omega$ $\overline{R_{HRS}} = 150\text{k}\Omega$ R_G = 1kΩ	YES	67
Zanotti et al. [26]	Stateful Mat. Imp. (Sim)	YES	8 RRAM	YES	28	560 ns (Reported)	518 pJ	V_{SET} = 2.15 V V_{COND} = 1.7 V_{RESET} = -1.45 V f_{op} = 0.05GHz $\overline{R_{LRS}} = 2.8\text{k}\Omega$ $\overline{R_{HRS}} = 150\text{k}\Omega$ R_G = 1kΩ	YES	≈ 30
Zanotti et al. [26]	SIMPLY (Sim)	YES	8 RRAM	YES	28	920 ns (Reported)	172 pJ	V_{SET} = 2.15 V V_{RESET} = -1.45 V V_{READ} = 0.2 V f_{op} = 25 MHz $\overline{R_{LRS}} = 2.8\text{k}\Omega$ $\overline{R_{HRS}} = 150\text{k}\Omega$ R_G = 1kΩ	YES	> $4.5 \cdot 10^6$ (no energy trade-off)

Author(s)	Type of LIM (Sim/ Exp)	Physics-Based Model	Number of Devices	Feasible in RRAM Array	Number of Elementary Steps	Delay	Energy	Circuit parameters**	Retains input values	Endurance before Refresh*
Zanotti et al. [31]	n-SIMPLY (Sim)	YES	8 RRAM	YES	15	60 ns (Reported)	4.2 pJ	V_{SET} = 3 V V_{RESET} = -2.8 V V_{READ} = 0.2 V f_{op} = 0.25GHz $\overline{R_{LRS}} = 10.5k\Omega$ $\overline{R_{HRS}} = 550k\Omega$ R_G = 10kΩ	YES	> $4.5{\bullet}10^6$ (no energy trade-off)
Junsangsri et al. [41]	CMOS LIM (Sim)	YES (FET) NO (RRAM)	41 FET + 4 RRAM	NO	NA	52 ps (Reported-excludes RRAM delay)	2.2 fJ (reported-excludes RRAM energy)	V_{DD} = 3.3 V $\overline{R_{LRS}}$ =30kΩ $\overline{R_{HRS}}$ =100 MΩ	YES	NR (Limited by FET Reliability)

This estimate can be trusted only when a physics-based model was used. Circuit parameters reported by the authors in their manuscripts. f_{op} is the frequency of single operation execution (e.g., 1/ t_{IMPLY}). NA stands for "not applicable." NR stands for not reported.*
In the table, reported indicates that the values were explicitly reported in the paper. Estimated indicates that the corresponding value was extrapolated from data in the paper since it was not explicitly reported.

Table 1.
Comparison of 1-bit FA implementations based on different LIM frameworks.

literature [3, 26, 27, 40], typically a 1-bit FA operation can be implemented using from 6 up to 9 RRAM devices, and sequences of IMPLY and FALSE operations which length vary from as low as 23 steps, when the inputs are overwritten, up to 136 steps when the sequence is not optimized. To further reduce the number of computing steps, the multi-input stateful material implication framework was proposed by Siemon et al. in [39], showing that by using three-inputs IMPLY operations the number of computing steps can be reduced down to 19 steps when using 8 RRAM devices. The n-SIMPLY framework discussed previously in Section 3.2, enables to reliably compute 4-inputs IMPLY operations, and thus it reduces the number of computing steps down to 15. In terms of number of computing steps, the different set of core operations used in the enhanced scouting logic and the MAGIC LIM frameworks provides additional advantages compared to most of the material implication-based solutions. Although, not explicitly reported in the literature, by combining AND, NAND, NOR, and OR operations in the enhanced scouting logic array implementation (see **Figure 8**), the 1-bit FA operation is executed in 15 steps, which could be further reduced as in the n-SIMPLY approach by increasing the parallelism of the OR/NOR operations (the parallelism of the AND/NAND operations cannot be increased in the architecture reported in [37]). MAGIC-based implementations in [38] can achieve the lowest number of computing steps (i.e., 13 steps) among the different LIM implementations reported in **Table 1**.

Although the number of computing steps is linked to the computing delay (i.e., $delay = \#steps \ \ t_{step}$) and can give an idea about the energy efficiency of the specific implementation (i.e., $E_{tot} = \sum_{i=1}^{\#steps} E_i$, where E_i is the energy dissipated in each step), the achievable time (t_{step}) and energy (E_i) per single computing step depend on the characteristics of the RRAM technology that is used on the adopted design choices. Also, the accuracy of the performance and reliability analysis depends on the type of compact model used in the circuit simulations. Indeed, compared to general purpose memristor models, the use of physics-based compact models can provide more accurate results when fast voltage pulses are used and enable to analyze important nonideal effects such as the logic state degradation. Thus, in **Table 1**, along with the energy and computing delay of different implementations from the literature, also the main circuit parameters and an indication regarding the type of compact model used in the simulations are reported. In general, to reduce the energy per computing step, increasing the LRS resistances can be a viable solution. However, larger voltages must be employed to program a device without increasing the programming pulse width, potentially conflicting with the desired circuit design space. In terms of energy efficiency, currently a hybrid FET-RRAM design from [41] shows the best performance. However, this implementation is not feasible in crossbar arrays, limiting the area efficiency, and simulations were performed with an extremely simplified RRAM model, suggesting the need for a more accurate performance analysis. Among the available data regarding crossbar-feasible implementations, MAGIC and n-SIMPLY achieve the best energy and delay performance. However, although MAGIC is slightly more efficient than n-SIMPLY, the 1-bit FA implementation does not retain the input states and is affected by the problem of logic state degradation, which can discourage its adoption in applications such as hardware accelerator for BNNs. In BNN the network parameters, stored in the nonvolatile resistance of RRAM devices, must be preserved through computations. Thus, the effect of logic state degradation introduces the need for frequent memory refresh cycles, causing high inefficiencies. Therefore, in the following sections, the design of a LIM-based accelerator for BNNs is discussed considering an n-SIMPLY implementation.

4. RRAM-based binarized neural networks accelerators

In BNNs [9] weights and activations of neural networks are represented with a single bit. Despite the considerable reduction in the required hardware resources, for some classification tasks, BNNs were demonstrated to achieve similar performance with respect to full precision neural network implementations, with only slight accuracy degradation [9]. The use of binary weights and activations, not only leads to a reduction of the size of the memory used to store the network parameters, but also reduces the complexity of the operations performed by each neuron of the network. As shown in **Figure 9a**, in full precision neural networks each neuron computes the dot product between its input activations (i.e., X) and the learned weights (i.e., W), adds a bias term (i.e., b) to the result, and finally applies a nonlinear activation function (i.e., *ReLu* in **Figure 9a**) to compute the output activation. In BNNs, these operations translate to simpler logic operations. Specifically, the dot product becomes a bitwise XNOR followed by an accumulation, or popcounting, of the results, and the activation function becomes the comparison with a learned threshold. Despite using less memory space to store the network parameters, BNNs are still characterized by a large number of parameters. Running inference tasks on conventional hardware would incur in the VNB, therefore degrading the performance. Thus, several IMC hardware accelerators based on RRAM technologies have been proposed in the literature [7, 8, 42], showing promising efficiency improvement without considerable accuracy degradation. In fact, binary storage is currently a more robust approach for state-of-the-arts RRAM technologies with respect to multibit storage on a single which is more affected by device nonidealities.

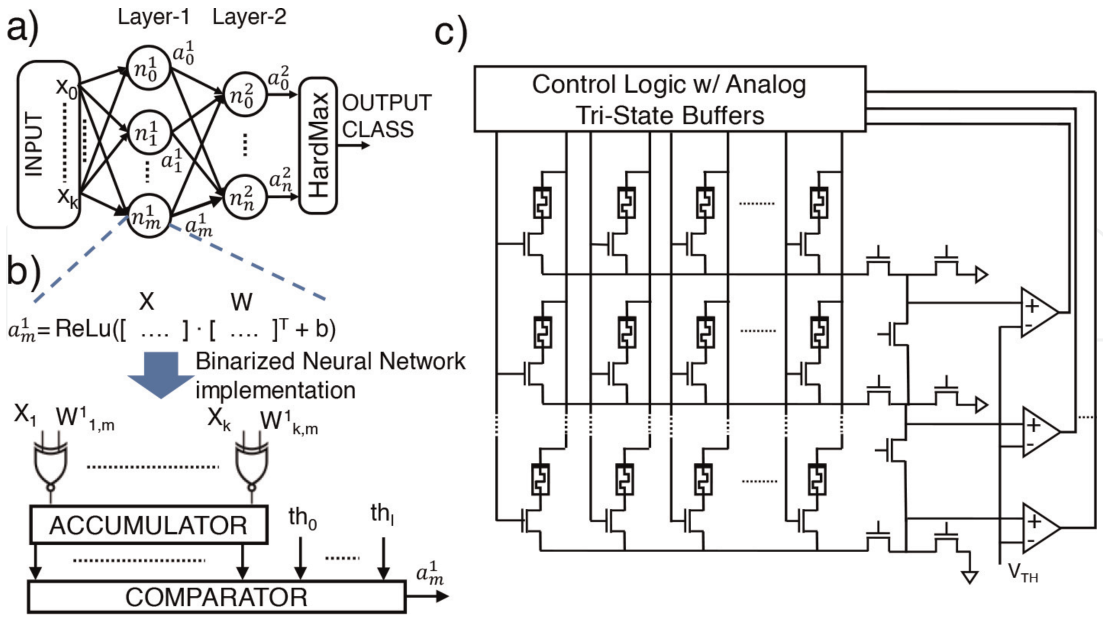


Figure 9.
(a) Sketch of a fully connected neural network, with k *inputs,* m *neurons in the hidden layer, and* n *neurons in the output layer. The operation performed by each neuron is shown. (b) Implementation of a neuron in BNNs. The vector-vector multiplication becomes the accumulation of the results from bitwise XNOR operations, and the activation function the comparison with a threshold. (c) 2D array of 1T1R devices that can be used to accelerate in hardware the operations required for classification tasks based on BNNs, in the SIMPLY framework. FETs control signals are managed by the control logic.*

In the following sections, three different BNN hardware accelerators based on 1T1R memory arrays are discussed and benchmarked. Specifically, Section 4.1 describes SIMPLY-based implementations, Section 4.2 discusses analog accelerators for VMM used in BNNs, and Section 4.3 describes a hybrid accelerator combining the previous two approaches on the same array.

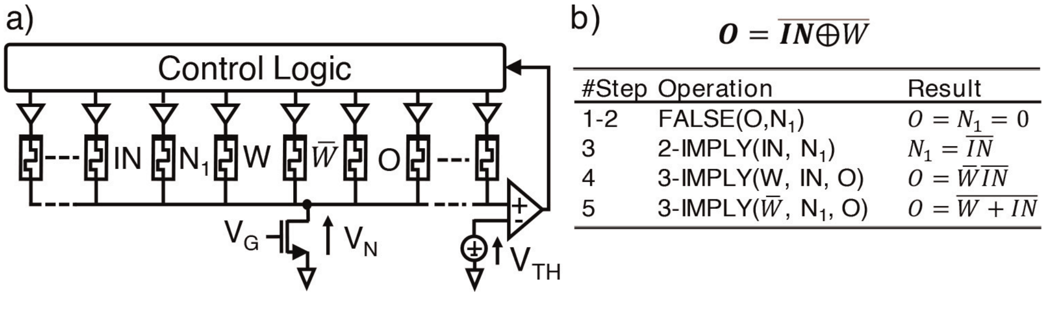
4.1 SIMPLY-based BNN accelerators

Since the core operations of BNNs are logic operations, SIMPLY-based architectures can be used to realize energy efficient hardware accelerators for inference tasks. Such accelerators can be realized on memory arrays and a peripheral circuit such as the one shown in **Figure 9c**, enabling the implementation of single instruction multiple data architectures, which can exploit the intrinsic parallelism of BNN inference algorithms [28, 31, 43]. In fact, depending on the number of comparators placed in the array periphery, and the design of the drivers and control logic, multiple devices in different rows and in the same column can be read and programmed in parallel. For instance, to perform an IMPLY operation in the SIMPLY framework on multiple rows of the array, the columns of interest are biased with the read voltage and the corresponding select lines are biased to turn on the selector FET. At the same time, FET devices in the array periphery are biased to implement R_G and bring the voltage at the row of interest to the input of a VSA. Then the column corresponding to the output device is biased with V_{SET} and FET devices in the array periphery provide a low conductive path for the devices that need to be programmed.

In the rest of this section, the implementation of the different core operations of BNNs on the SIMPLY architecture is discussed.

4.1.1 XNOR operation

In BNNs, a dot product operation is equivalent to a bitwise XNOR operation between the input activations and the weights of the neuron, followed by an accumulation of positive results. As shown in **Figure 10**, for BNN applications, each XNOR operation can be implemented with 5 RRAM devices and 5 computing steps [31], provided that both the weight (i.e., W) and its complement (i.e., $\overline{\boldsymbol{W}}$) are stored in the memory array and never overwritten. The other three devices store the input (i.e., *IN*), the output (i.e., *O*), and the partial result (i.e., N_1) of the XNOR. Since operations



$$\boldsymbol{O} = \overline{\boldsymbol{IN}\oplus W}$$

#Step	Operation	Result
1-2	FALSE(O,N_1)	$O = N_1 = 0$
3	2-IMPLY(IN, N_1)	$N_1 = \overline{IN}$
4	3-IMPLY(W, IN, O)	$O = \overline{W}\,\overline{IN}$
5	3-IMPLY($\overline{W}$, N_1, O)	$O = \overline{W + IN}$

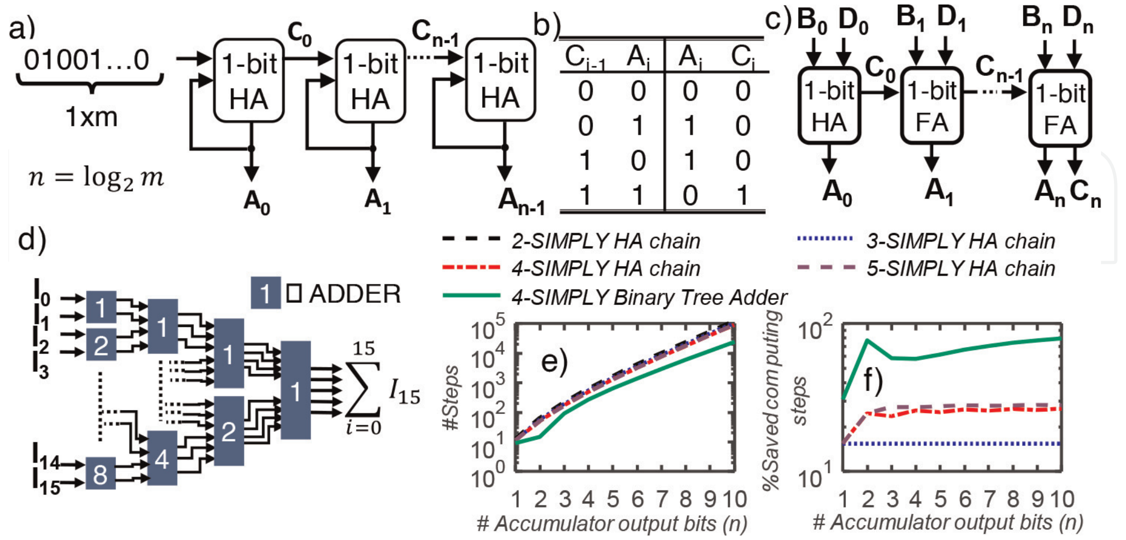
Figure 10.
(a) XNOR gate implementation used for BNN inference tasks in the SIMPLY framework. To reduce the number of computing steps, the weight, and its complement (i.e., W, and $\overline{W}$, respectively) are stored in the array and preserved through the computations. (b) Sequence of computing steps in the n-SIMPLY framework.

on the same row of an array are sequentially executed, N_1 can be reused in the subsequent XNOR operation when computing a dot product.

4.1.2 Accumulation operation

To implement the accumulation operation, different approaches corresponding to different sequences of IMPLY and FALSE operations can be followed. As proposed in [31, 43], the accumulation operation can be implemented with a chain of half adders (HAs), see **Figure 11a** and **b**. Each HA receives in input its output from the previous step and a new input bit when it computes the least significant bit (LSB), or the output carry from the previous HA in the chain otherwise. In the SIMPLY implementation, when a new bit is accumulated, the result of each HA in the chain is computed sequentially, starting from the LSB. However, based on the position of an HA in the chain (i) its result is computed only after 2^i input bits have been accumulated, since when fewer bits have been accumulated the corresponding HA output would be zero. Although the number of HAs grows as the *log*$_2$*()* of the number of input bits, the number of computing steps grows exponentially. As shown in **Figure 11e** and **f**, exploiting different maximum parallelism in the framework of n-SIMPLY to optimize the sequence of computing steps results in a > 15% reduction in the number of computing steps with respect to the 2-SIMPLY implementation [31], however, to further improve the efficiency different approaches can be followed.

Here we propose a SIMPLY-based implementation of a binary tree adder architecture as the one shown in **Figure 11d**. In each level of the tree, the results from the previous level are added two by two, until all the input bits are accumulated. Thus, the number of levels of the tree corresponds to the *log*$_2$*()* of the number of input bits. From one level of the tree to the following, the size of the FA increases by 1 bit. Each adder is implemented as a ripple-carry adder, see **Figure 11c**, with a 1-bit HA requiring 9 computing steps, and a 1-bit FA requiring 15 steps (see **Figure 6b**). This



C_{i-1}	A_i	A_i	C_i
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Figure 11.
(a) Implementation of the accumulation operation on m *input bits with a chain of* n *HAs, as in [31]. (b) Truth table of the 1-bit HA used in the HA chain. (c) n-bit ripple-carry adder implementation used as the core element of the binary tree adder-based accumulator. (d) Example of a binary tree adder used to accumulate 16 input bits. The number of levels in the binary tree grows as the* $\log_2(m)$*. (e) Comparison of the number of steps required to accumulate* $(2^n\text{-}1)$ *bits for different accumulator implementations. (f) Percentage of saved computing steps with respect to the* 2-SIMPLY HA chain *implementation of the different accumulator implementations.*

approach provides a considerable performance improvement compared to the HA chain implementation as shown in **Figure 11e** and **f**.

4.1.3 Comparison operation

In the original BNN implementation form Courbariaux et al. [9], weights and activations are encoded with +1 and − 1 values, meaning that the result of the accumulation operation spans from negative to positive values. Thus, the activation operation is commonly implemented using the *sign()* function. Conversely, in the SIMPLY-based implementation, weights and activations are encoded with 0 and 1, and the result of the accumulation is a positive value, ranging from zero to the number of products performed in the neuron. Thus, the activation function in SIMPLY-based implementations corresponds to the comparison with a threshold (i.e., half the number of products performed in the corresponding neuron). The comparator outputs a logic 1 only when the result of the accumulation is above the threshold. The logic function implementing this operation in the sum of products form is reported in **Figure 12a**. As discussed in [31], increasing the parallelism in the read step of n-IMPLY operation, considerably reduces the required number of computing steps required for a comparison. In fact, when using only 2-IMPLY operations the number of computing step grows exponentially with the number of inputs, while using n-IMPLY operations with n > 2 leads to linear trends, as shown in **Figure 12b**. More details regarding the different sequence of computing steps implementing the comparison operation are available in [31].

4.1.4 Hard max operation

To determine the output class of an inference task the hard max function can be used. This function determines the index of the maximum value among a group of elements. A sketch of the approach used for the SIMPLY-based implementation is shown in **Figure 12c**. Similarly to the binary tree adder, also in this case a binary tree structure is used, and at each level of the tree pairs of elements are compared. To do so, for each pair of elements, the one stored in the upper position of the array is copied in the same row of the other element together with its ID. Then the two elements are

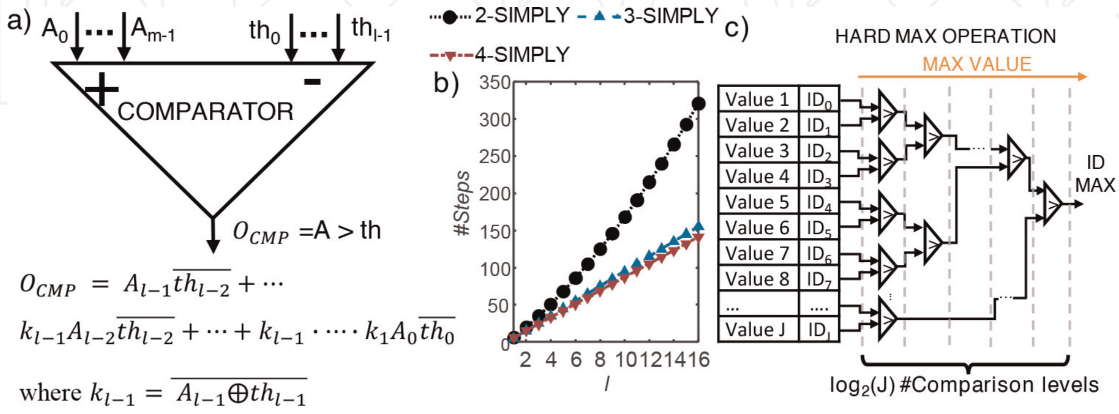


Figure 12.
(a) Comparator sketch and corresponding logic function, which outputs a logic 1 when a > th, *and a logic 0 otherwise. (b) Number of computing step required for different n-SIMPLY-based implementations and increasing number of compared bits (*l*). In the legend,* n *indicates the maximum degree of parallelism used in the parallel read step of IMPLY operations. (c) Sketch of the hard max activation function implemented in the n-SIMPLY framework. Inputs are compared pairwise using the comparator implementation in (a) and (b).*

compared using the same comparator implementation used for the neuron activation function. For each pair, the initial value and ID stored in the row are replaced with the larger element resulting from the comparison. In just *log*$_2$*(number of elements)* levels of the tree, the result is computed and stored in place of the last element. Additional information regarding the SIMPLY-based implementation is available in [11, 31].

4.1.5 Batch normalization operation/bias term addition

Batch normalization layers are often used in neural networks to speed up the training of the network parameters and consist in the scaling of the inputs of a layer using learned parameters. In BNNs, instead, batch normalization is introduced between the MAC operation and the *sign()* activation function, as discussed by Courbariaux et al. [9]. Thus, during training, the average ($\overline{m}$) and the standard deviation (σ) of the results of the MAC operations of a layer are computed, while a scaling (γ) and a bias terms (β) are learned and optimized through each iteration. In forward passes of the neural network these parameters are used to regularize the results of the MAC operations (a), so that the input to the *sign()* function is $\hat{a} = (a - \overline{m}) \bigodot \gamma \bigodot \frac{1}{\sigma} + \beta$.

As discussed in Section 4.1.3, in the SIMPLY-based implementation the activation function is implemented as the comparison with a threshold (th), that is equivalent to half the number of inputs of a layer if no bias term is added to the result of MAC operations. Thus, the effect of the batch normalization parameters can be introduced in the SIMPLY-based neural network by adjusting the value that is mapped to the threshold of the activation function for each neuron ($t\hat{h} = \frac{th-\beta}{\gamma}\sigma + m$).

In the output layer of a neural network, the hard max activation function is used to infer the output of the network. Thus, batch normalization or equivalently the addition of a bias term, needs to be performed also on the SIMPLY-based architectures. These operations consist in the addition of a fixed value to the results of the MAC operation, and can be implemented on the SIMPLY-based architecture by adding the bias term, that is stored on the memory array, using the ripple-carry adder described in Section 3.2.1 and shown in **Figure 11c**.

4.1.6 Array level implementation

As discussed in the previous sections, the SIMPLY-based implementation of BNN inference accelerators requires mapping to the resistive state of RRAM devices of memory arrays, the inputs, the neurons weights, the thresholds of the activation functions, the IDs of the output layer, and devices for storing partial results of computations. Ideally, infinitely large arrays would simplify this task by storing all the parameters of a single layer of the neural network onto a single array and the parameters of a single neuron all on a single row of that array. However, real memory arrays have a limited size due to nonideal effects, such as line parasitic resistance [44]. Thus, in practical applications, the parameters of a neural network are split onto multiple arrays, and the parameters of a single neuron onto multiple adjacent rows of the same sub-array, as shown in **Figure 13a**. When the parameters related to a single neuron are split onto multiple rows of the array, as in **Figure 13b**, each row of the array must still contain an equal number of devices for storing the inputs, the weights, their complement, and the outputs, for computing the partial results of the accumulation operations. Also, sufficient space on the array must be left to store support devices and other network parameters (e.g., the thresholds of the activation functions). After partial results are

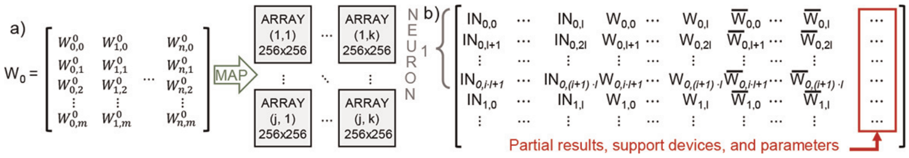

a)
$W_0 = \begin{bmatrix} W^0_{0,0} & W^0_{1,0} & & W^0_{n,0} \\ W^0_{0,1} & W^0_{1,1} & & W^0_{n,1} \\ W^0_{0,2} & W^0_{1,2} & \cdots & W^0_{n,2} \\ \vdots & \vdots & & \vdots \\ W^0_{0,m} & W^0_{1,m} & & W^0_{n,m} \end{bmatrix}$
MAP
ARRAY (1,1) 256x256
ARRAY (1,k) 256x256
ARRAY (j, 1) 256x256
ARRAY (j, k) 256x256
NEURON 1
b)
$IN_{0,0}$ … $IN_{0,l}$ $W_{0,0}$ … $W_{0,l}$ $\overline{W}_{0,0}$ … $\overline{W}_{0,l}$
$IN_{0,l+1}$ … $IN_{0,2l}$ $W_{0,l+1}$ … $W_{0,2l}$ $\overline{W}_{0,l+1}$ … $\overline{W}_{0,2l}$
$IN_{0,i\cdot l+1}$ … $IN_{0,(i+1)\cdot l}$ $W_{0,i\cdot l+1}$ … $W_{0,(i+1)\cdot l}$ $\overline{W}_{0,i\cdot l+1}$ … $\overline{W}_{0,(i+1)\cdot l}$
$IN_{1,0}$ … $IN_{1,l}$ $W_{1,0}$ … $W_{1,l}$ $\overline{W}_{1,0}$ … $\overline{W}_{1,l}$
Partial results, support devices, and parameters

Figure 13.
(a) Mapping of the neural network parameters on 256x256 *memory array. The number of arrays (i.e.,* j k*) depends on the number of inputs* (m) *and outputs* (n) *of each layer of the neural network. (b) Description of the mapping strategy used in a single array. Each row of the array stores portions of the input activations* (IN)*, the weights, and their complement, for a single neuron. Additional space to store partial results of the computations and additional parameters needs to be left and appropriately sized. The parameters and inputs of a single neuron may be split onto multiple rows of the array based on the number of available columns in the array.*

computed on each row, these must be copied to adjacent rows to compute the final result. Despite the increased complexity in the mapping of the neural network weights and the increased chip area occupied by the peripheral circuitry, the use of multiple arrays enables increasing the computation parallelism, reducing the computation latency. Results relative to different neurons stored on different sub-arrays can be computed in parallel. Also, the intrinsic parallelism of 2D array implementations of the SIMPLY architecture enables the parallel computation of partial results relative to a single neuron when its parameters are split onto multiple rows on the same array.

However, the number of devices on the same column of an array that can be programmed in parallel is constrained by the maximum current that can flow on a single array line, potentially decreasing the maximum parallelism that can be achieved. Indeed, this effect is connected to the current compliance used to program RRAM devices in the array, and the use of RRAM technologies that can reliably store binary values when programmed at low current compliance values would alleviate this constraint and enable the parallel programming of devices located onto multiple rows of the array.

4.2 BNN analog vector matrix multiplication accelerators

4.2.1 State of the art on RRAM-based analog BNN vector matrix multiplication accelerators

Thanks to their intrinsic reconfigurability, LIM computing architectures such as SIMPLY, are an effective solution for resource-constrained devices deployed at the edge of the network. However, the high level of reconfigurability comes at the cost of suboptimal performance when specific tasks need to be accelerated. For instance, considering BNNs inference tasks, most of the computations performed are required to compute the result of dot products [45]. Thus, designing optimized accelerators for the VMM operation would provide a considerable improvement of the overall accelerator performance. Thus, several analog/mixed-signal RRAM-based accelerators of the VMM used in BNNs have been proposed in the literature [6, 8, 42, 46, 47]. To encode the −1 and +1 weights of BNNs, a common solution adopted in analog/mixed-signal RRAM-based accelerators is to use a pair of devices (i.e., either 1T1R or 1R devices) programmed in complementary resistive states for each weight of the BNN. Driving these pairs of devices with appropriate voltage schemes enables to compute the result of XNOR operations using only positive RRAM conductances. However, different RRAM implementations vary depending on the array implementation, the sensing scheme (i.e., current sensing or voltage sensing), and peripheral circuitry. The main approaches that can be found in the literature are summarized in **Figure 14**. Specifically, **Figure 14a** shows a solution exploiting 1T1R arrays and a current sensing scheme to compute the bitwise XNOR and accumulation operations. In this approach, the weights of the BNN are encoded in pairs of devices located in adjacent rows of the same column of the array. The select lines of the selector transistors encode the input activations using complementary signals, see **Figure 14a**. In the array periphery, a transimpedance amplifier, implemented with an operational amplifier (OPA), ensures a virtual ground at the end of the column of the array, so that the current flowing in each column (i.e., I_{sum} in **Figure 14a**) is linearly proportional to the number of +1 results from the bitwise XNOR operations in that column. Also, the transimpedance amplifier converts the current to a voltage that is then digitized using an analog to digital converter (ADC). However, this solution is inefficient in terms of energy consumption and chip area, meaning that the transimpedance amplifier and the ADC must be shared among adjacent columns by means of analog multiplexers. A similar approach that is more efficient in terms of area occupation and energy efficiency was proposed by Yin et al. in [8], and the core elements of their solution are sketched in **Figure 14b**. The working principle is similar to the previous approach, with the main difference being that no OPA is used in the array periphery, thus saving considerable area and energy. However, without the OPA, no virtual ground is provided at the end

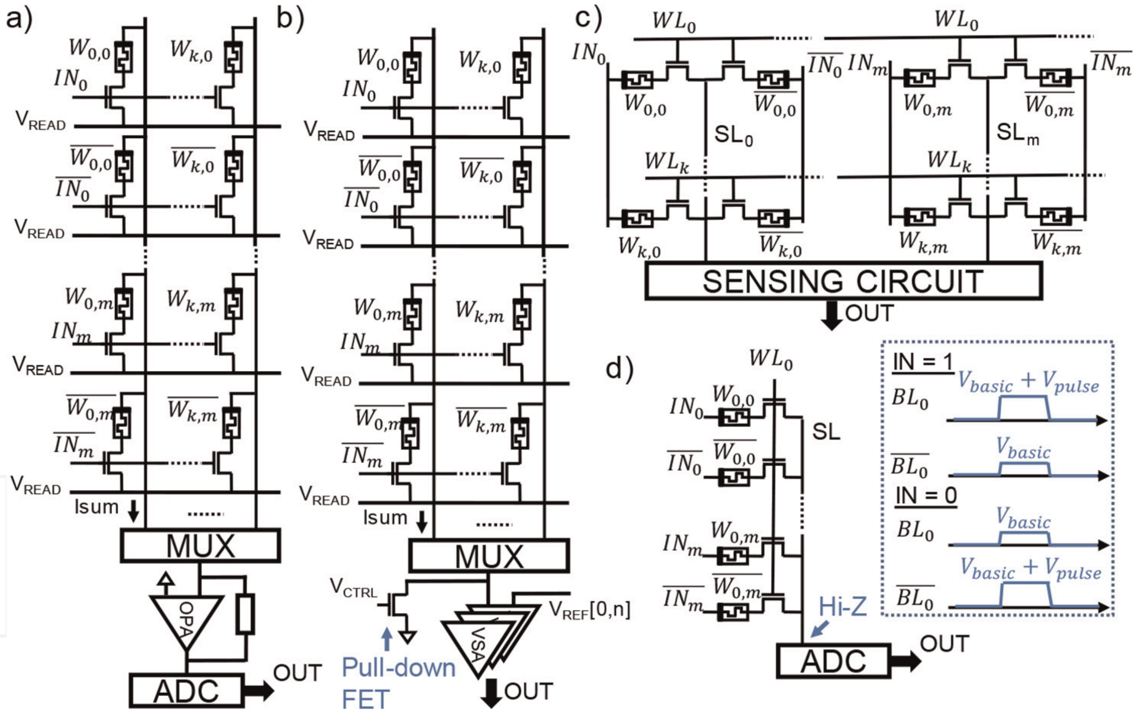


Figure 14.

Different implementations of the VMM operation with RRAM arrays. (a) 1T1R implementation in which each weight of the network is encoded as complementary resistive state of RRAM devices located in adjacent rows. Weights associated with the same neuron are stored in the same column of the array. An OPA provides a virtual ground node at the end of each column so that the current flowing in each column is equivalent to the sum of all the positive results of the bitwise XNOR operation. (b) Similar 1T1R implementation from [8], which uses a pull-down FET device and a flash ADC implemented with multiple VSAs instead of introducing a virtual ground node. (c) Sketch of the 2T2R implementation from [46]. Differently from a) and (b), the weights associated with the same neuron are located on the same row. A single vector-vector product is computed at a time by activating the desired word line (WL). Appropriate capacitive sensing circuits convert the results of the bitwise XNOR operations encoded in the select lines (SL) voltages to the output result. (d) 1T1R array implementation from [47], which exploits the select line capacitance to compute the result of the vector-vector multiplication. The SL is kept in hi-Z so that during a read-out step the voltage at the input of the ADC is proportional to the number of positive results of the bitwise XNOR operation.

of the columns of the array. The authors instead, introduced a pull-up or pull-down FET device, which creates a voltage divider with the equivalent resistance of the column which depends on the result of the bitwise XNOR operations. The voltage drop on the pull-down (pull-up) device increases (decreases) with the number of +1 from the bitwise XNOR operations in that column. Such voltage can be compared with a threshold using a VSA to compute the output activation. As discussed by Yin et al. in [8], multiple VSA and threshold voltages can be used to implement a flash ADC for improving the accuracy of the computation. The compact area and faster response of the VSAs compared to OPAs, not only reduce the area occupation but also increase the speed of the computation. In fact, the VSAs must be shared between fewer columns of the array compared with OPA-based solutions, thus increasing the throughput of the computation. A drawback of this approach is the nonlinear increment of the voltage at the input of the VSAs for an increasing number of positive results, which may require a fine tuning of the voltage thresholds to retain high accuracy in BNN inference tasks. Alternatively, approaches based on voltage mode sensing schemes have been recently proposed in the literature [46, 47], and shown to achieve high accuracy and energy efficiency. Specifically, Ezzadeen et al. proposed a solution based on 2T2R arrays which is sketched in **Figure 14c**. In this approach, the weights of each neuron are stored on the same row of the array, and the results for each neuron are computed one at a time. To do so, a single WL (see **Figure 14c**) is turned on at a time, and the input activations and their complements are applied to the corresponding pairs of devices which encode the neuron's weights. As a result, the voltage at each SL line (see **Figure 14c**) encodes the result of a single XNOR operation. In their paper, the authors proposed a capacitive sensing scheme to compute the result of the accumulation. Such a scheme requires a smaller chip area compared to other solutions, however at the cost of reduced computing speed since the result for a single neuron is computed at each step. Another voltage mode sensing scheme, sketched in **Figure 14d**, was proposed by Zhao et al. [47]. In this approach, the weights associated with the same neuron are encoded in adjacent RRAM devices programmed in complementary resistive states that are connected to the same SL (see **Figure 14d**). Input activations are encoded as a fixed bias (V_{bias}) term applied to both devices encoding a single neuron's weight, and an additional voltage contribution with amplitude *Vpulse* to the first or second line depending on the polarity of the input activation, as shown in **Figure 14d**. During a VMM, the WLs are activated, and the SLs are kept in Hi-Z. By exploiting the intrinsic capacitance of the array lines, the voltage on the SL lines is linearly proportional to the number of positive bitwise XNOR operations. Such voltage is then digitized by means of an ADC. Compared with the more common current mode sensing scheme, this approach is more energy efficient since smaller currents flow in the circuit.

Overall, each different scheme provides different tradeoffs, in terms of chip area (i.e., cost), computing speed, and energy. Thus, it is up to circuit designers to choose the design that better suits a specific application.

4.2.2 Circuit design tradeoffs: analysis of a case of study

In this section, we analyze in detail the tradeoffs existing in the BNN analog VMM accelerator shown in **Figure 14b** proposed by Yin et al. [8], since it can provide high-throughput combined with high energy efficiency, and a compact peripheral circuit design. An example of the core elements of a BNN VMM accelerator based on this approach is sketched in **Figure 15a**, where additional FET devices and the required control logic are also represented. As discussed in the previous section, the voltage at the input of the VSA increases nonlinearly with an increasing number of positive results of bitwise XNOR operations due to the existence of a voltage divider between the pull-down device and the equivalent resistance of the devices connected to the same column of the array. The reliability of this accelerator is linked to the linearity of the transfer characteristic and the read margin available at the input of the VSA. Indeed, a higher read margin over the whole span of possible inputs (i.e., the number of positive bitwise XNOR operations) would reduce the error rate of the VSA. Different design parameters can influence the linearity of the transfer characteristic and the available read margin at the input of the VSA. Specifically, the pull-down resistance (i.e., R_{PD}) changes the linearity of the transfer characteristic. As shown in **Figure 15b**, low R_{PD} values increase the linearity of the transfer, but too low values reduce the dynamic range, and thus the read margin, at the input of the VSA. Also, very low R_{PD} values may require excessively large FET devices which would reduce the area efficiency. Conversely, larger R_{PD} values increase the nonlinearity of the transfer characteristic, and too large values can quickly saturate the input of the VSA when just a few positive results are accumulated. Also, C2C variability can further reduce the read margin available at the input of the VSA, introducing possible overlaps between the distributions of the voltage at the input of the VSA for consecutive numbers of accumulated XNOR results, as shown **Figure 15c**. To increase the read margin higher power consumption can be traded by increasing the read voltage, provided that devices in HRS are not corrupted during a read step. All these circuit parameters must be optimized to ensure a reliable circuit operation. Still, a limited number of devices can be reliably read in parallel, introducing the need for the input split method described in [7]. In this approach, the computation of the complete sum of products is split into multiple steps, in which a partial number of the sum of products is computed, and the results of each step are accumulated using digital circuits. As suggested in [8], to improve the accuracy of the accelerator multiple VSA amplifiers and appropriately designed thresholds can be employed for each column, implementing a flash ADC converter. Specifically, in their work [8], the authors achieved high inference accuracy by using a 3-bit flash ADCs implemented with seven VSA, for computing operations with 64 inputs. Even when increasing the number of VSAs in the array

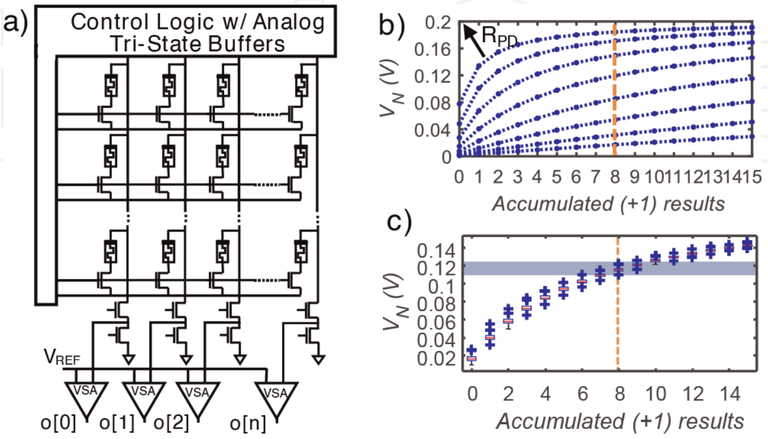


Figure 15.
(a) 1T1R memory array and peripheral circuits that can be used to accelerate the binary VMM operation in the analog domain, by storing the weights of the neural network and their complement in columns of the array, and controlling transistors select line depending on the input activations. As shown in (b), the current flowing in each column increases with the number of positive products. The transfer characteristic depends on the value of the pull-down resistance (R_{PD})*. (b) Effect of RRAM resistive state variability and RTN on the equivalent output voltage, showing possible overlaps between adjacent accumulated positive products.*

periphery, this approach can provide considerable efficiency improvement with respect to LIM-based accelerators since only read operations are performed without the need to use more energy-hungry programming steps.

4.3 Reconfigurable hybrid BNN SIMPLY/analog accelerator

To combine the reconfigurability of the SIMPLY architecture described in Section 4.1 and the efficiency of the BNN analog VMM accelerator discussed in Section 4.2.2, a hybrid architecture merging the two approaches on the same array was proposed in [28]. Here we discuss a variation of the architecture proposed in [28] which is less sensitive to sneak paths and requires fewer VSAs in the array periphery. Such architecture is shown in **Figure 16** and is similar to the SIMPLY-based architecture shown in **Figure 9c**. IMPLY and FALSE operations can be performed in the array as described in Section 4.1. Conversely, the architecture is slightly different to the one implementing the analog BNN VMM accelerator, shown in **Figure 15a**. Still the working principle is the same, with the only differences being that the currents encoding the results of multiplications flow in the rows of the array instead of the columns, that the columns of the array are biased with the read voltage, and that the pair of devices in complementary state encoding a single weight are encoded in adjacent columns instead of rows. To enable both computing paradigms the control logic and peripheral circuits need to be adapted, so that the reference voltage of the VSAs, and the bias voltage of the FET implementing either R_G or R_{PD} can be changed when performing computations in the SIMPLY or the analog BNN VMM frameworks, respectively. Also, if multiple VSA amplifiers are used to implement a flash ADC to improve the accuracy of the analog VMM operation, multiple rows of the array may share the same group of VSAs, thus reducing the maximum parallelism achievable, compared to the SIMPLY architecture in **Figure 9c**.

Despite the increased complexity of the peripheral circuits and the possible reduced parallelism, the proposed hybrid accelerator provides a more efficient exploitation of the hardware resources that are often scarce in low-power embedded devices.

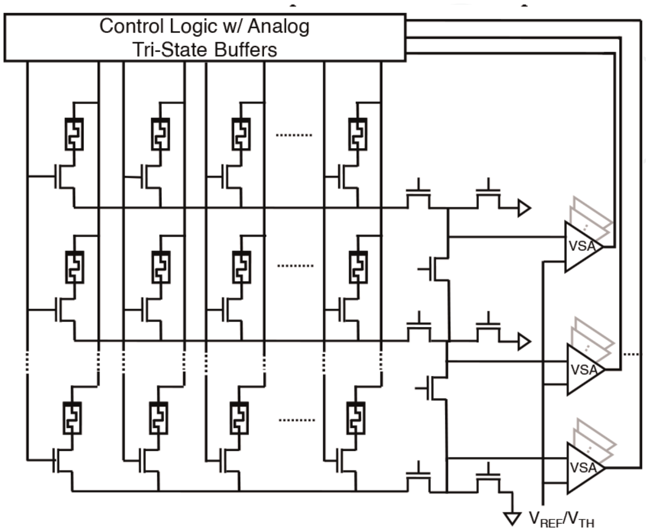


Figure 16.
*Sketch of the hybrid IMC accelerator enabling the coexistence of the SIMPLY and analog BNN VMM frameworks shown in **Figure 9c** and **15a**, respectively. Shaded VSAs highlight the possible use of flash ADCs in the architecture.*

4.4 Performance benchmark

In this section, the performance of the architectures discussed in sections 4.1–4.3, are estimated on a BNN inference task, and benchmarked against an implementation running on conventional embedded system hardware [48]. The performance was estimated on a classification task on the MNIST handwritten digits dataset. The 20x20 pixels images of the dataset were converted to black and white images before training. The adopted neural network consists of a single fully connected hidden layer with 1000 neurons and 10 output neurons and was trained on 9500 sample images using the DoReFa-Net algorithm [49]. Other 2500, and 2000 images were used for validation, and testing, respectively, and the trained neural network achieves an accuracy of 91.4% [29], which is comparable with other results in the literature and the BNN implementation considered as a reference [48]. The performance of each accelerator was estimated considering that the computations were executed on an appropriate number of 256x256 1T1R memory arrays. In the case of the SIMPLY-based implementations, the network parameters were mapped to memory arrays as discussed in Section 4.1.6, while in the hybrid architectures, the network weights and their complements were mapped to memory arrays, filling them completely, and parameters and devices required for computing logic operations in the SIMPLY framework were mapped onto other memory arrays. The two approaches require a similar number of arrays since most of the memory is used to store the neural network parameters. Energy and latency estimates were computed by mapping all the IMPLY, FALSE, and VMM operations required to implement the BNN inference task to the memory arrays. For the SIMPLY accelerator, two different implementations were studied to evaluate the performance improvement provided by the binary tree adder accumulator with respect to the HA chain implementation. All the test set was randomly shuffled and classified by the implemented neural network, preserving the memory states between classification tasks. In the case of SIMPLY-based implementations, the worst-case energy consumption for n-IMPLY and FALSE operations was estimated by considering the data of the maximum energy consumption for each input combination available in **Table 2** that were estimated by means of circuit simulations performed considering a 500 MHz clock frequency, meaning that IMPLY and FALSE operations are computed in 4 ns, and including the comparator overhead. The energy for BNN analog VMM operations was estimated considering RRAM devices approximated as resistors and computing the equivalent resistance for each active row of the array. To estimate the latency, when possible, operations are parallelized among different arrays, while inside each array two different degrees of parallelism were considered. Specifically, in the "serial" (see **Table 3**) case no

Operation	Input combination/Worst-case Energy
FALSE	(0/12 fJ), (1/190 fJ)
2-IMPLY	(00/509 fJ), (01/6.19 fJ), (11/6.5 fJ)
3-IMPLY	(000/409 fJ), (001/11.6 fJ), (011/13.1 fJ), (111/13.3 fJ)
4-IMPLY	(0000/409 fJ), (0001/11.6 fJ), (0011/13.1 fJ), (0111/13.3 fJ), (1111/13.7 fJ)

The effects of variability, RTN, and the comparator overhead, are included.

Table 2.
Worst-case energy estimates from circuits simulations of FALSE and n-IMPLY operations executed on the SIMPLY architecture.

Implementation	# 256x256 Arrays	# Steps	Latency	Energy	EDP Improvement
Embedded System [48]	NA	NA	17.35 ms	5.37 mJ	1
n-SIMPLY Serial HA chain	29	10,153,416	40.6 ms	3.27 μJ	$7•10^2$
n-SIMPLY Parallel HA chain	29	724,306	2.9 ms	3.27 μJ	$9.8•10^3$
n-SIMPLY Serial Binary Tree Adder	29	1,647,124	6.6 ms	1.69 μJ	$8.4•10^3$
n-SIMPLY Parallel Binary Tree Adder	29	406,280	1.6 ms	1.69 μJ	$3.4•10^4$
Hybrid Serial	28	214,795	859 μs	907 nJ	$1.2•10^5$
Hybrid Parallel	28	112,518	450 μs	907 nJ	$2.3•10^5$

(NA = not applicable).

Table 3.
Benchmark of different SIMPLY-based BNN accelerators against a state-of-the-art embedded system implementation from the literature.

additional parallelism was introduced, and operations were all executed sequentially. In the "parallel" (see **Table 3**) case the read step is executed in parallel on the rows of the array but, due to the high I_c of the considered technology (i.e., I_C = 100 μA), the conditional programming step is performed sequentially on the rows of the array. For the computation of BNN VMM operation in the hybrid accelerators, the same design choice performed in [8] was considered. Up to 128 lines were activated in parallel, 3-bit flash ADCs were used to digitize the partial results of the products, and each ADC was shared among 8 rows of the array. The digitized partial results were written onto other arrays, grouping on the same rows of the array the partial results related to the same neuron. Logic operations required to accumulate the results and to compute activation functions were executed in the SIMPLY framework.

The results of the simulations are reported in **Table 3** and show that all the proposed accelerators provide a > $7•10^2$ EDP improvement with respect to the reference embedded system implementation [48]. All the accelerators considerably reduce the energy consumption, and except for the "n-SIMPLY Serial HA chain" implementation, they can also considerably reduce the computing time. In the SIMPLY-based accelerators, the adoption of the binary tree adder accumulator reduces considerably the number of computing steps leading to considerable latency and energy savings. As expected, additional improvements can be achieved by performing VMM in the analog domain using the hybrid accelerators, which can achieve an EDP improvement of $>10^5$ and > 3.5 with respect to the embedded system implementation and the SIMPLY-based implementations, respectively.

The latency of SIMPLY-based implementations could be further improved by adopting RRAM technologies with lower current compliance provided that a sufficient read margin and high retention are preserved. Adopting lower current compliances would potentially increase the maximum number of devices that can be written in parallel. Also, lower current compliance values would reduce the energy consumption in both computing approaches further highlighting the advantages of both approaches.

In addition, to compare the performance of the n-SIMPLY and hybrid BNN accelerators, the performance and characteristics of RRAM-based BNN inference accelerators from the literature [6, 8, 42, 46, 47, 50, 51] are summarized in **Table 4**. In general, these works employ the different schemes discussed in Section 4.2.1 to

Author(s)	VMM Implementation	RRAM Array Topology (Size)	Sim./Exp.$^{\&}$	TOPS/W (For MAC ops.)	MNIST Inference Energy	MNIST Inference Latency	MNIST Inference EDP (J•s)
This Work n-SIMPLY Binary Tree Adder (Serial/Parallel)	LIM	1T1R (256x256)	Sim.	/	1.69 µJ	6.6 ms/ 1.6 ms	$\approx 1 \cdot 10^{-8}$/ $2.6 \cdot 10^{-9}$
This Work Hybrid (Serial/Parallel)	Analog + LIM (3-bits ADC + 8:1 MUX for the columns)	1T1R (256x256)	Sim.	/	907 nJ	859 µs/ 450 µs	$\approx 7.8 \cdot 10^{-10}$/ $4 \cdot 10^{-10}$
Minguet et al. [50]	Analog	1S1R	Sim. (28 nm CMOS)	/	42.5 µJ (Reported)	103 µs$^{\$\$}$ (Estimated)	$4.4 \cdot 10^{-9}$ (Estimated)
Yin et al. [8]	Analog (3-bits ADC + 8:1 MUX for the columns)	1T1R (128x64)	Exp. (90 nm CMOS)	24.1$^{\$}$ (Reported)	/	/	/
Yu et al. [42]	Analog (Online training)	1T1R (32 x 512x1024)	Exp. (130 nm CMOS)	/	2.4 mJ$^{\$\$\$}$ (Reported)	2.7 s$^{\$\$\$}$ (Reported)	$6.5 \cdot 10^{-3}$ (Estimated)
Zhao et al. [47]	Analog (Low-power voltage mode sensing scheme)	1T1R (32x32)	Sim. (180 nm CMOS)	19.7 (Reported)	/	/	/
Zhao et al. [47]	Analog (Low-power voltage mode sensing scheme)	1T1R (32x32)	Sim. (40 nm CMOS)	63 (Reported)	/	/	/
Pal Chowdhury [6]	Analog (Voltage sensing)	2R	Sim. (45 nm CMOS)	/	/	/	/
Ezzadeen et al. [46]	Analog (Capacitive neuron)	2T2R (32x32)	Exp. (130 nm CMOS)	/	/	/	/
Lopez et al. [51]	Analog (Subthreshold read scheme)	1S1R (32x32)	Sim. (28 nm CMOS)	/	≈ 62 nJ$^{\$\$\$\$}$ (estimated)		

$^{\&}$Exp. is used to indicate that at least one array prototype was fabricated, and its performance experimentally measured.$^{\$}$Best-case for a 64x64 layer, so that no external logic is required and included in the energy-efficiency estimate.$^{\$\$}$For arrays of arbitrary size, one neuron output computed per read operation (t_{READ} = 100 ns as reported). The ADC and external logic overhead and the limitation of the maximum current in a column is not considered.$^{\$\$\$}$Data from [52].$^{\$\$\$\$}$Estimated considering the reported energy per single bit read operation (i.e., 76 fJ/bit).

Table 4.
Performance comparison of different BNN inference accelerators from the literature.

accelerate in the analog domain the BNN VMM. However, each work uses different arrays sizes, topologies, devices, and technologies, thus complicating the comparison between different accelerators. Ideally, for a direct comparison between different implementations the same task should be executed on all the accelerators, however these data are rarely reported. Some works focus only on demonstrating the feasibility of their proposed implementation of the VMM [46], while other studies simulate a different inference task [6], or report different metrics that cannot be easily used to estimate the performance on a specific inference task. Specifically, in [8, 47] the TOPS/W metric is reported, however this metric indicates the maximum performance that can be achieved in specific conditions, which are not necessarily the one achieved by the circuit in a generic inference task. Still, some works [42, 50–52] directly report the performance, or sufficient data that can be used to estimate the performance of their accelerator on an MNIST handwritten digits classification task. Among these works, the results reported by Yu et al. in [42, 52] show the worst performance, however their solution was optimized for training directly on chip the entire network parameters, thus introducing additional overheads. Compared with the results reported by Minguet et al. in [50], the n-SIMPLY Binary Tree Adder implementation achieves similar EDP performance while the Hybrid implementation can further reduce the EDP. The lowest energy consumption for an MNIST inference task, was estimated from the work of Lopez et al. [51], where the authors proposed a subthreshold read scheme in which 1S1R devices are read using a read voltage that is lower than the threshold voltage of the selector device, achieving a read energy of 76 fJ/bit.

5. Conclusions

In this chapter, we discussed the implementation of different RRAM-based IMC accelerators describing an appropriate methodology for their analysis and design. The methodology is enabled by the use of physics-based compact models and consists in studying, by means of circuit simulations which also include devices nonideal effects, the performance and reliability of the core elements of IMC frameworks. The results of these simulations are then used to project the performance to larger architectures implemented on multiple arrays. An analysis of the reliability and performance of two LIM frameworks based on the material implication logic was presented. Due to its better performance and reliability, the SIMPLY architecture was used to implement a BNN hardware inference accelerator. The limitations of this approach were discussed and the benefits of using analog accelerators for the VMM operations were examined. Also, the implementation of an architecture which combines both approaches on the same hardware was discussed. The EDP estimates of these different RRAM-based BNN inference accelerators were compared against a state of the arts embedded system implementation, demonstrating considerable energy improvements. Overall, these hardware accelerators, by providing high reconfigurability and energy efficiency, represent a valuable solution for the implementation of future ultra-low power hardware.

Acknowledgements

Project funded under the National Recovery and Resilience Plan (NRRP), Mission 04 Component 2. Investment 1.5 – NextGenerationEU, Call for tender n. 3277 dated 30/12/2021. Award Number: 0001052 dated 23/06/2022.

Author details

Tommaso Zanotti*, Paolo Pavan and Francesco Maria Puglisi
Dipartimento di Ingegneria "Enzo Ferrari," Università di Modena e Reggio Emilia, Modena, Italy

*Address all correspondence to: tommaso.zanotti@unimore.it

References

[1] Xiao TP, Bennett CH, Feinberg B, Agarwal S, Marinella MJ. Analog architectures for neural network acceleration based on non-volatile memory. Applied Physics Reviews. 2020; **7**(3):031301. DOI: 10.1063/1.5143815

[2] Kvatinsky S, Belousov D, Liman S, Satat G, Wald N, Friedman EG, et al. MAGIC—Memristor-aided logic. IEEE Transactions on Circuits and Systems II: Express Briefs. 2014;**61**(11):895-899. DOI: 10.1109/TCSII.2014.2357292

[3] Kvatinsky S, Satat G, Wald N, Friedman EG, Kolodny A, Weiser UC. Memristor-based material implication (IMPLY) logic: Design principles and methodologies. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 2014;**22**(10):2054-2066. DOI: 10.1109/TVLSI.2013.2282132

[4] Borghetti J, Snider GS, Kuekes PJ, Yang JJ, Stewart DR, Williams RS. 'Memristive' switches enable 'stateful' logic operations via material implication. Nature. 2010;**8**(464):873. DOI: 10.1038/nature08940

[5] Zanotti T, Puglisi FM, Pavan P. Reliability-aware design strategies for Stateful logic-in-memory architectures. IEEE Transactions on Device and Materials Reliability. 2020;**20**(2):278-285. DOI: 10.1109/TDMR.2020.2981205

[6] Pal Chowdhury A, Kulkarni P, Nazm BM. MB-CNN: Memristive binary convolutional neural networks for embedded Mobile devices. Journal of Low Power Electronics and Applications. 2018;**8**(4):38. DOI: 10.3390/jlpea8040038

[7] Yin S, Kim Y, Han X, Barnaby H, Yu S, Luo Y, et al. Monolithically integrated RRAM- and CMOS-based In-memory computing optimizations for efficient deep learning. IEEE Micro. 2019;**39**(6):54-63. DOI: 10.1109/MM.2019.2943047

[8] Yin S, Sun X, Yu S, Seo JS. High-throughput In-memory computing for binary deep neural networks with monolithically integrated RRAM and 90-nm CMOS. IEEE Transactions on Electron Devices. 2020;**67**(10):4185-4192. DOI: 10.1109/TED.2020.3015178

[9] Courbariaux M, Hubara I, Soudry D, El-Yaniv R, Bengio Y. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. arXiv preprint. 2016. Available from: https://arxiv.org/abs/1602.02830

[10] Puglisi FM, Zanotti T, Pavan P. Unimore resistive random access memory (RRAM) Verilog-a model. nanoHUB. 2019. DOI: 10.21981/15GF-KX29. Available from: https://nanohub.org/publications/289/about/1#citethis

[11] Zanotti T, Pavan P, Puglisi FM. Comprehensive physics-based RRAM compact model including the effect of variability and multi-level random telegraph noise. Microelectronic Engineering. 2022;**27**:111886. DOI: 10.1016/j.mee.2022.111886

[12] Yu S, Wu Y, Chai Y, Provine J, Wong HSP. Characterization of switching parameters and multilevel capability in HfOx/AlOx bi-layer RRAM devices. In: Proceedings of 2011 International Symposium on VLSI Technology, Systems and Applications. Hsinchu, Taiwan: IEEE; 2011. pp. 1-2

[13] Zahoor F, Azni Zulkifli TZ, Khanday FA. Resistive random access memory (RRAM): An overview of materials, switching mechanism,

performance, multilevel cell (mlc) storage, modeling, and applications. Nanoscale Research Letters. 2020;**15**(1): 90. DOI: 10.1186/s11671-020-03299-9

[14] Wong HSP, Lee HY, Yu S, Chen YS, Wu Y, Chen PS, et al. Metal–Oxide RRAM. Proceedings of the IEEE. 2012; 100(6):1951–1970. DOI: 10.1109/JPROC.2012.2190369

[15] Kozicki MN, Barnaby HJ. Conductive bridging random access memory— Materials, devices and applications. Semiconductor Science and Technology. 2016;**31**(11):113001. DOI: 10.1088/0268-1242/31/11/113001

[16] Bersuker G, Gilmer DC, Veksler D, Yum J, Park H, Lian S, et al. Metal oxide RRAM switching mechanism based on conductive filament microscopic properties. In: 2010 International Electron Devices Meeting, San Francisco, CA, USA. New York City, USA: IEEE; 2010. pp. 19.6.1-19.6.4

[17] Celano U, Fantini A, Degraeve R, Jurczak M, Goux L, Vandervorst W. Scalability of valence change memory: From devices to tip-induced filaments. AIP Advances. 2016;**6**(8):085009. DOI: 10.1063/1.4961150

[18] Puglisi FM, Zagni N, Larcher L, Pavan P. Random telegraph noise in resistive random access memories: Compact modeling and advanced circuit design. IEEE Transactions on Electron Devices. 2018;**65**(7):2964-2972. DOI: 10.1109/TED.2018.2833208

[19] Panda D, Sahu PP, Tseng TY. A collective study on modeling and simulation of resistive random access memory. Nanoscale Research Letters. 2018;**13**(1):8. DOI: 10.1186/s11671-017-2419-8

[20] Zanotti T, Pavan P, Puglisi FM. Self-consistent automated parameter extraction of RRAM physics-based compact model. In: ESSDERC 2022 - IEEE 52nd European Solid-State Device Research Conference (ESSDERC). 2022. pp. 316-319

[21] Li H, Jiang Z, Huang P, Wu Y, Chen H, Gao B, et al. Variation-aware, reliability-emphasized design and optimization of RRAM using SPICE model. In: 2015 Design, Automation Test in Europe Conference Exhibition (DATE), Grenoble, France. New York City, USA: IEEE; 2015. pp. 1425-1430

[22] Bengel C, Siemon A, Cuppers F, Hoffmann-Eifert S, Hardtdegen A, von Witzleben M, et al. Variability-aware modeling of filamentary oxide-based bipolar resistive switching cells using SPICE level compact models. IEEE Trans Circuits Syst I. 2020;**67**(12):4618-4630. DOI: 10.1109/TCSI.2020.3018502

[23] Guan X, Yu S, Wong HP. A SPICE compact model of metal oxide resistive switching memory with variations. IEEE Electron Device Letters. 2012;**33**(10): 1405-1407. DOI: 10.1109/LED.2012.2210856

[24] Lehtonen E, Laiho M. Stateful implication logic with memristors. In: 2009 IEEE/ACM International Symposium on Nanoscale Architectures, San Francisco, CA, USA. New York City, USA: IEEE; 2009. pp. 33-36

[25] Xie L, Du Nguyen HA, Yu J, Kaichouhi A, Taouil M, AlFailakawi M, et al. Scouting logic: A novel Memristor-based logic Design for Resistive Computing. In: 2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI). 2017. pp. 176-181

[26] Zanotti T, Puglisi FM, Pavan P. A smart logic-in-memory architecture for low-power non-von Neumann

computing. IEEE Journal of the Electron Devices Society. 2020;**8**:1-1. DOI: 10.1109/JEDS.2020.2987402

[27] Lehtonen E, Poikonen JH, Laiho M. Two memristors suffice to compute all Boolean functions. Electronics Letters. 2010;**46**(3):230-231. DOI: 10.1049/el.2010.3407

[28] Zanotti T, Puglisi FM, Pavan P. Energy-efficient non-Von Neumann computing architecture supporting multiple computing paradigms for logic and Binarized neural networks. Journal of Low Power Electronics and Applications. 2021;**11**(3):29. DOI: 10.3390/jlpea11030029

[29] Zanotti T, Puglisi FM, Pavan P. Reliability and performance analysis of logic-in-memory based Binarized neural networks. IEEE Transactions on Device and Materials Reliability. 2021;**21**:1-1. DOI: 10.1109/TDMR.2021.3075200

[30] Lehtonen E, Poikonen J, Laiho M. Implication logic synthesis methods for memristors. In: 2012 IEEE International Symposium on Circuits and Systems, Seoul, Korea (South). New York City, USA: IEEE; 2012. pp. 2441-2444

[31] Zanotti T, Pavan P, Puglisi FM. Multi-input logic-in-memory for ultra-low power non-Von Neumann computing. Micromachines. 2021; **12**(10):1243. DOI: 10.3390/mi12101243

[32] Stine JE, Castellanos I, Wood M, Henson J, Love F, Davis WR, et al. FreePDK: An open-source variation-aware design kit. In: 2007 IEEE International Conference on Microelectronic Systems Education (MSE'07). 2007. pp. 173-174

[33] Kvatinsky S, Wald N, Satat G, Kolodny A, Weiser UC, Friedman EG. MRL — Memristor Ratioed logic. In: 2012 13th International Workshop on Cellular Nanoscale Networks and their Applications. 2012. pp. 1-6

[34] Ali KA, Rizk M, Baghdadi A, Diguet JP, Jomaah J. MRL crossbar-based full adder design. In: 2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS). 2019. pp. 674-677

[35] Hoffer B, Rana V, Menzel S, Waser R, Kvatinsky S. Experimental demonstration of Memristor-aided logic (MAGIC) using valence change memory (VCM). IEEE Transactions on Electron Devices. 2020;**67**(8):3115-3122. DOI: 10.1109/TED.2020.3001247

[36] Escudero López M. Reliability-Aware Circuit Design to Mitigate Impact of Device Defects and Variability in Emerging Memristor-Based Applications [Thesis]. TDX (Tesis Doctorals en Xarxa). Catalonia, Spain: Universitat Politècnica de Catalunya; 2020

[37] Yu J, Du Nguyen HA, Abu Lebdeh M, Taouil M, Hamdioui S. Enhanced scouting logic: A robust Memristive logic design scheme. In: 2019 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH), Qingdao, China. New York City, USA: IEEE; 2019. pp. 1-6

[38] Talati N, Gupta S, Mane P, Kvatinsky S. Logic design within Memristive memories using Memristor-aided loGIC (MAGIC). IEEE Transactions on Nanotechnology. 2016; **15**(4):635-650. DOI: 10.1109/TNANO.2016.2570248

[39] Siemon A, Drabinski R, Schultis MJ, Hu X, Linn E, Heittmann A, et al. Stateful three-input logic with Memristive switches. Scientific Reports. 2019;**9**(1):1-13. DOI: 10.1038/s41598-019-51039-6

[40] Cheng L, Zhang MY, Li Y, Zhou YX, Wang ZR, Hu SY, et al. Reprogrammable logic in memristive crossbar for in-memory computing. Journal of Physics D: Applied Physics. 2017;**50**(50):505102. DOI: 10.1088/1361-6463/aa9646

[41] Junsangsri P, Han J, Lombardi F. Logic-in-memory with a nonvolatile programmable metallization cell. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 2016;**24**(2): 521-529. DOI: 10.1109/TVLSI.2015.2411258

[42] Yu S, Li Z, Chen P, Wu H, Gao B, Wang D, et al. Binary neural network with 16 Mb RRAM macro chip for classification and online training. In: 2016 IEEE International Electron Devices Meeting (IEDM). 2016. pp. 16.2.1-16.2.4

[43] Zanotti T, Puglisi FM, Pavan P. Reconfigurable smart In-memory computing platform supporting logic and Binarized neural networks for low-power edge devices. IEEE Journal on Emerging and Selected Topics in Circuits and Systems. 2020;**10**(4):478-487. DOI: 10.1109/JETCAS.2020.3030542

[44] Yu S, Chen PY, Cao Y, Xia L, Wang Y, Wu H. Scaling-up resistive synaptic arrays for neuro-inspired architecture: Challenges and prospect. In: 2015 IEEE International Electron Devices Meeting (IEDM), Washington, DC, USA. New York City, USA: IEEE; 2015. pp. 17.3.1-17.3.4

[45] Welser J, Pitera JW, Goldberg C. Future computing hardware for AI. In: 2018 IEEE International Electron Devices Meeting (IEDM). 2018. pp. 1.3.1-1.3.6

[46] Ezzadeen M, Majumdar A, Bocquet M, Giraud B, Noël JP, Andrieu F, et al. Low-overhead implementation of Binarized neural networks employing robust 2T2R resistive RAM bridges. In: ESSCIRC 2021 - IEEE 47th European Solid State Circuits Conference (ESSCIRC). 2021. pp. 83-86

[47] Zhao Y, Yu J, Zhang D, Hu Q, Liu X, Jiang H, et al. A 0.02% accuracy loss voltage-mode parallel sensing scheme for RRAM-based XNOR-net application. IEEE Transactions on Circuits and Systems II: Express Briefs. New York City, USA: IEEE. 2022;**69**(6):2697-2701. DOI: 10.1109/TCSII.2022.3157767

[48] McDanel B, Teerapittayanon S, Kung HT. Embedded Binarized neural networks. In: Proceedings of the 2017 International Conference on Embedded Wireless Systems and Networks. Uppsala, Sweden: Junction Publishing; 2017. pp. 168-173

[49] Zhou S, Wu Y, Ni Z, Zhou X, Wen H, Zou Y. DoReFa-net: Training low Bitwidth convolutional neural networks with low Bitwidth gradients. arXiv preprint. 2016. Available from: https://arxiv.org/abs/1602.02830

[50] Minguet Lopez J, Hirtzlin T, Dampfhoffer M, Grenouillet L, Reganaz L, Navarro G, et al. OxRAM + OTS optimization for binarized neural network hardware implementation. Semiconductor Science and Technology. 2022;**37**(1): 014001. DOI: 10.1088/1361-6641/ac31e2

[51] Lopez JM, Rummens F, Reganaz L, Heraud A, Hirtzlin T, Grenouillet L, et al. 1S1R sub-threshold operation in crossbar arrays for low power BNN inference computing. In: 2022 IEEE International Memory Workshop (IMW), Dresden, Germany. New York City, USA: IEEE; 2022. pp. 1-4

[52] Yu S. Binary Neural Network and Its Implementation with 16 Mb RRAM Macro Chip [Internet]. Available from: https://www.src.org/calendar/e006125/yu-presentation2.pdf