Special Session Paper: Formal Verification Techniques and Reliability Methods for RRAM-based Computing-in-Memory

Chandan Kumar Jha*, Sumit Kumar Jha‡, Ulf Schlichtmann§, Rolf Drechsler*†
* Institute of Computer Science, University of Bremen, Germany
† Cyber-Physical Systems, DFKI GmbH, Germany
‡ University of Florida, Gainesville, FL 32611, USA
§Chair of Electronic Design Automation, TU Munich, Germany
{chajha, drechsler}@uni-bremen.de, sumit.jha@ufl.edu, ulf.schlichtmann@tum.de

***Abstract*—Computing-in-memory (CIM) has gained immense traction owing to the benefits it provides in power, performance, and area. CIM can be done on a large variety of memory elements like SRAM, DRAM, RRAM, etc. In this work, we focus on the techniques using RRAM for the computations. First, we explain the stateful and non-stateful techniques to perform Logic-in-Memory (LiM) operations using RRAM crossbars. We discuss the methods that are used to guarantee the correctness of the mapping/micro-operations obtained from mapping tools like SIMPLE-MAGIC. Second, we discuss the flow-based computing techniques that alleviate the write operations when performing the LiM operations and use the sneak-path currents to perform the computations. We discuss the formal verification strategies developed to guarantee the correctness of the logic operations using flow-based computing. Lastly, we discuss the techniques that have been developed to enable reliable computations using RRAM devices even in the presence of variations, such as error suppression and error reduction techniques. Finally, we explain a technique employing basis vectors to reduce the necessity to reprogram RRAM-based crossbars.**

***Index Terms*—RRAMs, computing-in-memory, logic-in-memory, multiply-and-accumulate, flow-based computing**

I. Introduction

Non-von Neumann techniques are being actively explored in computing to mitigate the challenges of the memory bottleneck. Most of these architectures have been enabled as a result of the immense progress in the emerging devices [1]. Computing-in-memory (CIM) has gained traction using SRAMs, DRAMs, and various emerging devices [2], [3]. RRAMs are one of such devices that have been extensively investigated. RRAMs are used to perform computations in the digital domain as well as the analog domain [4]–[7].

In the digital domain, the computations can be mapped to the memristor crossbar by first realizing functionally complete Boolean functions and then mapping the required design to these functions. An example of a functionally complete Boolean function using the MAGIC design style is shown in Fig. 1. Typically, the functions can be implemented as stateful logic like MAGIC [8], or non-stateful logic like Majority [3], [9]. Both computing styles have their advantages and disadvantages and are being simultaneously

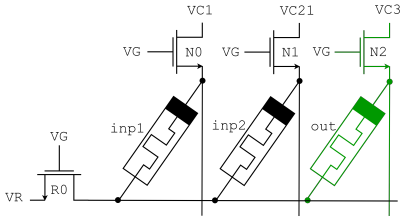

Fig. 1. NOR gate implementation using MAGIC Design Style

investigated and have also been experimentally validated [10], [11]. There are several works that have looked into the generation of the mapping for a given design to the RRAM crossbar operations [4], [5]. However, in recent years, formal verification of these mappings has only been explored [12]–[14]. This is crucial when developing automated design flows for CIM. Recent works have also identified bugs in some of the mappings obtained using these tools [12].

Complementary to the aforementioned CIM techniques, flow-based computing (see Section III) constitutes an alternative in-memory Boolean logic paradigm that transforms the inherent "sneak-path problem" into its computational engine. In this paradigm, each memristor in the crossbar is programmed to encode a literal or its negation, and when a bias voltage is applied to the input wordline, only those conductive paths whose conjunction of literals evaluates to true yield a measurable current at the output; all other paths remain at leakage levels [7], [15]–[17]. While this approach offers significant parallelism and energy efficiency, it also introduces challenges, such as unintended conductive loops, which necessitate formal verification methods. In addition to implementing designs on the memristor crossbar using logic operations, an alternative is to perform analog multiply-and-accumulate (MAC) operations.

Deep neural networks (DNNs) are widely being used successfully in many fields, e.g., image recognition and language processing. They also represent the foundation for generative AI. DNNs typically achieve their accuracy by using a large number of layers [18]. As a result, a neural

network will contain tens of millions of weights and need to perform hundreds of millions of MAC operations. Analog in-memory computing platforms, such as crossbar arrays based on emerging technologies, such as RRAM, have been proposed in order to efficiently execute such MAC operations. MAC operations are implemented in such platforms based on Ohm's law and Kirchhoff's current law, using analog devices. This results in high computation and energy efficiency.

However, these analog-based computing platforms suffer from manufacturing process variations [19], [20]. Also, owing to the huge size of today's neural networks, typically, an entire such network cannot be implemented onto an RRAM crossbar. As a result, frequent reprogramming is required, slowing down the computation efficiency of such a platform.

The rest of the paper is organized as follows. In Section II, we discuss the formal verification techniques that are used for the verification of LiM using stateful and non-stateful design styles. Section III discusses formal verification techniques for flow-based computing. In Section IV, we discuss the reliable computation for analog MAC operations using RRAMs and the techniques to reduce the necessity of reprogramming the crossbars. Our conclusions are discussed in Section V.

II. Formal Verification Methods for Stateful and Non-Stateful RRAM-based Computing-in-Memory

Logic-in-Memory (LiM) has been shown to offer significant benefits in terms of power, performance, and area compared to conventional computing. Hence, attempts have been made to automate the design flow for Logic-in-Memory (LiM) [21]. While LiM can be achieved using SRAMs, DRAMs, RRAMs, and other technologies, this work focuses on the design flows developed using RRAMs [2], [3], [22]. Mapping arbitrary designs to LiM has been extensively explored, and several automated design flows have been developed to achieve this goal. Mapping the designs onto the memristor crossbar can be achieved in several different design styles, depending upon the implementation of basic gates, i.e., MAGIC, IMPLY, FELIX, MAJ, etc [8], [9], [23], [24]. The basic principle is based on first implementing simple gates that are functionally complete, and then any design is mapped to these sets of gates. There are optimizations done to achieve the best mapping, i.e., in terms of the number of required RRAMs, number of gates, number of cycles required to do the computation, energy consumption, etc [4], [9], [24]. Since there are a wide variety of factors that need to be optimized, several automated methods for the mapping of the designs on the memristor crossbars have been explored [4], [5], [9], [25]. There also has been focus on the generation of Spice netlists from microoperations [26], which could then eventually be taped out to a chip.

While there have been several works that are focused on the generation of the mapping, fewer efforts have been made to guarantee its correctness. The correctness can traditionally be guaranteed by manual checking, which is tedious for large designs, or by functional simulations, which do not scale with design size and are slow. This gap is being addressed in recent years, where researchers are guaranteeing the correctness of the mapping using formal verification techniques. Formal verification techniques can guarantee 100 percent correctness and are ubiquitous in conventional design flows. Since the research related to LiM is moving in the same direction, it becomes crucial that verification flows are developed that are tailored to LiM architectures.

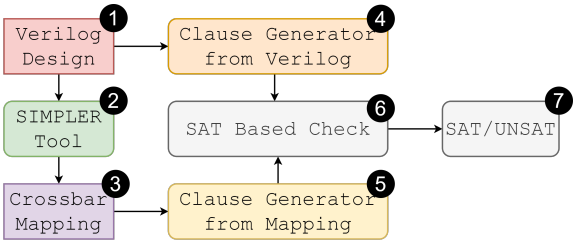

Fig. 2. VeriSIMPLER Framework adopted from [12].

We first discuss the prior works that have explored the formal verification techniques to guarantee the correctness of the mapping obtained. Formal verification of the mapping/micro-operations (MAP) is done with respect to a golden reference (GR) model by constructing a miter circuit and then asking the Boolean Satisfiability (SAT) to check for equivalence. If the MAP and GR are the same, there is no satisfying assignment that makes the output of the miter to be 1. Similarly, if the MAP is not equivalent to GR, a counterexample is found for which the MAP differs from GR [27], [28]. We then dive deeper into one of the recent works that has developed a formal verification methodology for the mapping obtained using the SIMPLER tool for the MAGIC design style. It has two input memristors and one output memristor. The output memristor is initialized to Logic '1' before the operation is performed.

In [29], the issue related to no guarantees on the correctness of the mapping generated by the SIMPLER MAGIC tool was highlighted as the tool does not provide any formal proof. The authors showed the limitation in the time required to perform the Spice-based simulations for correctness, and proposed using a behavioral model to accelerate the simulation to guarantee the correctness using VHDL simulations. However, this method is also based on simulations, which scale exponentially with the number of inputs in the design. Hence, they are impractical for larger designs.

In [5], a mapping framework for the Very Long Instruction Word (VLIW) based on the ReVAMP architecture was proposed. In addition to generating optimized mapping, the design also uses the ABC tool to formally verify the mapping generated using the tool. In this case, the mapping obtained is guaranteed to be correct as the formal verification process is integrated within the tool itself.

In [13], the authors proposed a formal verification methodology for the mapping obtained using the Majority (MAJ) logic. The authors proposed a file format for the 1T1R crossbar structure used to perform the logic operations. The authors translate this file into an intermediate data representation called the ReRAM Sequence Graph (ReSG).

The authors then generate the clauses from the original Majority Inverter Graph (MIG) and compare them against the clauses generated from the ReSG. This is done using the Python Z3 solver. Hence, the mapping can be formally verified and the guarantee for the correctness can be obtained.

In [14], a synthesis method was developed for the RRAM to generate the mapping. The verification methodology involved taking the mapping and generating various decision diagrams like Binary Decision Diagrams (BDDs), Multiplicative Binary Moment Diagrams (*BMDs), and Kronecker Multiplicative BMDs (K*BMDs) for verification. The adder designs with interleaved variable ordering gave significant improvement as compared to the Python Z3-based methods.

In [12], the author proposes veriSIMPLER, a state-of-the-art verification methodology for the mapping obtained from the SIMPLER MAGIC tool. The overall framework is shown in Fig. 2. This work does the formal verification of the mapping obtained from the SIMPLER mapping tool with the original Verilog design. It created a library of the basic gates available in the Verilog code to generate their corresponding clauses. There is a parser that generates the clause from the entire Verilog design. This clause serves as the golden reference. The clauses are then generated from the mapping obtained for the SIMPLER MAGIC tool. The tool maps the design to NOR and NOT gates, and the output needs to be initialized to a low resistance state before performing any operation. The clauses that are generated also take care of whether the output memristors have been initialized to a low resistance state. These clauses are then compared against the ones obtained from the Verilog design to perform the formal verification.

The authors identified a bug in the mapping that occurred whenever the mapping was done on the designs having their outputs directly connected to the input. This was identified as the technique was based on formal verification, and all the corner cases were analysed. The authors then also proposed a simple patch to correct the mapping and then performed formal verification on the corrected mapping. This was the first work that highlighted the issue with the mapping and further necessitated the point that formal verification is essential and needs to be incorporated in the design flow of LiM using RRAMs. In a recent work, the authors have proposed a formal verification methodology called veriSIM [30], to guarantee the correctness of the spice netlists generated for LiM using the MemSPICE tool [26]. It achieves this by generating the clauses from the spice netlists and verifies it against the clauses generated from the golden reference Verilog design.

There has also been research interest in the direction of performing formal error analysis for the analog computations using the memristor crossbars. The memristor crossbars can be used for the matrix vector multiplication (MVL), by exploiting the analog computations. Since there are non-idealities in the RRAMs, it can lead to larger deterioration in the output. In [31], a formal verification methodology for finding the maximum possible error in resistive-switching-based multilevel MVMs was developed.

III. Formal Verification Methods for Flow-Based In-Memory Computing

Resistive RAM (RRAM) crossbars implement *flow-based* in-memory Boolean logic [7] by programming each memristor at the intersection of a horizontal (wordline) and a vertical (bitline) nanowire to encode a literal or its negation, exploiting naturally occurring *sneak-path* currents as the computational primitive. When a bias voltage is applied to a designated input wordline, current propagates through all conductive paths composed of ON-state (low-resistance) memristors, each path corresponding to a minterm of the target Boolean function. If and only if the function evaluates to true under the current input assignment, a non-negligible current appears at a designated output wordline or bitline; otherwise, only leakage-level currents flow. Figure 3 shows a typical sneak path in a 4×3 crossbar. Flow-based computing [15]–[17] counterintuitively exploits the "sneak path problem" to perform computation using nanoscale RRAM crossbars.

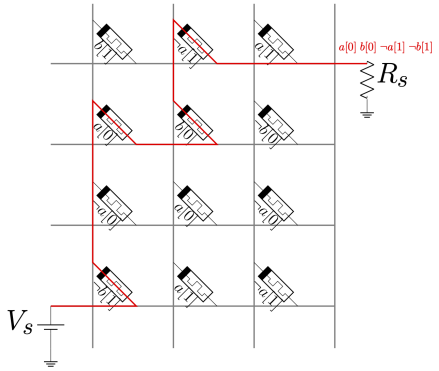

Fig. 3. Example sneak-path in a 4×3 RRAM crossbar. The red trace begins at V_s, passes through memristors $\neg b[1] \rightarrow a[0] \rightarrow b[0] \rightarrow \neg a[1]$, and finally drives the output resistor R_s computing the minterm $a[0]\ b[0]\ \neg a[1]\ \neg b[1]$.

However, if not designed correctly, the same network of sneak paths that implements intended minterms can also produce unintended conductive loops, introducing sneak-path leakage that corrupts outputs. In addition, real RRAM devices exhibit analog variability in their ON-state (R_{ON}) and OFF-state (R_{OFF}) resistances, narrowing sensing margins and potentially causing false logic interpretation absent a formal worst-case analysis. Thus, formal verification methods ranging from graph-based Boolean abstractions to bounded model checking are essential to ensure correctness, manage sneak-path constraints, and bound variability-induced errors in flow-based RRAM computing. Below, we survey three existing complementary approaches [32]–[34] that address the undirected, cyclic conductance graph induced by bidirectional ON-state memristors.

A. Graph-Based Path Extraction with Dynamic Shrinking

Flow-based RRAM crossbars create conductance graphs in which bidirectional ON-state memristors form cycles. Unlike classical combination circuits, which can be modeled as directed acyclic graphs, these cycles prevent the direct

application of standard SAT-based equivalence checking to flow-based computing. A graph-based path extraction framework overcomes this by casting verification as a graph-reachability problem [32]. Given a crossbar design D intended to implement Boolean formula φ, one constructs an undirected bipartite graph $G_D = (V, E)$, where each vertex in V represents a wordline (row) or bitline (column), and each edge in E corresponds to an ON-state memristor labeled by the literal it encodes (e.g., x or $\neg x$). Any simple path from the designated input node R_{in} (driven wordline) to the designated output node C_{out} (sensed bitline) corresponds to a minterm: the conjunction of literals labeling the edges along that path. Consequently, the function implemented by the crossbar φ_D is the disjunction of all these simple path implicants.

The direct enumeration of all the simple paths in G_D would lead to a combinatorial explosion. To mitigate this, two dynamic graph-shrinking rules are applied during path exploration. First, whenever a partial path has traversed literal x, any edge labeled $\neg x$ is removed, since any path containing both x and $\neg x$ is unsatisfiable. Second, if a partial path includes literal x, all remaining edges labeled x may be contracted into a single representative node as revisiting the same literal adds no new implicants. These two rules, the contradiction rule and the idempotence rule, significantly prune the search space by eliminating impossibilities and redundancies.

Once dynamic shrinking is exhausted, all remaining simple paths in the reduced graph G'_D are enumerated to extract a factored sum-of-products formula φ_D. In the subsequent phase, φ_D is compared against the specification φ by constructing a SAT-based equivalence miter. If the solver returns UNSAT, $\varphi_D \equiv \varphi$ and functional equivalence is certified; otherwise, a counterexample input demonstrates non-equivalence.

Experimental results [32] on 25 RevLib benchmarks validate the efficacy of dynamic shrinking. These findings confirm that graph-based path extraction, coupled with SAT-based miter checking on the extracted formula, can efficiently verify flow-based crossbars of moderate size. However, several opportunities remain open for future research. The use of paths coupled with rule-based simplifications could be replaced by more symbolic approaches that avoid such explicit enumeration. Methods that use richer data structures such as trees instead of paths may be able to scale better to large problems.

B. Time-Unrolled Bounded Model Checking

Rather than enumerate paths, one may *time-unroll* the undirected conductance graph into a directed acyclic model, enabling conventional SAT-based equivalence checking [33]. Starting from the compressed undirected bipartite graph $G_D = (V, E)$ obtained after removing OFF-state memristors and contracting constant ON edges, our approach introduces Boolean state variables for each wordline R_i^t and bitline C_j^t at each discrete time step $t = 0, 1, \ldots, T$, where $T = |V|$. At $t = 0$, the designated input row variable R_{in}^0 is asserted true, signifying the application of the bias voltage. For $t \geq 1$, the state transitions are defined as

$$R_i^t = \bigvee_{(i,j)\in E} \left(m_{ij} \wedge C_j^{t-1}\right), \quad i = 1, \ldots, M,\ 1 \leq t \leq T,$$

$$C_j^t = \bigvee_{(i,j)\in E} \left(m_{ij} \wedge R_i^{t-1}\right), \quad j = 1, \ldots, N,\ 1 \leq t \leq T,$$

with $R_{\text{in}}^t = 1$ for all $0 \leq t \leq T$. Here, m_{ij} is a Boolean constant indicating whether memristor (i, j) is ON under the current input assignment. The crossbar's implemented function then satisfies $\varphi_D = \bigvee_{t=0}^{T} R_{\text{out}}^t$, where R_{out} is the designated output row. Translating these transition relations and the output condition into CNF via Tseitin transformation yields a SAT instance $F(\varphi_D)$; combined with the specification φ as a miter $\big(F(\varphi_D) \oplus F(\varphi)\big)$, UNSAT implies $\varphi_D \equiv \varphi$.

A key challenge is that the SAT formula grows as $O(T|E|)$ clauses, which is prohibitive for large crossbars. To address this, a divide-and-conquer strategy fixes high-frequency input literals to concrete values, partitioning the original problem into smaller subproblems. After initial graph compression removing OFF memristors and contracting ON edges, input variables are sorted by occurrence frequency. The top k variables are recursively fixed to true or false, removing and contracting corresponding edges at each step. Once either the reduced graph falls below a threshold or a recursion depth limit is reached, the bounded model checking encoding is applied to the remaining subgraph and solved by SAT. Since each leaf subproblem is much smaller, parallel SAT solving dramatically reduces total runtime, outweighing the overhead of graph compression and decomposition.

On a suite of 19 MCNC benchmarks, the base time-unrolled formulation verified all instances within one hour, outperforming graph-based path extraction and prior neural-network heuristics [33]. Future efforts may focus on partitioning the bounded model checking problem using variables obtained by symbolic analysis such as model counting instead of frequency counts involving input literals.

C. Iterative Directed-Edge SAT Refinement

A third strategy [34] employs an iterative SAT refinement in which each ON-edge in the undirected conductance graph is assigned a direction under a given input, converting the graph into a directed acyclic graph. One introduces Boolean node variables R_i for each wordline i and C_j for each bitline j, indicating whether that wire conducts current. For each ON-state memristor at intersection (i, j) labeled by literal ℓ_{ij}, two "edge-orientation" variables $D_{i\to j}$ and $D_{j\to i}$ denote whether current flows from wordline i to bitline j or vice versa. Exactly one of these variables must be true whenever ℓ_{ij} evaluates to true. To enforce logical consistency, node and orientation variables satisfy

$$R_i \implies \bigvee_{\{\,j : m_{ij}=1\}} D_{j\to i}, \qquad C_j \implies \bigvee_{\{\,i : m_{ij}=1\}} D_{i\to j},$$

where m_{ij} is true if and only if ℓ_{ij} holds under the current input assignment. Assigning exactly one orientation to each

ON-edge prevents any 2-node cycle ($R_i \leftrightarrow C_j$); however, longer directed cycles (length ≥ 3) may persist.

To detect such spurious cycles, one constructs a directed graph G_{dir} whose vertices are $\{R_i\} \cup \{C_j\}$, including edge $(u \rightarrow v)$ whenever the SAT model sets the corresponding orientation variable to true. A standard directed-cycle detection algorithm is applied to G_{dir}. If no cycle is found, the assignment yields a valid conductive path from input to output that diverges from the specification, and non-equivalence is declared with a counterexample. If a directed cycle is detected, a single "cycle-blocking" clause is added to the SAT instance to forbid exactly that cycle. Specifically, if the cycle traverses $\{ D_{i_1 \to i_2}, D_{i_2 \to i_3}, \ldots, D_{i_k \to i_1} \}$, one introduces

$$\neg D_{i_1 \to i_2} \vee \neg D_{i_2 \to i_3} \vee \cdots \vee \neg D_{i_k \to i_1}.$$

The SAT solver is then re-invoked, and this cycle-blocking process repeats until the formula becomes UNSAT certifying equivalence, or yields a cycle-free satisfying assignment demonstrating non-equivalence.

Since only cycles actually produced by the SAT model are eliminated, the number of added clauses remains modest in practice. Moreover, this SAT formulation uses only $O(M + N + K)$ variables, where M and N are the numbers of wordlines and bitlines, and K is the number of ON-state memristors whose literals evaluate to true. This contrasts with time-unrolled formulations, which introduce a multiplicative factor of $T = |V|$ in variables. Empirical evaluation [34] on benchmark circuits shows that iterative directed-edge SAT refinement converges rapidly and typically require only a handful of cycle-blocking clauses per input assignment.

This approach often outperforms time-unrolled bounded model checking by an order of magnitude on medium to large crossbars, while maintaining a compact encoding that captures all potential conductive paths under ideal binary switching. Future research directions may include a tighter integration of the counterexample-guided inductive synthesis approach [35].

IV. Dealing with Non-idealities in RRAMs

RRAM crossbar arrays can perform multiply-and-accumulate (MAC) operations efficiently using Ohm's and Kirchhoff's laws. In RRAM cells, the conductances are programmed to specific values to represent computational weights of neural networks. However, such an analog system is inherently susceptible to process variations and environmental noise [36], [37]. Consequently, weights in neural networks implemented on the RRAM crossbar arrays can deviate from the expected values, leading to severe accuracy degradation.

The traditional approach to alleviate this problem is to reprogram the crossbar arrays, typically multiple times, to counteract the variations and keep the weight values accurate. In contrast to traditional approaches, we propose an alternative approach: error suppression and error compensation can be implemented in neural networks to deal with variations [38], [39].

In addition, due to limited resources, on-chip RRAM crossbars usually do not offer sufficient capacity to store the weights of a complete neural network at one time. Thus, to run a complete neural network, an RRAM crossbar is reprogrammed and reused many times to represent different weights. As a result, these frequent reprogramming operations can significantly decrease the computational efficiency of RRAM-based in-memory-computing and eventually make this new computing paradigm unusable.

To address this challenge, we propose BasisN, a technique that uses base decomposition, representing computational kernels as combinations of global basis vectors shared across all layers to reduce and even avoid reprogramming in RRAM crossbars [40].

A. Error suppression and error compensation in RRAM-based in-memory-computing

To avoid the amplification of errors across successive layers, Lipschitz constant regularization is introduced to suppress errors propagating throughout the network. A function $f : X \to Y$ is Lipschitz constrained if

$$\|f(x_1) - f(x_2)\|_p \leq k\|x_1 - x_2\|_p, \quad \forall x_1, x_2 \in \mathcal{X} \tag{1}$$

where the p-norm $\|\cdot\|_p$ denotes the p-norm distance between two vectors, and the Lipschitz constant $L(f) = k$ describes how f scales with respect to input changes. This property can be used to suppress errors in one layer. Assume the nominal input to a given layer is denoted as x_1, the input affected by variations in the previous layers is denoted as x_2, and the function of the layer that maps its input to the output is denoted as $f_i(\cdot)$. Then, the deviation in the output of the layer from the nominal value can be bounded by k. By constraining $k \leq 1$, any change in the input will not be amplified.

Assume that the variation of a weight has a log-normal distribution as

$$w = w_{\text{nominal}} * e^{\theta}, \quad \text{where } \theta \sim \mathcal{N}(0, \sigma^2) \tag{2}$$

where w_{nominal} is the nominal value of a trained weight, and θ is a Gaussian random variable with σ as its standard deviation. For each layer in the neural network, its function f_i can be decomposed into an affine transformation followed by a ReLU activation function. The ReLU function does not amplify input deviations, and its Lipschitz constant is always equal to 1. Therefore, to suppress error amplification in this layer, we require

$$\|(w \circ e_\theta) \cdot x_1 + b - [(w \circ e_\theta) \cdot x_2 + b]\|_p \leq k\|x_1 - x_2\|_p \tag{3}$$

where w is the weight matrix, b is the bias vector, and $\circ$ denotes the element-wise multiplication. We then use $\mu_{e^\theta} + 3 \cdot \sigma_{e^\theta}$ to bound the random variable e^θ, and the weights can be constrained as

$$\|w\|_p \leq \lambda, \quad \lambda = \frac{k}{e^{\frac{\sigma^2}{2}} + 3 \cdot \sqrt{(e^{\sigma^2} - 1)e^{\sigma^2}}} \tag{4}$$

We use the L_2 norm to bound w, and define the loss function as

$$\mathcal{L} = \mathcal{L}_{\text{CE}} + \beta \cdot \sum_{w_i \in \mathcal{W}} \left\| w_i^\top w_i - \lambda^2 I \right\|_2 \tag{5}$$

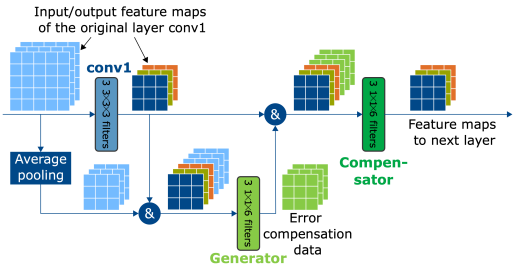

Fig. 4. Error compensation modules for a convolutional layer (adopted from [38]).

where $\mathcal{L}_{\text{CE}}$ is the original cross-entropy loss, w_i is the weight matrix of the ith layer, $\mathcal{W}$ is the set of all layer weight matrices, and β is a regularization hyperparameter. By applying the regularization term, the Lipschitz constraint is applied across all layers, and errors in all layers in the neural network can be suppressed.

Furthermore, lightweight error compensation is introduced to early layers of the neural network to enhance the accuracy. The compensation modules are illustrated in Figure 4 [38]. A generator first creates compensation data from layer inputs and outputs. The generator is a small convolutional neural network with m filters of size $1 \times 1 \times (l + n)$, where l and n are the numbers of input feature maps and output feature maps in the original layer (conv1), respectively. Afterwards, the compensation data is used by a compensator to reduce the errors propagated through this layer. The compensator is also a convolutional layer that takes the compensation data and the output feature maps of the original layer as input. This compensator contains n filters of size $1 \times 1 \times (n + m)$. To reduce the computational overhead, both the generator and the compensator use 1×1 convolution kernels. The error compensation is executed on digital circuits.

The locations and the number of filters of the compensation modules are determined using reinforcement learning (RL) with a policy neural network. To balance accuracy and computational cost, the reward function for training the policy neural network considers both inference accuracy improvement and the additional computational overhead introduced by the generator and compensator. In the experiments, the computational overhead was constrained to only 1%, 2%, or 3% of the total number of original network weights.

According to the experimental results in [38] using VGG16 [41] and LeNet-5 [42] to process CIFAR-100, CIFAR-10, and MNIST, the proposed method can recover inference accuracy from as low as 1.69% under variations and noise to over 95% of the original accuracy. Moreover, the proposed method is very general and can be combined with existing techniques to enhance computing accuracy of RRAM-based in-memory-computing platforms [43]–[47], as well as other analog computing solutions such as optical neural networks [48]–[53].

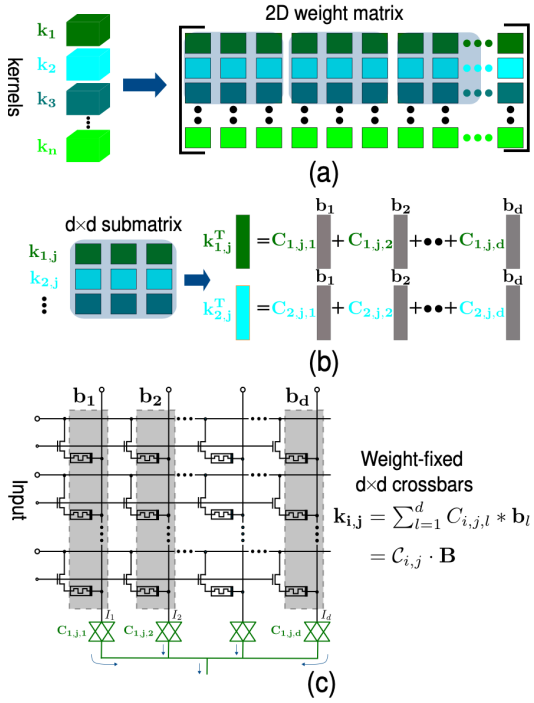

Fig. 5. Representation of the weights of a convolutional layer using BasisN. a) kernel reshaping. b) kernel representation as a linear combination of basis vectors. c) kernel implementation using basis vectors pre-programmed in a crossbar (adopted from [40]).

B. Programming reduction in RRAM in-memory-computing by base decomposition

To implement neural networks with various sizes onto the on-chip RRAM crossbar arrays, layers of a neural network are usually reshaped into 2D matrices and segmented into small submatrices as in Figure 5(a).

As any vector in a vector space can be represented as a linear combination of basis vectors spanning that space, we assume the weight matrices in a neural network can also be represented by combinations of a set of basis vectors. Figure 5(b) illustrates a $d \times d$ submatrix represented by weighted combinations of basis vectors as

$$k_{i,j} = \sum_{l=1}^{d} C_{i,j,l} \cdot b_l \tag{6}$$

where i denotes the kernel index, j denotes the partition index, and d denotes the crossbar dimension. $\{b_1, \ldots, b_d\}$ is the set of basis vectors stored in the crossbars and shared by all layers of the neural network mapped onto the crossbars and $C_{i,j,l}$ is the coefficient of the kernel partition $k_{i,j}$ for the lth basis vector.

The basis vectors are programmed onto the RRAM crossbar arrays once and the different weight matrices are realized by implementing different control coefficients of the basis vectors. To allow efficient hardware implementation of the neural network, the coefficients $C_{i,j,l}$ are restricted to specific

values or quantized values. Figure 5(c) shows an example using 1-bit control coefficients. The coefficients are realized by transmission gates at the bottom of the crossbar columns. By controlling the current flow with the transmission gates, the control bits $\{0, 1\}$ are realized. To improve the expressiveness of the weight matrices, multi-bit coefficients are necessary and are implemented through time multiplexing. At each time step, the computation for a single bit significance of the control coefficients is performed. The partial results from the time steps are shifted according to their bit significance and accumulated in the output registers in the digital domain.

Afterwards, the neural network is trained to match the combination of basis vectors. In the training, the basis vectors are initialized as random orthogonal matrices, and the coefficients are initialized to random values under the quantization constraints. To optimize the basis vectors and coefficients simultaneously, an alternating training manner is adopted. Specifically, the training alternates between optimizing the control coefficients while keeping the global basis vectors fixed, and fine-tunes the global basis vectors while keeping the control coefficients fixed.

The framework was evaluated on ResNet34 [54] and DenseNet121 [55] across CIFAR-100 and ImageNet datasets using 256×256 RRAM crossbars, and the control coefficients were set to 4 bits. With similar inference accuracy, cycles per inference and energy-delay product were reduced to below 1% compared with applying row-wise [56] and block-wise [57] reprogramming on crossbars, while the training and hardware costs are negligible.

V. Conclusion

In this paper, we summarize the recent efforts that are being investigated to guarantee the correctness of computations using RRAMs. We first discuss the formal verification techniques that are being investigated for LiM design styles that require write operations while performing the computations. Boolean satisfiability-based methods are being used for the verification of the mapping and the micro-operations for LiM using design styles like MAJ and MAGIC. We then discuss the flow-based computing [7] technique that uses the sneak paths in the memristor crossbar to perform computations, also called flow-based computing. Since the sneak paths are used to implement the Boolean function, formal verification is used to ensure that the Boolean functions are correctly implemented using the desired sneak paths. Unlike LiM techniques using MAGIC, SAT-based techniques cannot be used directly. Hence, three separate strategies for formal verification are shown. Lastly, we discuss the methodology that ensures reliable computations of MAC operations in memristor crossbars. It focuses on ensuring reliable computations, i.e., error suppression and error compensation techniques. In addition, the BasisN framework reduces the number of reprogramming cycles.

Acknowledgements

This work was supported by the German Research Foundation (DFG) within the Projects PLiM (DR 287/35-2), Project 457473137, and Reinhart Koselleck Project PolyVer under Grant DR 287/36-1, and National Science Foundation award 2408925.

References

[1] A. Chen, J. Hutchby, V. Zhirnov, and G. Bourianoff, *Emerging nanoelectronic devices*. John Wiley & Sons, 2014.

[2] O. Mutlu, A. Olgun, G. F. Oliveira, and I. E. Yuksel, "Memory-centric computing: Recent advances in processing-in-dram," in *2024 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2024, pp. 1–4.

[3] S. Shirinzadeh, M. Soeken, P.-E. Gaillardon, and R. Drechsler, "Logic synthesis for rram-based in-memory computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 7, pp. 1422–1435, 2017.

[4] R. Ben-Hur, R. Ronen, A. Haj-Ali, D. Bhattacharjee, A. Eliahu, N. Peled, and S. Kvatinsky, "Simpler magic: Synthesis and mapping of in-memory logic executed in a single row to improve throughput," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2434–2447, 2019.

[5] D. Bhattacharjee, A. Chattopadhyay, S. Dutta, R. Ronen, and S. Kvatinsky, "Contra: area-constrained technology mapping framework for memristive memory processing unit," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9.

[6] T. Schnittka, C. K. Jha, S. Ahmadi-Pour, and R. Drechsler, "River: Sneak path aware read-based in-memory computing for 1t1m memristive crossbars," in *2025 IEEE 28th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*. IEEE, 2025, pp. 31–36.

[7] S. K. Jha, D. E. Rodriguez, J. E. Van Nostrand, and A. Velasquez, "Computation of boolean formulas using sneak paths in crossbar computing," Apr. 19 2016, US Patent 9,319,047.

[8] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Magic—memristor-aided logic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 11, pp. 895–899, 2014.

[9] S. Shirinzadeh, M. Soeken, P.-E. Gaillardon, and R. Drechsler, "Fast logic synthesis for rram-based in-memory computing using majority-inverter graphs," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 948–953.

[10] A. Bende, S. Singh, C. K. Jha, T. Kempen, F. Cüppers, C. Bengel, A. Zambanini, D. Nielinger, S. Patkar, R. Drechsler *et al.*, "Experimental validation of memristor-aided logic using 1T1R TaO_x rram crossbar array," in *2024 37th International Conference on VLSI Design and 2024 23rd International Conference on Embedded Systems (VLSID)*. IEEE, 2024, pp. 565–570.

[11] L. Brackmann, T. Ziegler, D. J. Wouters, and S. Menzel, "Experimental verification and evaluation of non-stateful logic gates in resistive ram," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2024.

[12] C. K. Jha, K. Qayyum, K. Ç. Coşkun, S. Singh, M. Hassan, R. Leupers, F. Merchant, and R. Drechsler, "verisimpler: An automated formal verification methodology for simpler magic design style based in-memory computing," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2024.

[13] K. Bhunia, A. Deb, K. Datta, M. Hassan, S. Shirinzadeh, and R. Drechsler, "Resg: A data structure for verification of majority-based in-memory computing on reram crossbars," *ACM Transactions on Embedded Computing Systems*, vol. 23, no. 6, pp. 1–24, 2024.

[14] K. Qayyum, A. Kole, K. Datta, M. Hassan, and R. Drechsler, "Exploring the potential of decision diagrams for efficient in-memory design verification," in *Proceedings of the Great Lakes Symposium on VLSI 2024*, 2024, pp. 502–506.

[15] D. Chakraborty and S. K. Jha, "Automated synthesis of compact crossbars for sneak-path based in-memory computing," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 770–775.

[16] Z. Alamgir, K. Beckmann, N. Cady, A. Velasquez, and S. K. Jha, "Flow-based computing on nanoscale crossbars: Design and implementation of full adders," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2016, pp. 1870–1873.

[17] A. U. Hassen, D. Chakraborty, and S. K. Jha, "Free binary decision diagram-based synthesis of compact crossbars for in-memory computing," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 65, no. 5, pp. 622–626, 2018.

[18] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 2015.

[19] D. Niu, Y. Chen, C. Xu, and Y. Xie, "Impact of process variations on emerging memristor," in *ACM/IEEE Design Automation Conference (DAC)*, 2010, pp. 877–882.

[20] G. Pedretti, E. Ambrosi, and D. Ielmini, "Conductance variations and their impact on the precision of in-memory computing with resistive switching memory (rram)," in *2021 IEEE International Reliability Physics Symposium (IRPS)*. IEEE, 2021, pp. 1–8.

[21] C. Marchand, I. O'Connor, M. Cantan, E. T. Breyer, S. Slesazeck, and T. Mikolajick, "Fefet based logic-in-memory: an overview," in *2021 16th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*. IEEE, 2021, pp. 1–6.

[22] C.-J. Jhang, C.-X. Xue, J.-M. Hung, F.-C. Chang, and M.-F. Chang, "Challenges and trends of sram-based computing-in-memory for ai edge devices," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 5, pp. 1773–1786, 2021.

[23] S. Kvatinsky, A. Kolodny, U. C. Weiser, and E. G. Friedman, "Memristor-based imply logic design procedure," in *2011 IEEE 29th International Conference on Computer Design (ICCD)*. IEEE, 2011, pp. 142–147.

[24] S. Gupta, M. Imani, and T. Rosing, "Felix: Fast and energy-efficient logic in memory," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–7.

[25] X. Qian, C. Lv, Z. He, and W. Qian, "A recursive partition-based in-memory simd computation scheduler for memory footprint minimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[26] S. Singh, C. K. Jha, A. Bende, V. Rana, S. Patkar, R. Drechsler, and F. Merchant, "Memspice: Automated simulation and energy estimation framework for magic-based logic-in-memory," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2024, pp. 282–287.

[27] A. Gupta, M. K. Ganai, and C. Wang, "Sat-based verification methods and applications in hardware verification," in *International School on Formal Methods for the Design of Computer, Communication and Software Systems*. Springer, 2006, pp. 108–143.

[28] E. I. Goldberg, M. R. Prasad, and R. K. Brayton, "Using sat for combinational equivalence checking," in *Proceedings Design, Automation and Test in Europe. Conference and Exhibition 2001*. IEEE, 2001, pp. 114–121.

[29] M. A. Zaman, R. Joshi, and S. Katkoori, "High level modeling of memristive crossbar arrays," in *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2020, pp. 524–529.

[30] C. K. Jha, S. Singh, K. Qayyum, A. Bende, M. Hassan, V. Rana, F. Merchant, and R. Drechsler, "verisim: Formal verification of spice netlists for magic-based logic-in-memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2025.

[31] K. C. Coskun, C. K. Jha, M. Hassan, and R. Drechsler, "Formal verification of error bounds for resistive-switching-based multilevel matrix-vector multipliers," in *2025 26th International Symposium on Quality Electronic Design (ISQED)*, 2025, pp. 1–8.

[32] S. Thijssen, S. Kumar Jha, and R. Ewetz, "Equivalence checking for flow-based computing," in *2022 IEEE 40th International Conference on Computer Design (ICCD)*, 2022, pp. 656–663.

[33] S. Thijssen, S. Singireddy, M. R. H. Rashed, S. K. Jha, and R. Ewetz, "Verification of flow-based computing systems using bounded model checking," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[34] S. Thijssen, M. R. H. Rashed, M. R. Ahmed, S. Singireddy, S. K. Jha, and R. Ewetz, "Equivalence checking for flow-based computing using iterative sat solving," in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '24. New York, NY, USA: Association for Computing Machinery, 2025. [Online]. Available: https://doi.org/10.1145/3676536.3676721

[35] S. K. Jha, *Towards automated system synthesis using sciduction*. University of California, Berkeley, 2011.

[36] D. Niu, Y. Chen, C. Xu, and Y. Xie, "Impact of process variations on emerging memristor," in *ACM/IEEE Design Automation Conference(DAC)*, 2010.

[37] B. Liu, H. Li, Y. Chen, X. Li, Q. Wu *et al.*, "Vortex: Variation-aware training for memristor x-bar," in *ACM/IEEE Design Automation Conference(DAC)*, 2015.

[38] A. Eldebiky, G. L. Zhang, G. Boecherer, B. Li, and U. Schlichtmann, "CorrectNet: Robustness enhancement of analog in-memory computing for neural networks by error suppression and compensation," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023.

[39] ——, "CorrectNet+: Dealing with hw non-idealities in in-memory-computing platforms by error suppression and compensation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 2, pp. 573–585, 2023.

[40] A. Eldebiky, G. L. Zhang, X. Yin, C. Zhuo, I.-C. Lin *et al.*, "Basisn: Reprogramming-free rram-based in-memory-computing by basis combination for deep neural networks," in *IEEE/ACM International Conference on Computer-Aided Design(ICCAD)*, 2024.

[41] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *ArXiv*, 2014.

[42] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard *et al.*, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[43] L. Chen, J. Li, Y. Chen, Q. Deng, J. Shen *et al.*, "Accelerator-friendly neural-network training: Learning variations and defects in rram crossbar," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2017.

[44] S. Zhang, G. L. Zhang, B. Li, H. H. Li, and U. Schlichtmann, "Aging-aware lifetime enhancement for memristor-based neuromorphic computing," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2019, pp. 1751–1756.

[45] G. Charan, J. Hazra, K. Beckmann, X. Du, G. Krishnan *et al.*, "Accurate inference with inaccurate rram devices: Statistical data, model transfer, and on-line adaptation," in *ACM/IEEE Design Automation Conference (DAC)*, 2020.

[46] S. Zhang, G. L. Zhang, B. Li, H. H. Li, and U. Schlichtmann, "Lifetime enhancement for rram-based computing-in-memory engine considering aging and thermal effects," in *IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2020, pp. 11–15.

[47] Y. Zhu, G. L. Zhang, T. Wang, B. Li, Y. Shi *et al.*, "Statistical training for neuromorphic computing using memristor-based crossbars considering process variations and noise," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020, pp. 1590–1593.

[48] Y. Shen, N. C. Harris, S. Skirlo, M. Prabhu, T. Baehr-Jones *et al.*, "Deep learning with coherent nanophotonic circuits," *Nature photonics*, vol. 11, no. 7, pp. 441–446, 2017.

[49] Y. Zhu, G. L. Zhang, B. Li, X. Yin, C. Zhuo *et al.*, "Countering variations and thermal effects for accurate optical neural networks," in *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2020, pp. 1–7.

[50] J. Gu, Z. Zhao, C. Feng, M. Liu, R. T. Chen *et al.*, "Towards area-efficient optical neural networks: an fft-based architecture," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2020.

[51] J. Gu, Z. Zhao, C. Feng, H. Zhu, R. T. Chen *et al.*, "Roq: A noise-aware quantization scheme towards robust optical neural networks with low-bit controls," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020.

[52] A. Eldebiky, B. Li, and G. L. Zhang, "Nearuni: Near-unitary training for efficient optical neural networks," in *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2023.

[53] S. Fei, A. Eldebiky, G. L. Zhang, B. Li, and U. Schlichtmann, "An efficient general-purpose optical accelerator for neural networks," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2025.

[54] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[55] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[56] E. J. Merced-Grafals, N. Dávila, N. Ge, R. S. Williams, and J. P. Strachan, "Repeatable, accurate, and high speed multi-level programming of memristor 1t1r arrays for power efficient analog computing applications," *Nanotechnology*, vol. 27, no. 36, p. 365202, 2016.

[57] W.-L. Chen, F.-Y. Gu, C. Lin, G. L. Zhang, B. Li *et al.*, "A novel and efficient block-based programming for reram-based neuromorphic computing," in *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2023.