

# Enabling Computing-in-Memory with Conventional and Emerging Device Technologies

Sutirtha Bhattacharyya<sup>1</sup>, Mohamed Nadeem<sup>1</sup>, Sajjad Parvin<sup>1</sup>, Sallar Ahmadi-Pour<sup>1</sup>,  
Rolf Drechsler<sup>1,2</sup>, Chandan Kumar Jha<sup>1</sup>

<sup>1</sup>Institute of Computer Science, University of Bremen, Bremen, Germany

<sup>2</sup>Cyber-Physical Systems, DFKI, Bremen, Germany

{sutirtha, mnadeem, parvin, sallar, drechsler, chajha}@uni-bremen.de

**Abstract**—*Compute-in-Memory (CIM)* is an emerging paradigm that addresses the limitations of the memory bottleneck in conventional computing architectures by enabling computation within memory structures. Unlike traditional ASIC design flows, CIM requires novel design automation methodologies spanning device, circuit, architecture, and system levels. This work provides an overview of recent advances in CIM, with emphasis on developments in device technologies, circuit design methodologies, synthesis techniques, and formal verification frameworks. This work describes recent advancements in CIM using three memory technologies: *Static Random-Access Memory (SRAM)*, *Dynamic Random-Access Memory (DRAM)*, and *Resistive Random-Access Memory (RRAM)*. In this paper, we highlight the similarities in CIM at the higher level of abstraction, especially in synthesis and verification, and call for technology-independent automated flows. We believe that a technology-independent framework can help the transferability of synthesis and verification techniques across different technologies. This will lead to the identification of common open challenges and will promote standardization and provide a common basis for evaluating and developing methodologies across different technological platforms.

**Index Terms**—*Compute-in-Memory (CIM)*, *Static Random-Access Memory (SRAM)*, *Dynamic Random-Access Memory (DRAM)*, *Resistive Random-Access Memory (RRAM)*.

## I. INTRODUCTION

The last few decades have seen tremendous growth in the performance of processors, while memory performance has improved at a much slower pace. As a result, the performance gap between processors and memory has continued to widen over the years [1]. However, traditional von Neumann architectures separate processing and storage of data. Hence, various techniques are used to mitigate this performance gap between the processor and the memory [2], [3]. Despite these techniques, the continuous growth of data-centric applications such as machine learning has further exposed the limitations of these traditional architectures, as the transfer of data between two physically separated components dominates the latency and energy consumption [4]. *Compute-in-Memory (CIM)* provides a potential solution to this bottleneck by bringing computation closer to, or directly into, where the data is stored [5]–[7]. It exploits parallelism and physical characteristics of the memory arrays to perform computations and improve throughput, latency, and energy.

CIM has been implemented across a broad range of technologies like conventional memory technologies such as *Static Random-Access Memory (SRAM)* [8] and *Dynamic Random-Access Memory (DRAM)* [9], [10], and emerging non-volatile devices like *Resistive Random-Access Memory (RRAM)* [11], [12]. SRAM is particularly attractive for CIM due to its low-power operations, faster access time, and compatibility with advanced CMOS technologies, as it is on the same chip as the processor. SRAM-based CIM architectures have demonstrated both digital and analog computations ranging from Boolean functions to *Multiply-and-Accumulate (MAC)* operations [8]. The overview of SRAM-based CIM is shown in Figure 1 ①. DRAM provides high density, large storage capacity, and high parallelism,

enabling computation either near the memory arrays or directly within them, as shown in Figure 1 ②. These features make DRAM-based CIM attractive for bulk data processing and memory-intensive applications like machine learning [13]–[15].

Emerging technologies provide additional advantages for CIM by enabling computation through the characteristics of the memory devices themselves. In particular, RRAM has gained considerable interest owing to its non-volatility, high density, and compatibility with CMOS-based integration [12]. Analog RRAM CIM mainly exploits the programmable conductance states for highly parallel MAC and *Matrix-Vector Multiplication (MVM)* operations [16], [17], while digital RRAM CIM can execute Boolean functions directly within the memory arrays [18]. Figure 1 ③ shows the overview of RRAM-based CIM. Logic styles such as *Memristor-Aided Logic (MAGIC)* [19], Majority Logic [20], and FELIX [21] have demonstrated how the resistance states of RRAM can represent logic values and enable computation. Depending upon the application's requirements, the best technology can be identified and then used to perform CIM.

Realizing the full potential of CIM technologies requires not only sufficient circuit and device structures, but also efficient design automation and verification techniques that allow translation of high-level specifications into correct device-level implementations. The increasing complexity of CIM architectures has therefore shifted research attention from manually optimized circuits toward automated synthesis, mapping, scheduling, and verification methodologies. Figure 1 provides an overview of the discussed CIM technologies and their synthesis and verification flows. At the compute and memory levels, the three technologies show different characteristics and trade-offs. SRAM-based CIM offers low-power operations, fast access, and compatibility with advanced CMOS nodes, but suffers from lower density and higher area overhead. DRAM-based CIM provides high density, large storage capacity, and high parallelism. This makes DRAM-based CIM effective for data-intensive applications like machine learning. RRAM-based CIM provides high density and non-volatility and enables computation directly within memory arrays. Both RRAM- and SRAM-based CIM can compute arbitrary boolean functions and perform MAC and MVM operations through *Digital Compute-in-Memory (DCIM)* and *Analog Compute-in-Memory (ACIM)*, respectively. The design automation step shows the stages that takes a high-level specification to its corresponding CIM implementation through synthesis, scheduling and design space exploration. The verification step shows possible techniques to verify that a CIM implementation satisfies its functional requirements through simulation and formal verification techniques. The sub-techniques like fuzzing or *Boolean Satisfiability (SAT)*-based equivalence checking are closer to the broader technique they represent.

The objective of this survey is also to identify common design automation and verification abstractions across different CIM

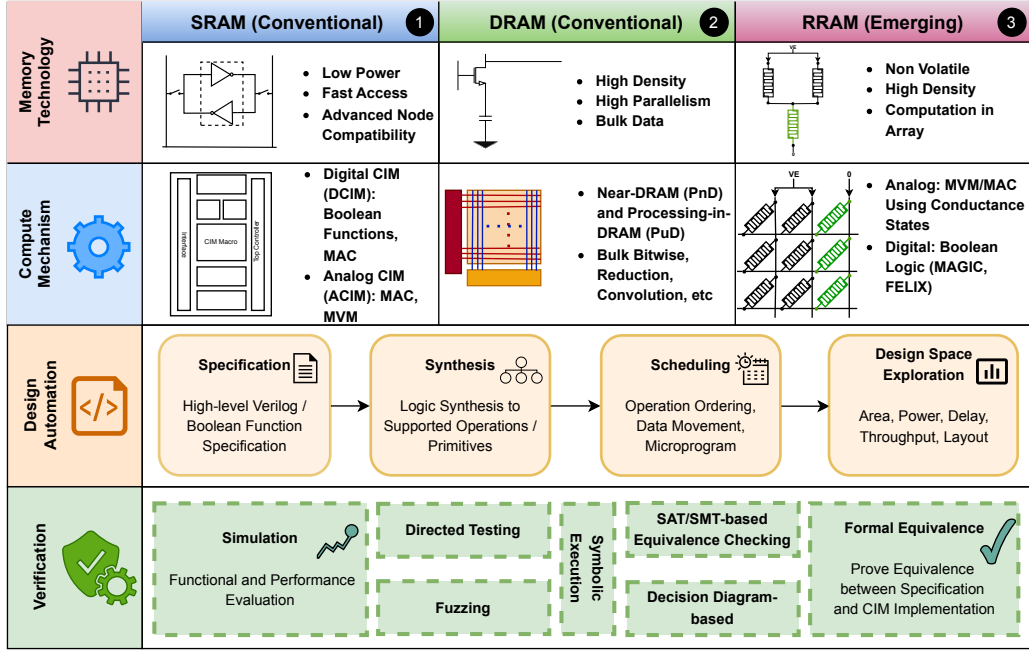


Figure 1: Across the Layer Landscape of Compute-in Memory

technologies. Although SRAM, DRAM, and RRAM vary in their device technologies and computational mechanisms, their higher-level design flows share several similar stages. A high-level specification is transformed into supported operations through synthesis, followed by scheduling and optimization before the final implementation. Similarly, verification aims to guarantee that the final implementation preserves the intent of the original specification. While the different CIM technologies vary at the device and computation levels, the synthesis and verification flows are presented using common higher-level abstractions. These sections of Figure 1 therefore aim to illustrate the potential for a technology-independent design flow. We therefore identify similarities and technology-specific differences in synthesis and verification and discuss opportunities for technology-independent automation frameworks.

In this work, we discuss the aspects of CIM from device technologies and circuit design to automated synthesis and verification frameworks. We first discuss conventional SRAM-based and DRAM-based CIM technologies with their associated synthesis techniques, followed by emerging RRAM-based CIM and its synthesis and verification techniques. We also consider automation challenges arising from synthesis, scheduling, and physical implementation, motivating the need for efficient formal verification techniques. Finally, we review existing formal verification methodologies and identify open challenges for developing verification flows for future CIM systems.

## II. SRAM-BASED CIM

SRAM is a well-suited candidate for CIM because its low-power operation, fast access time, and compatibility with advanced technology nodes [22] makes it superior to many existing CIM technologies. SRAM-based CIM can perform in both digital also known as DCIM and analog fashion which are known as ACIM. DCIM utilizes digital signals and logic, which provides substantially better robustness to process, voltage, and temperature variations than their ACIM counterparts [8]. Additionally, DCIM simplifies its integration with conventional digital systems compared to ACIM. Therefore, these

characteristics make SRAM an attractive choice for scalable CIM architectures.

Prior works have investigated the circuit and architectural design of SRAM-based CIM. These studies have proposed modified SRAM bitcells [23], compute-enabled read ports [24], local multiplication structures [25], bit-serial and bit-parallel datapaths [26], and different organizations of memory and compute logic [27]. Therefore, in the literature, it has been demonstrated that SRAM arrays can efficiently support operations ranging from simple Boolean functions to full-precision MAC operations. However, most of these architectures are manually designed and optimized for a particular precision, array organization, workload, or technology. As the number of possible design choices grow, manually exploring SRAM designs, computation precision, multiplier topology, accumulation structure, peripheral circuitry, and physical implementation becomes increasingly difficult.

This limitation has recently shifted from the design of individual tailored SRAM-based CIM circuits toward the automated synthesis of SRAM-based DCIM macros. Meaning, synthesis should be distinguished from the large literature on CIM circuit design. The objective is no longer merely to propose a new compute-capable SRAM macro, but to automate the transformation of architectural or performance requirements into a realizable CIM macro. The emergence of such tools is relatively recent. One of the first major efforts in this direction was AutoDCIM [28], which introduced a template-based automated compiler for DCIM macros. AutoDCIM takes user-defined architectural specifications and generates a corresponding DCIM implementation, including memory organization and arithmetic structures, showing that SRAM-based CIM can be produced in a compiler-like fashion rather than a fully manual approach. However, its template-driven approach relies on predefined architectural patterns, meaning that key design decisions are still left to the user. Next, in [29], the authors proposed ARCTIC, which extended the AutoDCIM approach by supporting a wider range of numerical formats and configurable precisions, including floating-point computation, and by generating the associated arithmetic and

memory structures. ARCTIC and AutoDCIM are very similar in the synthesis and generation of DCIM, meaning both rely on template-based approaches. However, ARCTIC integrates built-in self-test into the CIM macro, treating CIM testability as part of the compilation flow.

In [30], the authors take a different approach compared to previous works, namely, SynDCIM, which performs performance-aware sub-circuit synthesis by selecting from a library of arithmetic components (e.g., adders and compressors) based on area, power, and timing constraints, moving beyond fixed template-based designs. Building on top of SynDCIM, in [31], the authors introduce SEGA-DCIM, which proposes higher-level design-space exploration, automatically evaluating SRAM-CIM configurations to identify Pareto-optimal trade-offs in area, power, delay, and throughput. Extending beyond architectural optimization, DAMIL-DCIM focuses on physical synthesis [32]. DAMIL-DCIM uses dataflow-aware floorplanning and placement techniques to preserve locality between memory and compute and reduce interconnect overhead in SRAM-CIM macros. Finally, OpenC<sup>2</sup> provides an open-source end-to-end framework for generating and evaluating digital CIM designs, serving as a reproducible infrastructure that integrates and supports different architectures and compilation methodologies [33].

[34] introduces OpenACM, which extends SRAM-based CIM synthesis by introducing approximate computing into the design space, moving beyond traditional tools that focus only on exact arithmetic and performance, power, and area optimization. It incorporates exact, approximate, and logarithmic multipliers. It treats computational accuracy as an additional optimization dimension, which is especially useful for machine-learning workloads where slight accuracy loss can be tolerated for gains in energy or area efficiency. In addition, OpenACM integrates automated SRAM-based CIM generation, variation-aware characterization, and a full physical-design flow, arithmetic design, approximation, and synthesis. Its extension, OpenACMv2, further advances this approach by enabling cross-layer optimization that jointly explores architectural choices, approximate arithmetic, SRAM parameters, and device-level decisions under accuracy constraints. Though the tool is claimed to be open-source, it relies on a commercial VCS tool from Synopsys.

Despite the advances in automated synthesis and physical implementation, verification of SRAM-based CIM remains largely under-explored. Existing approaches have mainly targeted testing and fault detection rather than formally guaranteeing functional correctness of the CIM implementation. For example, [29] incorporates built-in self-tests into the CIM macro as part of the compilation flow and [35] investigates test strategies for CIM 8T SRAM. While these works are important for identifying manufacturing and circuit-level faults, they do not address the problem of formally verifying that a synthesized SRAM CIM macro implements the intended specification. As automated synthesis flows expand the design space, simulation-based validation becomes increasingly impractical, highlighting the need for formal verification techniques that can provide correctness guarantees for generated CIM implementations. However, formal verification of SRAM-based CIM remains largely unexplored.

The challenges discussed are not unique to SRAM based CIM. While other memory technologies introduce different circuit and device features, they raise similar questions regarding automated design and verification.

### III. DRAM-BASED CIM

The limitations of SRAM density and area overhead have motivated the exploration of alternative memory technologies for CIM.

Among these, DRAM [9], [10] is a well-suited memory technology for CIM due to its high density, large storage capacity, and internal structure, which enables many data elements to be accessed and processed concurrently. These characteristics make DRAM suitable for bulk data processing in memory-intensive applications such as machine-learning inference [13], [36], including recurrent neural networks [37], transformers and large language models [14], [15], model inference [38], and sparse matrix-vector multiplication [39].

The DRAM-based architectures generally follow either *Processing near DRAM (PnD)* or *Processing using DRAM (PuD)*. Both approaches offer distinct trade-offs. PnD-based architectures place dedicated processing units close to DRAM banks or arrays to reduce data movement while supporting a wide range of arithmetic and application-specific operations [14], [15], [40]–[42]. In contrast, PuD-based architectures perform computation directly inside the memory arrays by exploiting the behavior of DRAM cells and peripheral circuitry to perform primitive operations such as data initialization, copying, and bitwise logic operations directly within the memory arrays [39], [43]–[49].

Several works focused on enabling primitive operations within DRAM. RowClone [43] introduced bulk data initialization and copying as fundamental operations using consecutive row activations, avoiding unnecessary data movement between DRAM and the processor. Ambit [44] extended the DRAM computational capabilities. Through simultaneous *Triple-Row Activation (TRA)* and by exploiting the internal parallelism of DRAM, it enables bulk bitwise operations, including majority, AND, and OR functions. Furthermore, Ambit supports the NOT function using a *Dual-Contact Cell (DCC)*, which consists of a cell capacitor and two transistors connected to both sides of a sense amplifier.

Despite the success of RowClone and Ambit in performing primitive operations within DRAM, they require modifications in the memory cell to support TRA and DCC. Several works have shown that computational operations can be performed in *Commercial Off-The-Shelf (COTS)* DRAM without modifying the memory chip. ComputeDRAM [45] utilizes controlled command sequences with reduced timing parameters to perform row copy, majority, AND, and OR functions in COTS DDR3 chips. FracDRAM [46] extends COTS DRAM computations by storing fractional values in the DRAM cells and using them as inputs to perform majority operations through simultaneous four-row activation. Recently, the work [50] has introduced a functionally complete Boolean logic in COTS DDR4 chips, supporting NOT, NAND, NOR, AND, and OR operations.

As the complexity of target functions increases, manually constructing the required sequences of in-DRAM operations is challenging and error-prone. Therefore, automated synthesis approaches have been introduced to translate high-level functions into executable DRAM operation sequences [48], [51], [52]. SIMDGRAM [48] introduced a synthesis framework for implementing complex in-DRAM operations. It starts with a desired Boolean or arithmetic function and represents it using majority and NOT operations (i.e., *Majority-Inverter Graph (MIG)*). The resulting representation is then optimized to reduce the number of operations required for its implementation. Next, SIMDGRAM assigns the operands to designated compute rows and translates the optimized representation into a sequence of Activate-Activate-Precharge / Activate-Precharge operations that perform operations introduced in RowClone and Ambit. SIMDGRAM optimizes the operation sequence to reduce the number of DRAM-row activations and stores the optimized sequence as a  $\mu$ Program in the main memory. During execution, a *bbop* instruction triggers the memory controller to execute the corresponding  $\mu$ Program and store

the result in DRAM.

CHOPPER [51] extends the DRAM-based *Processing In Memory* (PIM) synthesis flow by focusing on high-level programmability. While SIMDram synthesizes individual functions, CHOPPER translates high-level parallel programs into instructions for bit-serial PuD architectures. It employs a bit-slicing compiler to automate memory allocation and generate bit-sliced code for a DRAM subarray, which is then translated into assembly instructions for the target architecture. It also optimizes the generated instructions using operation scheduling, value reuse, and instruction renaming to reduce data movement in both DRAM intra-subarray and data spilling (i.e., data movement outside the subarray due to insufficient subarray storage). CHOPPER demonstrates that the generated code achieves average speedups of  $1.20\times$  and  $1.26\times$  over Ambit and SIMDram, respectively, when data fit within DRAM subarrays, while the average speedup increases to  $12.61\times$  and  $9.81\times$  when data spilling is applied.

Recently, PIMsynth [52] proposed a fully automated end-to-end synthesis framework for bit-serial PIM architectures. While works [48], [51] focus on DRAM-based PuD architectures, PIMsynth targets both digital and analog PIM architectures. It starts with a Verilog description of the desired function and uses open-source logic synthesis tools such as Yosys [53] and ABC [54] to map the function to bit-serial operations. It also relies on LLVM [55] to perform instruction scheduling and register allocation before generating optimized micro-operation code. PIMsynth demonstrates that the generated micro-operation code performs within  $1.08\times$  and  $1.54\times$  of hand-optimized PIM implementations.

As the complexity of code generated by DRAM-based synthesis approaches increases, ensuring its functional equivalence to the desired function becomes crucial. Existing synthesis frameworks mainly integrate validation and testing techniques rather than formal verification. While validation and testing use simulation to identify incorrect behavior, formal verification aims to establish functional correctness for all possible inputs. For example, PIMsynth integrates the PIMEval simulation framework [56] to validate the functional behavior of its generated code. However, simulation generally provides limited input coverage and therefore cannot guarantee complete functional correctness.

Several works have investigated the formal verification of DRAM controllers [57], [58]. For example, the work [58] proposed a verification framework that automatically generates *SystemVerilog Assertions* (SVA) from DRAM protocol descriptions to verify that DRAM controllers satisfy the specific protocol requirements. However, these approaches focus on verifying controllers and protocols rather than ensuring functional correctness of the code generated by DRAM-based synthesis. Therefore, formally establishing the functional equivalence between a desired function and its generated in-DRAM code remains an open research problem.

#### IV. RRAM-BASED CIM

The architectural and computational characteristics of DRAM-based CIM differ substantially when compared to resistive memory technology. This motivates the exploration of RRAM as another promising technology for CIM. RRAM-based devices show promising potential for CIM [11], [12] owing to its high density, and CMOS compatibility. RRAM-based CIM can be broadly classified into analog and digital approaches [16]–[18], [59]. In analog CIM, the conductance of RRAM cells is used to perform highly parallel MAC and MVM operations and is mostly limited to these. On the other hand, digital RRAM CIM can perform operations for arbitrary Boolean functions directly on the memory array. This makes digital

CIM useful for *Electronic Design Automation* (EDA) as any arbitrary digital function can be mapped to RRAM crossbars. The development of RRAM-based CIM has therefore progressed from device-level demonstrations to automated synthesis, mapping, and verification.

At the device level, RRAM cells can be configured to *High Resistance State* (HRS) and *Low Resistance State* (LRS) based on the applied voltages. The states HRS and LRS can then be treated as logic 0 and logic 1 respectively to perform Boolean operations. Several design styles like MAGIC [19], Majority [20], and FELIX [21] have been proposed for implementation of Boolean logic on RRAM crossbars. While several theoretical and simulation-based studies have shown the possibility of logic execution using these states, practical implementation on fabricated arrays has also gained interest [59]–[61]. Among these, one of the demonstrations [59] validates logic computation of MAGIC gates using *One-Transistor-One-Resistor* (1T1R)  $TaO_x$  RRAM arrays. This work also demonstrates that more than 85% of the total gate energy is consistently consumed during the initialization phase, highlighting that the cost of preparation of the cells can be as expensive as the actual computation. More recently, a demonstration [62] on a CMOS-integrated  $TaO_x$ -based 1T1R RRAM computing fabric shows how intrinsic cycle-to-cycle and device-to-device variability limit logic reliability in RRAM based CIM architectures. However, the study also derives the device specifications and requirements to achieve error-free stateful CIM operations in the 1T1R RRAM arrays.

To synthesize a Boolean function for RRAM-based digital CIM, the synthesis process must account for the physical organization of the RRAM crossbar and its operational constraints. Thus, the synthesis flow for RRAM CIM is divided into the synthesis and the scheduling (also mapping) part. The synthesis step transforms the target arbitrary Boolean function into a netlist of supported operations. For example, MAGIC [19] supports NOT and NOR operations, so the target function is represented as a sequence of NOR and NOT operations. Note that, since NOR is a universal gate, any arbitrary Boolean function can be represented using it. Some works like [63] also explore other gates like multi-input OR-inverter graphs for mapping. The main goal of the synthesis step is to reduce the size of the netlist since each operation consumes time and energy. To address this, most works apply traditional logic synthesis techniques like rewriting [64] and resubstitution [65] from tools like ABC [54] and Mockturtle [66]. MASIM [67] uses the optimization sequence command *resyn2* from ABC [54] to reduce the size of the netlist, while other works such as [20] modify the MIG optimization algorithm to reduce the operations using multiple complemented inputs, which requires more clock cycles and RRAM cells. Works like [68] also explore HDL-centric synthesis and verification for RRAM crossbars.

Once the synthesis of the target function to supported operations is done, the scheduling step determines the instruction sequence of the operations to correctly map the function as instructions to hardware. The majority of works [18], [20], [67], [69]–[71] in this regard follow a *Single Instruction Multiple Data* (SIMD) setup where the function computation happens in one row (or column) and each column (or row) computes the same function in parallel. This style of computing achieves a high throughput, but each of the operations in the function must be executed in serial. The goal of the scheduling step is to get an efficient schedule since different instruction sequences (schedules) may have different performance efficiency when they are implemented on hardware. The performance efficiency in scheduling is measured in terms of the memory footprint of the proposed instruction sequence, i.e., minimization of the number of cells required in the computation process. Works like

OptiSIMPLER [69] formulate this as a SAT problem and use an *Satisfiability Modulo Theory (SMT)* solver to generate the optimal instruction sequence. Since SMT does not scale well, this approach is limited to smaller netlists. Scheduling algorithms in works like [20], [70], [71] use heuristic-based approaches to support larger netlists. [20] uses a priority-based algorithm that determines the highest-priority ready operation to be executed based on the number of fan-ins that can be removed after the operation. STAR [70] uses Strahler number [72] to obtain an optimistic estimate of the memory footprint required to accommodate the fan-ins and uses this estimate to guide the scheduling of stateful logic operations toward improved memory utilization. [71] partitions the larger netlist into several small netlists and obtains their optimal schedules, which are then combined to get the result for the larger netlist. These works focus on computing with a single RRAM array. However, the use of multiple arrays may be needed when the memory footprint exceeds the number of available cells in the array. In these cases, the energy consuming cross array copy instruction must be reduced while preparing the instruction schedule. MASIM [67] addresses this through a priority-based scheduling algorithm that uses the number of copy instruction insertions before the current operation and the estimate of future copy instruction insertions as metrics. Several works also target the Non-SIMD setup. In this setup, operations in the netlist can be performed in parallel. The main goal of the scheduling algorithms in this setup is to reduce the latency for computing the target function on one input pattern. Works like [73]–[75] propose algorithms for scheduling in the Non-SIMD setup. SIMPLE [73] formulates the problem as an optimization problem, while Yadav [74] and SAID [75] use heuristics to map larger netlists into 2D RRAM crossbars.

During the synthesis and scheduling steps, the original design specification undergoes several transformations to be ultimately mapped into the RRAM crossbar for CIM. The correctness of these transformations need to be verified at the mapping and the circuit level. One of the initial works [76] focused on generating correct mapping for *Very Long Instruction Word (VLIW)* based on the ReVAMP architecture and used the ABC [54] tool to guarantee correctness. Later, several works [77]–[81] also focused solely on verification using SAT- and *Decision Diagram (DD)*-based methodologies since popular mapping frameworks like SIMPLER MAGIC [18] do not guarantee correctness. One of the first works [77] showed the correctness of mapped micro-operations for Majority-based designs. Initially, clauses are generated from the golden reference MIG. The authors propose a data structure called *ReRam Sequence Graph (ReSg)* to generate clauses from the mapped micro-operations. These two sets of clauses are then checked for equivalence using the Z3 Solver [82]. DD based verification with a similar goal was also explored in [78], which consumed significantly less time for verification compared to the Z3-based method. VeriSIMPLER [80] proposed a verification methodology for SIMPLER MAGIC mapping based on clause generation from golden Verilog and the generated mapping. The proposed method successfully identified a critical bug in the SIMPLER MAGIC mapping methodology that addressed the cases when the input and output were directly connected in the golden design. In [79] the authors proposed a multi-input NOR-based mapping for the MAGIC design style and used a *ac sat-* based methodology for verification. For the verification of automatically generated SPICE netlists from tools like MEMSPICE [83], VeriSIM [81] proposed a clause equivalence-based approach comparing clauses generated from *Piece-Wise Linear (PWL)* voltage files against clauses generated from the golden Verilog.

While RRAM-based CIM has been studied extensively for the

verification of design transformations and mappings, the formal verification of RRAM-based CIM controllers remains largely unexplored. In contrast, controller verification has received attention in DRAM-based CIM systems. Thus, developing formal verification techniques for RRAM-based CIM controllers remains an open research problem.

## V. CROSS TECHNOLOGY DESIGN AUTOMATION AND VERIFICATION

The survey of SRAM-, DRAM-, and RRAM-based CIM technologies reveals that despite considerable differences at the device and circuit level, the design automation flows exhibit commonality. In all three technologies, a high-level specification must be transformed into a set of operations that are supported by the target memory technology. This transformation is then followed by scheduling or mapping that determines how the operations are executed in the memory. Design space exploration for optimization is then performed according to technology-dependent objectives like area, power, delay, throughput, data movement, or memory footprint. Thus, while memory-level operations and objectives differ, synthesis, scheduling, and optimization represent stages across CIM technologies.

A similar pattern can be observed in verification. The fundamental verification problem is to guarantee that the generated CIM implementation preserves the functionality of the original CIM specification. The CIM implementation varies across memory technologies. SRAM approaches generate CIM macros, DRAM approaches generate microprograms or operation sequences, RRAM approaches generate mappings, micro-operations, or circuit-level implementations. While existing methods show SAT, SMT, and DD-based approaches for verification for some technologies, the different representations make a unified approach for verification difficult. We therefore call for a higher-level intermediate representation that can be generated through synthesis methodologies of all memory technologies, making a unified verification approach feasible.

These observations motivate a technology-independent design automation flow in which common abstractions and verification mechanisms are combined with memory technology-specific objectives. Such a framework can accept a technology-independent functional specification and perform technology-specific synthesis and scheduling while maintaining a common verification interface. Developing such abstractions with standard benchmarks and evaluation criteria can enable reuse of design automation and verification technologies across emerging CIM technologies.

## VI. CONCLUSION AND FUTURE WORKS

CIM has evolved from device and circuit level demonstrations into automated architectures encompassing DRAM, SRAM, and RRAM technologies. Recent works has introduced automated synthesis, mapping, and design space exploration techniques that enable more scalable CIM implementations. However, our survey shows that these approaches, despite targeting substantially different underlying technologies, address several common higher-level problems: transforming a functional specification into supported operations, optimizing their execution, mapping them to the target memory structure, and verifying the correctness of the resulting implementation.

This cross-technology perspective calls for an opportunity to move beyond technology-specific workflows toward technology-independent, reusable methods. In particular, common intermediate representations, interfaces between synthesis and technology-specific objectives, and scalable verification approaches could enable design automation to be transferred across different CIM technologies.

Future research directions should therefore investigate unified synthesis and verification frameworks that preserve technology specific constraints while providing common abstraction at the higher levels. This would help develop more systematic and reusable design flows for CIM systems.

## VII. ACKNOWLEDGEMENTS

This work was supported by the European Horizon MSCA Doctoral Network REACT “Self-AwaRe NEuromorphic ArChiTectures: Security, Reliability and Energy-Efficiency” under Grant No. 101226463. Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.



Funded by  
the European Union

## REFERENCES

- [1] C. C. Liu *et al.*, “Bridging the processor-memory performance gap with 3d ic technology,” *IEEE Design & Test of Computers*, vol. 22, no. 6, pp. 556–564, 2005.
- [2] S. Mittal, “A survey of recent prefetching techniques for processor caches,” *ACM Computing Surveys (CSUR)*, vol. 49, no. 2, pp. 1–35, 2016.
- [3] A. J. Smith, “Cache memories,” *ACM Computing Surveys (CSUR)*, vol. 14, no. 3, pp. 473–530, 1982.
- [4] M. Horowitz, “1.1 computing’s energy problem (and what we can do about it),” in *2014 IEEE international solid-state circuits conference digest of technical papers (ISSCC)*. Ieee, 2014, pp. 10–14.
- [5] A. Sebastian *et al.*, “Memory devices and applications for in-memory computing,” *Nature nanotechnology*, vol. 15, no. 7, pp. 529–544, 2020.
- [6] A. Ankit *et al.*, “Circuits and architectures for in-memory computing-based machine learning accelerators,” *IEEE Micro*, vol. 40, no. 6, pp. 8–22, 2020.
- [7] P. Mannocci *et al.*, “In-memory computing with emerging memory devices: Status and outlook,” *APL Machine Learning*, vol. 1, no. 1, 2023.
- [8] Z. Chen *et al.*, “Pico-ram: A pvt-insensitive analog compute-in-memory sram macro with in situ multi-bit charge computing and 6t thin-cell-compatible layout,” *IEEE Journal of Solid-State Circuits*, vol. 60, no. 1, pp. 308–320, 2025.
- [9] O. Mutlu *et al.*, “Processing data where it makes sense: Enabling in-memory computation,” *Microprocessors and Microsystems*, vol. 67, pp. 28–41, 2019. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0141933118302291>
- [10] O. Mutlu *et al.*, “Memory-centric computing: Recent advances in processing-in-dram (invited),” in *2024 IEEE International Electron Devices Meeting (IEDM)*, 2024, pp. 1–4.
- [11] A. Chen *et al.*, *Emerging nanoelectronic devices*. John Wiley & Sons, 2014.
- [12] Z. Sun *et al.*, “A full spectrum of computing-in-memory technologies,” *Nature Electronics*, vol. 6, no. 11, pp. 823–835, 2023.
- [13] M. He *et al.*, “Newton: A dram-maker’s accelerator-in-memory (aim) architecture for machine learning,” in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 372–385.
- [14] G. Singh *et al.*, “A dram-based near-memory architecture for accelerated and energy-efficient execution of transformers,” in *Proceedings of the Great Lakes Symposium on VLSI 2024*, ser. GLSVLSI ’24. Association for Computing Machinery, 2024, p. 57–62.
- [15] S. Han *et al.*, “Near-memory llm inference processor based on 3d dram-to-logic hybrid bonding,” in *2025 62nd ACM/IEEE Design Automation Conference (DAC)*, 2025, pp. 1–7.
- [16] X. Chen *et al.*, “Rram-based analog in-memory computing,” in *2021 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*. IEEE, 2021, pp. 1–6.
- [17] E. P.-B. Quesada *et al.*, “Experimental assessment of multilevel rram-based vector-matrix multiplication operations for in-memory computing,” *IEEE Transactions on Electron Devices*, vol. 70, no. 4, pp. 2009–2014, 2023.
- [18] R. Ben-Hur *et al.*, “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.
- [19] S. Kvatinisky *et al.*, “Magic—memristor-aided logic,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 11, pp. 895–899, 2014.
- [20] M. Soeken *et al.*, “An mig-based compiler for programmable logic-in-memory architectures,” in *Proceedings of the 53rd Annual Design Automation Conference*, 2016, pp. 1–6.
- [21] S. Gupta *et al.*, “Felix: Fast and energy-efficient logic in memory,” in *Proceedings of the International Conference on Computer-Aided Design*, 2018, pp. 1–7.
- [22] C.-J. Jhang *et al.*, “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] Y. Zhang *et al.*, “Recryptor: A reconfigurable cryptographic cortex-m0 processor with in-memory and near-memory computing for iot security,” *IEEE Journal of Solid-State Circuits*, vol. 53, no. 4, pp. 995–1005, 2018.
- [24] W. Lu *et al.*, “A 28-nm 16-kb sram computing-in-memory macro with dual bitline computing for boolean logic operation and bwn mac operation,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 34, no. 5, pp. 1591–1603, 2026.
- [25] J.-W. Su *et al.*, “Two-way transpose multitbit 6t sram computing-in-memory macro for inference-training ai edge chips,” *IEEE Journal of Solid-State Circuits*, vol. 57, no. 2, pp. 609–624, 2022.
- [26] S. Okumura *et al.*, “A ternary based bit scalable, 8.80 tops/w cnn accelerator with many-core processing-in-memory architecture with 896k synapses/mm<sup>2</sup>,” in *2019 Symposium on VLSI Technology*, 2019, pp. C248–C249.
- [27] A. El Arrassi *et al.*, “Dream-cim: A digital sram-based cim accelerator for energy- and area-efficient edge ai,” *IEEE Transactions on Circuits and Systems for Artificial Intelligence*, vol. 2, no. 3, pp. 211–221, 2025.
- [28] J. Chen *et al.*, “Autodcim: An automated digital cim compiler,” in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.
- [29] H. Zhang *et al.*, “Arctic: Agile and robust compute-in-memory compiler with parameterized int/fp precision and built-in self test,” in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2024, pp. 1–6.
- [30] K. Shao *et al.*, “Syndcim: A performance-aware digital computing-in-memory compiler with multi-spec-oriented subcircuit synthesis,” in *2025 Design, Automation & Test in Europe Conference (DATE)*, 2025, pp. 1–7.
- [31] H. Diao *et al.*, “Sega-dcim: Design space exploration-guided automatic digital cim compiler with multiple precision support,” in *2025 Design, Automation & Test in Europe Conference (DATE)*. IEEE, 2025, pp. 1–7.
- [32] C. Wang *et al.*, “Damil-dcim: A digital cim layout synthesis framework with dataflow-aware floorplan and milp-based detailed placement,” in *2025 Design, Automation & Test in Europe Conference (DATE)*, 2025, pp. 1–7.
- [33] T. Dong *et al.*, “Openc2: An open-source end-to-end hardware compiler development framework for digital compute-in-memory macro,” in *2025 Design, Automation & Test in Europe Conference (DATE)*, 2025, pp. 1–2.
- [34] Y. Zhou *et al.*, “Openacm: An open-source sram-based approximate cim compiler,” in *2026 Design, Automation & Test in Europe Conference (DATE)*. IEEE, 2026, pp. 1–7.
- [35] T.-L. Tsai *et al.*, “Testing of in-memory-computing memories with 8 t srams,” *Microelectronics Reliability*, vol. 123, p. 114215, 2021.
- [36] G. F. Oliveira *et al.*, “Accelerating neural network inference with processing-in-dram: From the edge to the cloud,” *IEEE Micro*, vol. 42, no. 6, p. 25–38, Nov. 2022.
- [37] B. Kim *et al.*, “Mvid: Sparse matrix-vector multiplication in mobile dram for accelerating recurrent neural networks,” *IEEE Transactions on Computers*, vol. 69, no. 7, pp. 955–967, 2020.
- [38] D. Niu *et al.*, “184qps/w 64mb/mm<sup>2</sup> 3d logic-to-dram hybrid bonding with process-near-memory engine for recommendation system,” in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65, 2022, pp. 1–3.
- [39] J. Kang *et al.*, “Spdram: Efficient in-dram acceleration of sparse matrix-vector multiplication,” *IEEE Access*, vol. 12, pp. 176 009–176 021, 2024.
- [40] H. Asghari-Moghaddam *et al.*, “Chameleon: Versatile and practical near-dram acceleration architecture for large memory systems,” in *2016*

49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2016, pp. 1–13.

- [41] Y. Zhao *et al.*, “Um-pim: Dram-based pim with uniform & shared memory space,” in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024, pp. 644–659.
- [42] J. Wang *et al.*, “Nicepim: Design space exploration for processing-in-memory dnn accelerators with 3-d stacked-dram,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 5, pp. 1456–1469, 2024.
- [43] V. Seshadri *et al.*, “Rowclone: Fast and energy-efficient in-dram bulk data copy and initialization,” in *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2013, pp. 185–197.
- [44] V. Seshadri *et al.*, “Ambit: In-memory accelerator for bulk bitwise operations using commodity dram technology,” in *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2017, pp. 273–287.
- [45] F. Gao *et al.*, “Computedram: In-memory compute using off-the-shelf drams,” in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-52. Association for Computing Machinery, 2019, p. 100–113.
- [46] F. Gao *et al.*, “Fracdram: Fractional values in off-the-shelf dram,” in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022, pp. 885–899.
- [47] X. Xin *et al.*, “Elp2im: Efficient and low power bitwise operation processing in dram,” in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 303–314.
- [48] N. Hajinazar *et al.*, “Simdram: a framework for bit-serial simd processing using dram,” in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS ’21. Association for Computing Machinery, 2021, p. 329–345.
- [49] G. F. Oliveira *et al.*, “Mimdram: An end-to-end processing-using-dram system for high-throughput, energy-efficient and programmer-transparent multiple-instruction multiple-data computing,” in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2024, pp. 186–203.
- [50] İ. E. Yüksel *et al.*, “Functionally-Complete Boolean Logic in Real DRAM Chips: Experimental Characterization and Analysis,” in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2024, pp. 280–296.
- [51] X. Peng *et al.*, “CHOPPER: A Compiler Infrastructure for Programmable Bit-serial SIMD Processing Using Memory in DRAM,” in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE Computer Society, 2023, pp. 1275–1288.
- [52] D. Guo *et al.*, “Pimsynth: A unified compiler framework for bit-serial processing-in-memory architectures,” *IEEE Computer Architecture Letters*, vol. 24, no. 2, pp. 277–280, 2025.
- [53] C. Wolf *et al.*, “Yosys - a free verilog synthesis suite,” in *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*, 2013.
- [54] R. Brayton *et al.*, “Abc: An academic industrial-strength verification tool,” in *International Conference on Computer Aided Verification*. Springer, 2010, pp. 24–40.
- [55] C. Lattner *et al.*, “Llvm: a compilation framework for lifelong program analysis & transformation,” in *International Symposium on Code Generation and Optimization*, 2004. CGO 2004., 2004, pp. 75–86.
- [56] F. A. Siddique *et al.*, “Architectural modeling and benchmarking for digital dram pim,” in *2024 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2024, pp. 247–261.
- [57] D. Sahoo *et al.*, “Formal modeling and verification of controllers for a family of dram caches,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2485–2496, 2018.
- [58] L. Steiner *et al.*, “A framework for formal verification of dram controllers,” in *Proceedings of the 2022 International Symposium on Memory Systems*, ser. MEMSYS ’22. Association for Computing Machinery, 2023.
- [59] A. Bende *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.
- [60] L. Brackmann *et al.*, “Experimental verification and evaluation of non-stateful logic gates in resistive ram,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 72, no. 5, pp. 2029–2038, 2024.
- [61] B. Hoffer *et al.*, “Experimental demonstration of memristor-aided logic (magic) using valence change memory (vcm),” *IEEE Transactions on Electron Devices*, vol. 67, no. 8, pp. 3115–3122, 2020.
- [62] A. Bende *et al.*, “Device-to-logic variability propagation in rram-based logic-in-memory architectures,” *Neuromorphic computing and engineering*, vol. 6, no. 2, p. 024009, 2026.
- [63] F. Shirinzadeh *et al.*, “Fan-in aware graph-based optimization for mac-based in-memory computing,” in *2026 IEEE 29th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*. IEEE, 2026, pp. 1–5.
- [64] H. Riener *et al.*, “On-the-fly and dag-aware: Rewriting boolean networks with exact synthesis,” in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 1649–1654.
- [65] S.-Y. Lee *et al.*, “A simulation-guided paradigm for logic synthesis and verification,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 8, pp. 2573–2586, 2021.
- [66] M. Soeken *et al.*, “The epfl logic synthesis libraries,” *arXiv preprint arXiv:1805.05121*, 2018.
- [67] X. Qian *et al.*, “Masim: An energy-efficient multi-array scheduler for simd logic-in-memory architectures,” in *2025 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2025, pp. 1–9.
- [68] F. Shirinzadeh *et al.*, “An hdl-centric synthesis and verification approach for rram-based neuromorphic systems,” *Microprocessors and Microsystems*, p. 105278, 2026.
- [69] D. Bhattacharjee *et al.*, “Synthesis and technology mapping for in-memory computing,” in *Emerging Computing: From Devices to Systems: Looking Beyond Moore and Von Neumann*. Springer, 2022, pp. 317–353.
- [70] F. Wang *et al.*, “Star: Synthesis of stateful logic in rram targeting high area utilization,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 5, pp. 864–877, 2020.
- [71] X. Qian *et al.*, “A recursive partition-based in-memory simd computation scheduler for memory footprint minimization,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 44, no. 6, pp. 2105–2118, 2024.
- [72] D. Auber, “Using strahler numbers for real time visual exploration of huge graphs,” in *International Conference on Computer Vision and Graphics*, vol. 1, 2002, p. 3.
- [73] R. B. Hur *et al.*, “Simple magic: Synthesis and in-memory mapping of logic execution for memristor-aided logic,” in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 225–232.
- [74] D. N. Yadav *et al.*, “Look-ahead mapping of boolean functions in memristive crossbar array,” *Integration*, vol. 64, pp. 152–162, 2019.
- [75] V. Tenace *et al.*, “Said: A supergate-aided logic synthesis flow for memristive crossbars,” in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 372–377.
- [76] D. Bhattacharjee *et al.*, “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.
- [77] K. Bhunia *et al.*, “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.
- [78] K. Qayyum *et al.*, “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.
- [79] S. Nabipour *et al.*, “Multi-input magic synthesis and verification for in-memory computing design,” in *ISMVL*, 2025, pp. 178–183.
- [80] C. K. Jha *et al.*, “verisimpler: An automated formal verification methodology for simpler magic design style based in-memory computing,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 71, no. 9, pp. 4169–4179, 2024.
- [81] C. K. Jha *et al.*, “verisim: Formal verification of spice netlists for magic-based logic-in-memory,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 45, no. 2, pp. 845–854, 2025.
- [82] L. De Moura *et al.*, “Z3: An efficient smt solver,” in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.
- [83] S. Singh *et al.*, “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.