

Fast and Compact Unary Sorting in Memristive Memory

Fabian Seiler[†], Nima Taherinejad[†], and M. Hassan Najafi⁺

[†]ECLECTX Team, Heidelberg University, Heidelberg, Germany

⁺Case Western Reserve University, Cleveland, USA

{fabian.seiler, nima.taherinejad}@ziti.uni-heidelberg.de, najafi@case.edu

Abstract—Sorting is a fundamental operation in many data-intensive applications, yet conventional architectures are limited by the cost of data movement between memory and processing units. In-Memory Computing (IMC) mitigates this bottleneck by executing logic directly within memory arrays. However, emerging memory devices exhibit significant variability and switching non-idealities, motivating the use of unconventional paradigms such as unary (stochastic) computing, which offers inherent fault tolerance. Recent work has demonstrated unary sorting in memristive memory using MAGIC NOR and NOT operations. This approach incurs logic decomposition overhead and imposes restrictive device-level requirements. This work proposes an improved memristive unary sorting architecture based on FELIX OR and NAND stateful logic primitives. By directly mapping unary maximum and minimum operations to OR- and NAND-based realizations, the proposed Compare-and-Swap (CAS) unit reduces logic depth, temporary storage, and initialization overhead. SPICE simulations demonstrate reductions of 28.6% in latency, 27.5% in energy consumption, and 20% in area compared to the State-of-the-Art (SoA) unary sorting architecture. Application-level evaluation using a 3×3 median filter further demonstrates improved robustness under increasing operation failure rates, while comparable binary sorting approaches exhibit significantly degraded output quality due to their higher sensitivity to device-level errors. These results highlight the potential of combining unary computing with richer stateful logic primitives for efficient and reliable memristive IMC systems.

I. INTRODUCTION

Sorting is a fundamental operation in many data-intensive applications and often dominates their execution time and energy consumption. Although hardware accelerators can improve sorting throughput through parallelism, conventional architectures remain constrained by the von Neumann model, where data must be moved repeatedly between memory and processing units. As a result, the overall cost of sorting is dominated by data movement rather than by the sorting computation itself [1]–[5]. Fig. 1(a) illustrates this bottleneck in conventional sorting architectures. Memristive In-Memory Computing (IMC) addresses the data movement challenge by enabling computation directly inside memory arrays using stateful logic operations [6]–[9]. In particular, memristive crossbars allow Compare-and-Swap (CAS) operation, which is fundamental to sorting architectures, to be executed within memory, eliminating costly data transfers and enabling highly parallel sorting [4], as illustrated in Fig. 1(b). However, practical memristive devices remain susceptible to variability, switching failures, and soft errors [10]–[13]. These non-idealities can significantly affect conventional binary radix-based stateful logic implementations and motivate unconven-

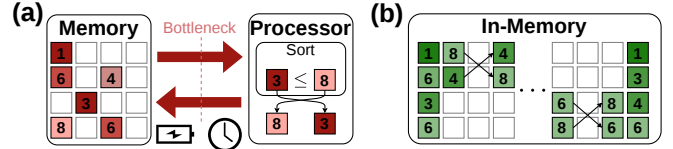


Fig. 1: Sorting architectures: (a) data movement cost in conventional architectures (b) in-memory sorting architecture.

tional paradigms such as Unary Computing (UC) [14]–[16], which provide inherent fault tolerance.

Recent work has demonstrated unary sorting in memristive memory by exploiting the correspondence between minimum and maximum functions and simple logical operations on unary bit-streams [4]. Fig. 2 illustrates the underlying stateful logic operations and their unary equivalent. Compared to conventional binary, UC improves robustness against device-level faults, since individual bit errors affect the represented value uniformly rather than causing large positional errors [17]. The State-of-the-Art (SoA) unary sorting architecture relies on MAGIC [7] NOR and NOT operations to implement unary CAS units [4]. While functionally complete, this approach requires multiple logic decompositions and imposes restrictive device requirements, increasing latency, area overhead, and implementation complexity [18], [19]. Specifically, minimum and maximum functions must be synthesized through several NOR and NOT operations, increasing the number of required state transitions and temporary storage locations.

In this work, we revisit unary in-memory sorting using the extended stateful logic primitives provided by FELIX [8]. By exploiting its native OR and NAND operations, we derive a more efficient unary CAS architecture that reduces logic depth, temporary storage requirements, and initialization overhead while relaxing device constraints. The proposed design remains fully compatible with unary sorting networks and preserves the inherent robustness of UC. The main contributions of this work are summarized as follows:

- We propose a mixed-logic unary CAS architecture that exploits FELIX OR and NAND operations to directly implement unary minimum and maximum functions, eliminating costly NOR and NOT decompositions.
- We develop an improved unary in-memory sorting architecture that reduces latency, energy consumption, and area by 28.6%, 27.5%, and 20%, respectively, compared to the SoA MAGIC-based unary sorting design.
- We evaluate the proposed method using circuit-level SPICE simulations and analyze its latency, energy, area,

and reliability under realistic device assumptions.

- With a cycle-accurate simulator, we demonstrate the effectiveness of the proposed design in a median-filtering application and compare its robustness against both unary MAGIC-based and binary sorting architectures under device failures.

The rest of the paper is organized as follows. Section II reviews related work and necessary background, and Section III presents the proposed method. Sections IV and V cover the circuit- and application-level evaluations, respectively. Section VI concludes the paper.

II. BACKGROUND AND RELATED WORK

A. Memristive In-Memory Computing and Stateful Logic

The growing performance and energy cost of data movement between memory and processing units has motivated the development of IMC architectures, where computations are performed directly at the location of the data [1], [2], [9], [20], [21]. Among the various IMC technologies, memristive memories have attracted considerable interest due to their non-volatility, high density, low standby power, and compatibility with crossbar-based architectures [6], [12], [22], [23]. In memristive crossbars, data are stored as resistive states, Low Resistance State (LRS) and High Resistance State (HRS), and can be processed directly inside the memory array. This significantly reduces communication overhead while enabling massive parallelism. A prominent approach for memristive IMC is stateful logic, in which logical values are represented by memristor resistance states and Boolean operations are executed directly within the memory array [6]–[9]. MAGIC is one of the most widely adopted stateful logic families. It realizes computation through NOR and NOT operations [7] through conditional switching of the output from LRS→HRS if all inputs are ‘0’. Although functionally complete, complex functions often require multiple NOR-based decompositions, increasing latency, temporary storage requirements, and write activity. FELIX [8] builds upon MAGIC and introduces native OR and NAND circuits (see Fig. 2a-b). As the minimum and maximum functions in unary sorting map naturally to AND and OR operations, FELIX provides a promising foundation for more efficient unary in-memory sorting architectures.

B. Unary Computing and Sorting

UC is an alternative to conventional binary, offering simple, noise-tolerant solutions for complex arithmetic operations.

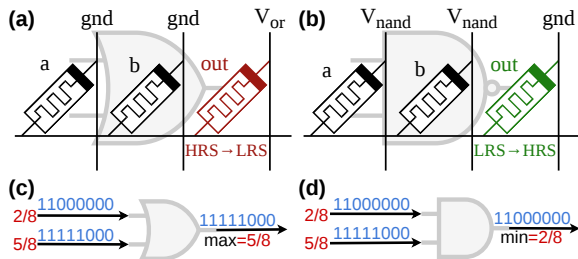


Fig. 2: Stateful logic gates for (a) OR and (b) NAND [8]. Equivalent unary bit-stream functions for (c)-d) OR and AND.

These properties make UC particularly attractive for emerging memristive IMC [4], [24]. Unlike weighted binary radix, UC weights all digits equally. Numbers are encoded uniformly by a sequence of one value (e.g., 1) followed by a sequence of the other value (e.g., 0), forming a stream of 1’s and 0’s known as a unary bit-stream. The value of a unary bit-stream is determined by the frequency of 1’s within it. For example, 11000000 is a unary bit-stream representing 2/8 or 1/4. In the unary domain, the minimum and maximum functions correspond to bitwise AND and OR operations, respectively [14]. As a result, CAS operations can be implemented efficiently inside or near memory arrays using stateful logic primitives [4].

Sorting has been extensively studied in both software and hardware domains. While software algorithms such as QuickSort and MergeSort remain widely used, hardware sorting networks such as Batcher or bitonic networks provide highly parallel implementations suitable for acceleration [5], [25]–[27]. Earlier memristive sorting networks realized *min* and *max* using complementary resistive switching structures [28]–[30]. These approaches map input voltages to an output voltage ($V \rightarrow V$) using memristive circuit elements, contrary to stateful logic where operands and results are memristive states ($R \rightarrow R$), so data never leaves the crossbar. Recent work has extended sorting to memristive IMC, demonstrating binary near-memory approaches [31], [32] and both binary and unary in-memory sorting architectures [4]. Particularly, the unary sorting framework in [4] exploits the correspondence between *min* or *max* functions and AND or OR operations to realize sorting networks within memristive crossbars, but relies exclusively on MAGIC NOR and NOT logic, requiring multiple steps. This motivates the search for stateful logic primitives that implement unary CAS operations more directly.

III. PROPOSED METHOD

A. Improved Unary Sorting Unit

Unary sorting networks are built from basic CAS units, each computing the minimum and maximum of two inputs represented as unary bit-streams [14]. Let $A = a_i$ and $B = b_i$ denote two unary streams. In UC the minimum and maximum functions are computed bitwise as

$$\min(A, B)_i = a_i \wedge b_i, \quad \max(A, B)_i = a_i \vee b_i. \quad (1)$$

The resulting output streams preserve the unary encoding and can therefore be directly propagated to subsequent stages of the sorting network. The SoA memristive unary sorting architecture presented in [4] realizes these functions exclusively using MAGIC NOR and NOT operations [7]. Since neither AND nor OR are directly available, both functions must be synthesized through logic decompositions, i.e.,

$$x \wedge y = \text{NOR}(\text{NOT}(x), \text{NOT}(y)) \quad (2)$$

$$x \vee y = \text{NOT}(\text{NOR}(x, y)). \quad (3)$$

Although functionally correct, these decompositions introduce additional logic stages, temporary storage locations, and initialization cycles. Furthermore, the underlying MAGIC logic

family requires memristive devices satisfying the condition $|v_{\text{set}}| > 2|v_{\text{reset}}|$ [7], [18], [19], limiting compatibility with several highly scalable memristive technologies, including HfOx-based Valence Change Mechanism (VCM) devices [19]. Consequently, existing in-memory unary sorting architectures suffer from both architectural overhead and restrictive device requirements. To address these limitations, we exploit the native OR and NAND operations provided by FELIX logic [8]. The unary maximum operation directly corresponds to an OR and can be implemented using a single stateful operation, i.e.,

$$\max(A, B)_i = \text{OR}(a_i, b_i). \quad (4)$$

The unary minimum operation is realized using a NAND-based implementation of AND. Instead of using a conventional NOT operation, which would reintroduce the device constraints of MAGIC, we exploit the equivalence $\text{NAND}(x, 1) \equiv \text{NOT}(x)$;

$$\min(A, B)_i = \text{NAND}(\text{NAND}(a_i, b_i), 1_i), \quad (5)$$

where 1_i denotes the i -th element of a pre-initialized unary reference vector $\vec{1}$. Since multiple CAS units are executed throughout a sorting network, this reference vector can be generated once and reused for all subsequent operations, thereby reducing initialization overhead. Compared to the MAGIC-based implementation, the proposed FELIX-based implementation simultaneously reduces logic decomposition overhead and relaxes device requirements. As a result, fewer logic operations, temporary memristors, and initialization cycles are required while enabling compatibility with a broader range of memristive technologies.

B. Algorithmic Implementation

Fig. 3(a) illustrates the sequence of stateful logic operations required to compute the unary minimum and maximum functions ($\bar{\wedge}$ and $\vee$ indicate NAND and OR, $\perp$ and $\top$ indicate SET and RESET, respectively). The state of each memristor row is updated at every cycle according to the applied FELIX operation, while rows without active operations retain their stored information. Fig. 3(b) shows the corresponding parallelized crossbar mapping for unary bit-streams of length $N = 2^n$, where each column is associated with one of the operations shown in the flow diagram. The figure illustrates the logical placement of data and operations rather than the transistor-level crossbar structure. We assume a 1T1R organization, where each memristor is connected to an access transistor that enables selective activation of the cells participating in a stateful logic operation. Algorithm 1 summarizes the proposed unary CAS implementation. For each unary bit position, a NAND operation first computes an intermediate result. A second NAND operation together with the reference vector (pre-initialized before the CAS, $Z \leftarrow \vec{1}$) generates the minimum output stream. The maximum output stream is then computed directly using a single OR operation. Compared to the SoA implementation, we eliminate the expensive NOR and NOT decompositions while reducing the number of required state transitions and temporary storage locations.

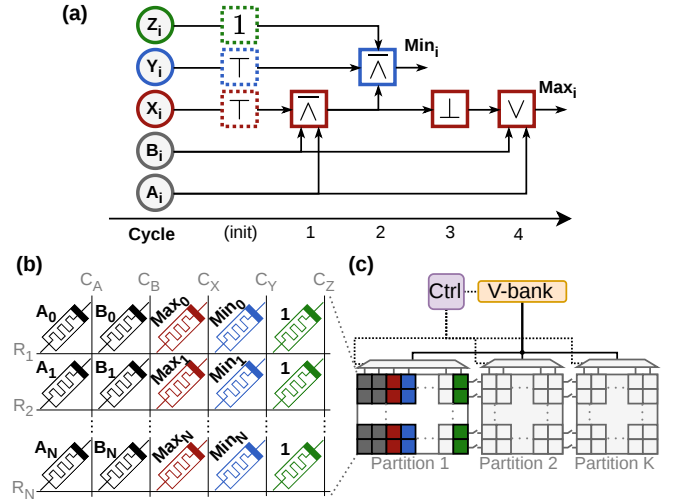


Fig. 3: Proposed CAS unit: (a) sequence of stateful logic operations to compute unary minimum and maximum, (b) parallelized CAS crossbar mapping for unary streams of length $N = 2^n$, and (c) embedding CAS units within a crossbar architecture consisting of K partitions. The columns are connected to the voltage bank via demultiplexers.

Algorithm 1: Proposed Unary Sorting CAS Unit

- 1 **Inputs:** Crossbar $C_{n,m}$, unary streams A and B , pre-initialized reference vector $\vec{1}$ stored in Z .
- 2 **for all** $i \in [1, n]$ **do in parallel**
- 3 Initialize $X_i \leftarrow 1, Y_i \leftarrow 1$;
- 4 $X_i \leftarrow \text{NAND}(A_i, B_i)$
- 5 $Y_i \leftarrow \text{NAND}(X_i, Z_i)$
- 6 Re-initialize $X_i \leftarrow 0$
- 7 $X_i \leftarrow \text{OR}(A_i, B_i)$
- 8 **end**
- 9 **return:** $C_{n,m}$ with $Y = \min(A, B)$ and $X = \max(A, B)$

C. Complete Sorting System

Following [4], we employ a Batcher sorting network consisting of multiple stages of parallel CAS operations. During each stage, pairs of unary operands are compared and exchanged until a globally sorted sequence is obtained (see Section V for more details). Parallelized CAS operations are enabled by partitions that can be connected via switches, as shown in Figure 3(c). Up to three columns of each partition can be individually connected to a level of the reference voltage bank using demultiplexers controlled by a state machine in the control unit. Since the proposed CAS preserves the unary minimum and maximum semantics, the sorting-network structure and functionality remain unchanged. The core contribution of this work lies in replacing the underlying MAGIC-based CAS with the proposed FELIX-based implementation. The reduced logic and initialization requirements of each CAS propagate across the sorting network, reducing the execution time of its sequential stages and the overall energy and memristor requirements. Importantly, the inherent robustness of unary encoding is preserved, while the reduced number of stateful

TABLE I: Circuit-level control voltages ($t_{\text{cyc}} = 10\mu\text{s}$).

Operation	OR	NAND	NOR	NOT	SET	RESET
Voltage	0.55V	0.35V	0.45V	0.45V	0.4V	-0.4V

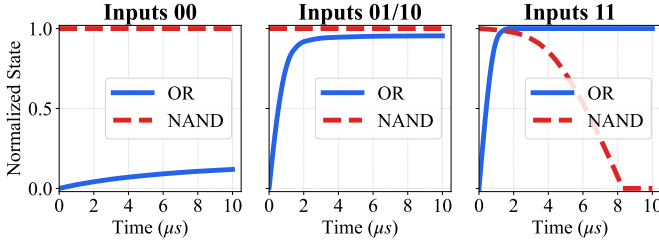


Fig. 4: Circuit-level simulations of FELIX OR and NAND operations using the realistic SDC memristor model from [10].

logic operations provides an additional reliability benefit over the MAGIC-based unary implementation.

IV. CIRCUIT-LEVEL METRICS AND COMPARISON

A. Circuit-Level Simulation

To evaluate the proposed architecture, we perform SPICE simulations of the individual stateful logic operations using the SDC memristor model from [10], since it is compatible with both MAGIC and FELIX operations and hence enables a fair apple-to-apple comparison. We employ the nominal parameter set of the model for the energy characterization. Unlike the idealized threshold-based device assumptions used in [4], the SDC model is fitted to experimentally characterized Resistive RAM (RRAM) devices and captures state-dependent switching and resistance evolution. The surrounding Complementary Metal-Oxide Semiconductor (CMOS) circuitry for selection is represented by ideal switches. Hence, the simulations characterize the memristive logic operations rather than a complete transistor- and parasitic-aware crossbar implementation. For a fair comparison, MAGIC and FELIX operations are evaluated using the same conditions, including the memristor model, switching assumptions, and fixed cycle time of $t_{\text{cyc}} = 10\mu\text{s}$. The applied control voltages are summarized in Table I.

Fig. 4 illustrates the resulting switching behavior of the FELIX OR and NAND operations. The energy of each logic operation is obtained from the corresponding SPICE simulation and averaged over all valid input combinations, since the dissipated energy depends on the initial resistance states of the participating devices. The resulting values, summarized in Table II, therefore provide a common device-level basis for comparing the two logic implementations. As switching characteristics and operating voltages depend strongly on the employed memristive technology, these values should be interpreted as technology-dependent estimates rather than absolute full-system energy costs.

B. Energy and Resource Analysis

Using the operation counts derived in Section III, the total energy of a unary CAS unit is estimated from the mean energy of its constituent logic and initialization operations, i.e.,

$$E_{\text{MAGIC}} = 3E_{\text{NOT}} + 2E_{\text{NOR}} + 5E_{\text{SET}} \quad (6)$$

$$E_{\text{Proposed}} = 2E_{\text{NAND}} + E_{\text{OR}} + 3E_{\text{SET}} + E_{\text{RESET}} \quad (7)$$

TABLE II: Energy consumption per logic operation [pJ].

Logic	00	01	10	11	Mean
OR(a,b)	11.75	77.47	77.47	133.22	74.98
NAND(a,b)	11.79	45.68	45.68	25.60	32.19
NOR(a,b)	25.00	26.68	26.68	15.26	23.41
NOT(b)	21.97	38.22	–	–	30.10
SET(b)	44.83	115.04	–	–	79.94
RESET(b)	8.89	11.64	–	–	10.27

Although an individual NOR operation consumes less energy than an OR operation (23.41 pJ versus 74.98 pJ), the SoA implementation requires a larger number of logic operations and SET initializations. Since initialization dominates the overall energy consumption in MAGIC designs [33], the total energy of the SoA CAS unit increases significantly. The proposed design further benefits from replacing two SET operations with a single RESET operation. Since RESET is performed while the device resides in its high-resistance state, the resulting current and switching energy are significantly lower than for SET operations. Consequently, the proposed FELIX-based CAS reduces the energy consumption from 536.79 pJ to 389.43 pJ per unary bit, corresponding to a reduction of 27.45%. To provide a fair baseline for comparison, the reported energy includes the initialization of the reusable reference vector $\mathbf{1}$. In practical sorting networks this vector is generated only once and reused across multiple CAS operations, resulting in additional energy savings beyond the reported baseline comparison. Table III summarizes the circuit-level resource requirements of the proposed unary CAS unit. Since all unary bits are processed in parallel, the execution latency remains constant regardless of bit-stream length. In contrast, the required crossbar area and total energy scale linearly with the unary stream length.

The reported metrics cover the memristive cells and stateful logic operations of the CAS, but exclude access-transistor, peripheral, and interconnect overheads. Consequently, the reported resource reduction refers to the required memristor count rather than total physical circuit area. Both implementations use the same 1T1R organization and largely share the required control circuitry (see Figure 3). The main additional peripheral requirement of the proposed approach over [4] is support for an additional voltage level to realize both OR and NAND. The impact on total system-level area and energy due to scope limitations is not included in the present evaluation.

C. Comparison with State-of-the-Art

Since prior voltage-domain and read-based memristive sorting approaches are not directly and fairly comparable to stateful unary sorting, for the quantitative comparison we focus on the MAGIC-based unary architecture in [4]. Table IV summarizes the resulting improvements. By exploiting native FELIX OR and NAND operations, the proposed CAS reduces the required logic operations and temporary storage while relaxing the device requirements of MAGIC. Consequently, it reduces the latency and energy per CAS by 28.6% and 27.5%, respectively, and the required memristor count by 20%.

Compared to binary sorting architectures, unary approaches generally require larger memory arrays and higher energy

TABLE III: Circuit-level metrics of the proposed unary CAS unit.

Bitstream Length	Required Dim. ^a	Input Memristors	Computing Memristors	OR Ops	NAND Ops	Init Cycles	Re-init Cycles	Operation Cycles	Energy (nJ)	MED (1% fail/OP)	MRED (1% fail/OP)
2^4	16×4	32	32	16	32	1	1	3	6.2308	14.95	8.52%
2^6	64×4	128	128	64	128	1	1	3	24.9233	8.46	5.25%
2^8	256×4	512	512	256	512	1	1	3	99.6931	5.77	3.93%
2^{10}	1024×4	2048	2048	1024	2048	1	1	3	398.7725	5.03	3.62%
$N = 2^n$	$N \times 4$	$2N$	$2N$	N	$2N$	1	1	3	$0.3894N$	$\mathcal{O}(N^{-0.53}) + 4.53$	$\mathcal{O}(N^{-0.54}) + 3.41\%$

^a The reusable reference vector $\vec{1}$ is shared across multiple CAS units and therefore not included in the reported dimensions.

TABLE IV: Comparison of unary in-memory sorting systems.

Metric	MAGIC-based [4]	Proposed
Native Operations	NOR, NOT	OR, NAND
Device Requirement	$ v_{\text{set}} > 2 v_{\text{reset}} $	None
Overhead Memristors	n	0
Initialization Cycles	2	2
Area per CAS (memristors)	$5n$	$4n$
Latency per CAS (cycles)	7	5
Energy per CAS (pJ)	$536.79n$	$389.43n$

consumption due to the unary representation of values. However, unary CAS operations are significantly simpler and can be executed with lower latency while avoiding complex comparator and multiplexer structures [14]. More importantly, UC provides inherent fault tolerance, which is particularly attractive for memristive IMC systems where device variability, resistance drift, and stochastic switching remain major challenges [11], [34], [35]. The reliability of stateful logic systems is strongly influenced by the number of dependent operations, since each operation introduces an additional probability of failure. In binary representations, a fault affecting bit position i may alter the represented value by 2^i or propagate through comparator and multiplexer circuits, potentially resulting in incorrect sorting decisions. In contrast, a single error in a unary bit-stream of length N changes the represented value by only $1/N$, providing natural error resilience. To quantify the impact of device failures on numerical accuracy, Table III additionally reports the Mean Error Distance (MED) and Median Relative Error Distance (MRED) obtained from 1000 randomized sorting operations assuming a per-operation failure probability of 1%. As expected, increasing unary precision reduces the relative impact of individual failures, resulting in lower MED and MRED values for longer streams. Lastly, the proposed CAS requires fewer cascaded logic operations than the MAGIC-based implementation, reducing the probability of error propagation and improving overall reliability.

V. APPLICATION-LEVEL ANALYSIS

A. Application in Median Filtering

To evaluate the practical impact of the proposed architecture, we consider a 3×3 median filter for image denoising. Median filters are widely used to remove impulse noise, such as salt-and-pepper noise (where randomly selected pixels are replaced by either the minimum or maximum intensity value), while preserving image edges and structural information. The filtering operation is performed by sorting the nine pixel values within each filter window and selecting the median element. Following [4], the median filter is implemented using a Batcher sorting network, as illustrated in Fig. 5.

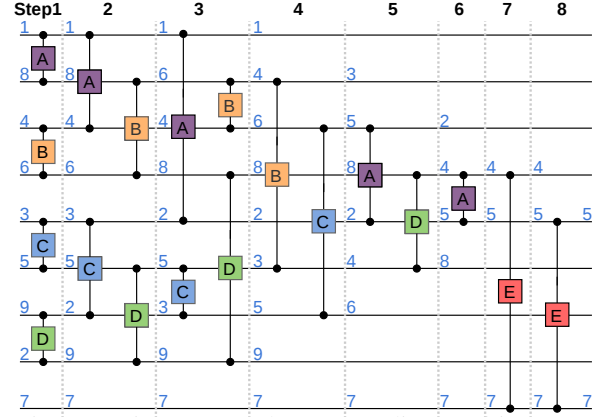


Fig. 5: Processing steps and corresponding partitions (A-E) of the 3×3 median filter based on a Batcher sorting network [4].

The proposed unary CAS units replace the original MAGIC-based implementation without modifying the overall filtering algorithm. Consequently, any improvements achieved at the CAS level translate directly to the complete median filtering application. The sorting network is partitioned into five stages (A–E), allowing all CAS units within a stage to execute in parallel before synchronizing with the subsequent stage. Since operation reliability is strongly device- and operating-condition dependent, we instead perform a parameterized failure-probability sweep to determine the operation reliability required to achieve a given application-level quality independently of a particular device technology. For the considered variability model [10], the resulting MAGIC and FELIX operation failure rates are on the order of 10^{-1} (similar to reliability limitations reported in comparable silicon experiments [11]), at which the application becomes impractical. The results show that the evaluated unary sorting application remains robust up to failure probabilities of approximately 10^{-3} . The sweep therefore provides a technology-independent reliability target for future device and circuit implementations rather than predicting the behavior of a particular technology. To evaluate robustness under device non-idealities, a cycle-accurate simulator is employed in which each stateful logic operation is assigned a configurable failure probability (for this experiment, we assume equal failure rates for all operations). The resulting filtering quality is evaluated using the Peak Signal-to-Noise Ratio (PSNR), and Mean Structural Similarity Index Measure (MSSIM) [36] metrics under varying per-operation failure probabilities. To ensure statistically meaningful results, we use the image dataset from [37], which contains a diverse set of commonly used image-processing benchmarks. Each experiment is repeated 100 times per image to capture the

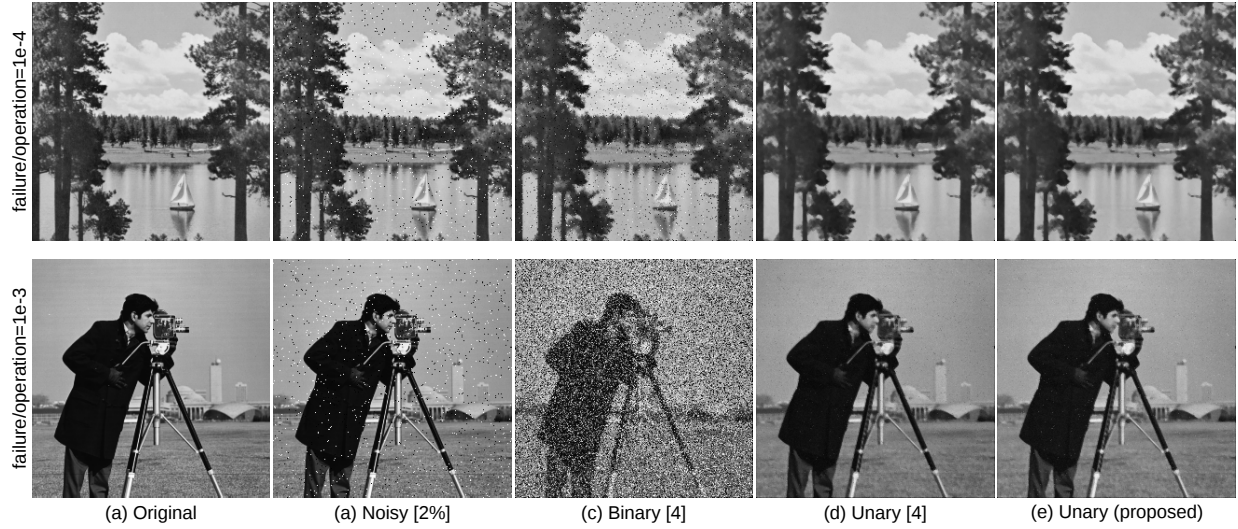


Fig. 6: Exemplary denoising results of the 3×3 median filter under operation failure rates of 10^{-4} (top) and 10^{-3} (bottom).

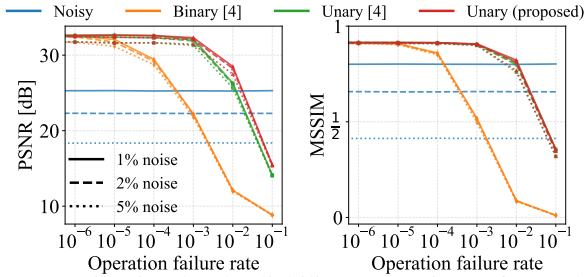


Fig. 7: Application-level reliability comparison of CAS-based median filters under increasing operation failure rates.

impact of stochastic operation failures. Furthermore, the evaluation is repeated at salt-and-pepper noise levels of 1%, 2%, and 5% corrupted pixels to analyze the interaction between device failures and application-level noise. Fig. 6 presents representative denoising examples that illustrate the improved resilience of unary sorting over binary sorting under device-level operation failures. The results are summarized in Fig. 7, which reports the mean PSNR and MSSIM obtained across the entire dataset as a function of per-operation failure rate.

B. Application-Level Comparison and Discussion

Fig. 7 compares the binary and MAGIC-based unary approaches from [4] with the proposed unary sorting architecture under increasing operation failure rates. While binary sorting requires fewer operations overall, its accuracy is highly sensitive to errors in individual weighted bits, particularly within comparator and multiplexer circuits. Unary representations exhibit significantly higher robustness, as each bit contributes equally to the encoded value and a single bit error can affect the represented value by only $1/N$. Moreover, multiple bit errors may cancel each other, limiting their combined effect. Hence, the dominant robustness improvement over binary sorting originates from the unary representation itself and is shared by both unary implementations. Within the unary approaches, the proposed FELIX-based architecture provides an additional reliability improvement by requiring

fewer sequential stateful logic operations per CAS. Since each additional operation introduces another opportunity for failure, the shorter FELIX-based execution achieves higher success rates than the MAGIC-based implementation under the same per-operation failure probability. These results demonstrate that the proposed OR- and NAND-based architecture preserves the inherent robustness of UC while reducing latency, energy consumption, and the required memristor count compared to the MAGIC-based unary implementation. The approach, therefore, represents a promising direction for unary IMC systems based on emerging memristive technologies.

VI. CONCLUSION

This work presented an improved unary memristive in-memory sorting architecture based on FELIX OR and NAND stateful logic operations. By replacing the NOR and NOT decompositions used in prior unary sorting approaches, the proposed CAS unit reduces logic depth, temporary storage requirements, and initialization overhead while relaxing device constraints. Circuit-level simulations demonstrated reductions of 28.6% in latency, 27.5% in energy consumption, and 20% in area compared to the SoA unary sorting architecture. Application-level evaluation using a 3×3 median filter further showed improved robustness under device failures. While unary sorting requires higher area and energy than binary approaches due to the 2^n representation of values, it reduces CAS latency from $\mathcal{O}(n)$ to $\mathcal{O}(1)$ and maintains high application-level accuracy until an operation failure rate of 10^{-3} , whereas binary methods are acceptable only until 10^{-5} . These results demonstrate that richer stateful logic primitives can significantly improve the efficiency and practicality of unary memristive IMC systems. Future work will focus on larger networks for complex workloads and silicon testing.

ACKNOWLEDGEMENTS

This work was supported by the German Research Foundation (DFG) under Project 550835853, Hector Stiftung, and in part by the National Science Foundation (NSF) under Grant 2609436.

REFERENCES

- [1] O. Mutlu, S. Ghose, J. Gómez-Luna, R. Ausavarungnirun, M. Sadrosadati, and G. F. Oliveira, "A modern primer on processing in memory," 2025. [Online]. Available: <https://arxiv.org/abs/2012.03112>
- [2] A. Boroumand, S. Ghose, Y. Kim, R. Ausavarungnirun, E. Shiu, R. Thakur, D. Kim, A. Kuusela, A. Knies, P. Ranganathan, and O. Mutlu, "Google workloads for consumer devices: Mitigating data movement bottlenecks," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '18. Association for Computing Machinery, 2018, p. 316–331.
- [3] N. TaheriNejad, "In-memory computing: Global energy consumption, carbon footprint, technology, and products status quo," pp. 1–6, 2024.
- [4] M. R. Alam, M. H. Najafi, and N. TaheriNejad, "Sorting in memristive memory," *ACM Journal on Emerging Technologies in Computing Systems*, pp. 1–22, 2022.
- [5] A. Jalilvand, F. S. Banitaba, S. N. Estiri, S. Aygun, and M. H. Najafi, "Sorting it out in hardware: A state-of-the-art survey," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 30, no. 4, Jun. 2025. [Online]. Available: <https://doi.org/10.1145/3734797>
- [6] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. Stanley Williams, "The missing memristor found," *Nature*, vol. 453, pp. 80–83, 2008.
- [7] 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.
- [8] S. Gupta, M. Imani, and T. Rosing, "Felix: fast and energy-efficient logic in memory," in *Proceedings of the International Conference on Computer-Aided Design*, ser. ICCAD '18. New York, NY, USA: Association for Computing Machinery, 2018.
- [9] Z. Sun, S. Kvatinsky, X. Si, A. Mehonic, Y. Cai, and R. Huang, "A full spectrum of computing-in-memory technologies," *Nature Electronics*, vol. 6, no. 11, pp. 823–835, 2023.
- [10] G. Gulafshan, H. Hu, D. Raber-Radakovits, L. Vassallo, G. C. Marques, J. Aghassi-Hagmann, and N. Taherinejad, "Realistic behavioral model for rram capturing non-idealities," *Communications Materials*, vol. 6, no. 1, pp. 1–13, 2025.
- [11] A. Bende, S. Singh, C. Kumar Jha, D. Storelli, D. Nielinger, R. Drechsler, R. Dittmann, S. Menzel, F. Merchant, and V. Rana, "Device-to-logic variability propagation in rram-based logic-in-memory architectures," *Neuromorphic Computing and Engineering*, vol. 6, no. 2, p. 024009, may 2026.
- [12] X. Liu, Z. Pan, Z. Xia, L. Li, Q. Deng, Y. Pan, J. Li, and N. Huo, "Memristors based on 2d materials: Bridging device theory and potential applications," *Advanced Functional Materials*, vol. 36, no. 20, p. e20816, 2026.
- [13] C. Simonides, D. Gausepohl, P. M. Hinkel, F. Seiler, and N. Taherinejad, "Approximated 2-bit adders for parallel in-memristor computing with a novel sum-of-product architecture," *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, vol. 10, pp. 135–143, 2024.
- [14] M. H. Najafi, D. J. Lilja, M. D. Riedel, and K. Bazargan, "Low-Cost Sorting Network Circuits Using Unary Processing," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 8, pp. 1471–1480, Aug 2018.
- [15] P. Schober, M. H. Najafi, and N. Taherinejad, "High-Accuracy Multiply-Accumulate (MAC) Technique for Unary Stochastic Computing," *IEEE Transactions on Computers*, vol. 71, no. 6, pp. 1425–1439, 2022.
- [16] D. Wu, J. Li, R. Yin, H. Hsiao, Y. Kim, and J. S. Miguel, "Ugemm: Unary computing architecture for gemm applications," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 377–390.
- [17] M. Riahi Alam, M. H. Najafi, N. TaheriNejad, M. Imani, and L. Peng, "Stochastic computing for reliable memristive in-memory computation," in *Proceedings of the Great Lakes Symposium on VLSI 2023*, ser. GLSVLSI '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 397–401.
- [18] M. S. Q. Truong, L. Shen, A. Glass, A. Hoffmann, L. R. Carley, J. A. Bain, and S. Ghose, "Adapting the racer architecture to integrate improved in-rram logic primitives," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 12, no. 2, pp. 393–407, 2022.
- [19] B. Hoffer, V. Rana, S. Menzel, R. Waser, and S. Kvatinsky, "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.
- [20] D. Ielmini and H.-S. P. Wong, "In-memory computing with resistive switching devices," *Nature electronics*, vol. 1, no. 6, pp. 333–343, 2018.
- [21] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory devices and applications for in-memory computing," *Nature nanotechnology*, vol. 15, no. 7, pp. 529–544, 2020.
- [22] E. Lehtonen and M. Laiho, "Stateful implication logic with memristors," in *2009 IEEE/ACM International Symposium on Nanoscale Architectures*, 2009.
- [23] X. Wang *et al.*, "Memristor-based XOR gate for full adder," in *CCC2016*, 2016, pp. 5847–5851.
- [24] J. P. C. de Lima, M. S. Moghadam, S. Aygun, J. Castrillon, M. H. Najafi, and A. A. Khan, "All-in-memory stochastic computing using rram," in *2025 62nd ACM/IEEE Design Automation Conference (DAC)*, 2025, pp. 1–7.
- [25] D. Cederman and P. Tsigas, "Gpu-quicksort: A practical quicksort algorithm for graphics processors," *ACM J. Exp. Algorithmics*, vol. 14, Jan. 2010. [Online]. Available: <https://doi.org/10.1145/1498698.1564500>
- [26] K. E. Batchier, "Sorting networks and their applications," in *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, ser. AFIPS '68 (Spring). New York, NY, USA: Association for Computing Machinery, 1968, p. 307–314.
- [27] A. Farmahini-Farahani, H. J. Duwe III, M. J. Schulte, and K. Compton, "Modular design of high-throughput, low-latency sorting units," *IEEE Transactions on Computers*, vol. 62, no. 7, pp. 1389–1402, 2013.
- [28] M. Frátrik, M. Klimo, O. Šuch, and O. Škvarek, "Memristive sorting networks," *physica status solidi c*, vol. 12, no. 1-2, pp. 233–237, 2015. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/pssc.201100000>
- [29] L. Nielsen, O. Such, M. Klimo, R. Waser, and E. Linn, "Memristive sorting networks enabled by electrochemical metallization cells," *Int. J. Unconv. Comput.*, vol. 12, no. 4, pp. 303–317, 2016.
- [30] T. Breuer, L. Nielsen, B. Roesgen, R. Waser, V. Rana, and E. Linn, "Realization of minimum and maximum gate function in ta2o5-based memristive devices," *Scientific Reports*, vol. 6, 2016.
- [31] L. Yu, Z. Jing, Y. Yang, and Y. Tao, "Fast and scalable memristive in-memory sorting with column-skipping algorithm," in *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2022, pp. 590–594.
- [32] L. Yu, T. Zhang, Z. Wang, X. Wang, Z. Pan, B. Wang, Z. Jing, J. Liu, Y. Li, Z. Xie *et al.*, "A fast and reconfigurable sort-in-memory system based on memristors," *Nature Electronics*, vol. 8, no. 7, pp. 597–609, 2025.
- [33] S. Singh, C. K. Jha, A. Bende, P. L. Thangkiew, V. Rana, S. Patkar, R. Drechsler, and F. Merchant, "Should we even optimize for execution energy? rethinking mapping for magic design style," *IEEE Embedded Systems Letters*, vol. 15, no. 4, pp. 230–233, 2023.
- [34] F. Seiler, Gulafshan, and N. TaheriNejad, "An efficient robust serial imply-based in-memristor adder," in *2025 Cross-Disciplinary Conference on Memory-Centric Computing (CCMCC)*, 2025, pp. 1–7.
- [35] M. Riahi Alam, M. H. Najafi, N. Taherinejad, M. Imani, and R. Gotumukkala, "Stochastic computing in beyond von-neumann era: Processing bit-streams in memristive memory," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 5, 2022.
- [36] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.
- [37] F. Seiler and N. TaheriNejad, "Accelerated image processing through imply-based ncarry approximated adders," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 71, no. 11, pp. 5141–5154, 2024.