MAC-based Mapping of Adder Architectures to RRAM Crossbars

Fatemeh Shirinzadeh*, Saeideh Nabipour*, Kamalika Datta*†
Chandan Kumar Jha†, Saeideh Shirinzadeh*‡, Rolf Drechsler*†
* Cyber-Physical Systems, DFKI GmbH, Germany
† Institute of Computer Science, University of Bremen, Germany
‡ Fraunhofer Institute for Systems and Innovation Research (ISI), Karlsruhe, Germany
{fatemeh.shirinzadeh, saeideh.nabipour, saeideh.shirinzadeh}@dfki.de, {kdatta, chajha, drechsler}@uni-bremen.de

***Abstract*—This paper presents a comparative evaluation of seven adder architectures mapped to RRAM crossbars using a MAC-based logic style. By analyzing designs across 8- to 64-bit widths, we assess trade-offs in total evaluation cycles, crossbar area, and logical depth. The results highlight Ladner-Fischer and Kogge-Stone adders as the most efficient for high-bit-width applications, offering superior parallelism and scalability in MAC-based in-memory computing systems. Hence, these adder designs can be used as a reference for future works related to the optimization of adders for MAC-based logic style.**

I. Introduction

In-memory computing architectures have emerged as a promising alternative to traditional von Neumann architectures due to their ability to reduce data movement and improve computational efficiency. Among various device technologies, memristor-based architecture has gained significant traction, offering both high density and low energy operation for logic and memory integration [1], [2]. Recent studies have explored arithmetic circuit implementations, such as adders, using digital techniques like Memristor Aided Logic (MAGIC) design style [3]. In a notable contribution, [4] analyzed several adder architectures, including Ripple Carry (RC), Carry-Lookahead (CL), Kogge-Stone (KS), Brent-Kung (BK), and Serial Prefix (SE), mapped to MAGIC-style memristor crossbars. Their study demonstrated that serial prefix offers superior performance in terms of gate count and computation cycles when compared to traditional RC designs. This highlights the importance of architecture-aware mapping in in-memory computing design. However, this work focuses on the serial execution of operations, which are mapped to multiple rows of the crossbars [5]. Hence, RC designs were found to be optimal, as the mapping used does not exploit any parallelism.

Inspired by their approach, this work investigates *Multiply and Accumulate (MAC) based logic mapping*, which is critical for arithmetic-intensive applications such as neural networks and digital signal processing. In MAC-based computing, optimizing the mapping of adders is essential since addition is a core operation performed repeatedly across parallel computing units.

To evaluate and compare adder architectures within this context, we rely on formal descriptions of prefix adders. These are typically divided into three stages: generation of propagate/generate signals, carry computation, and sum generation.

$$g_i = a_i \wedge b_i, \quad p_i = a_i \oplus b_i \tag{1}$$

$$g_{i:j} = g_{i:k} \vee (p_{i:k} \wedge g_{k-1:j}), \quad p_{i:j} = p_{i:k} \wedge p_{k-1:j} \tag{2}$$

$$c_{i+1} = g_i \vee (p_i \wedge c_i) \tag{3}$$

$$s_i = p_i \oplus c_{i-1} \tag{4}$$

Equation (1) defines the initial generate and propagate terms, which are then used in the carry computation tree in Equation (2). Equation (3) calculates the carry bits, while Equation (4) derives the final sum. Various adder architectures implement the prefix operation in Equation (2) differently to balance trade-offs between depth (parallelism), area, and fan-out.

In this paper, we compare the suitability of different adder architectures for MAC-based mapping in terms of *parallelization capability*, *area overhead*, and *logical complexity*. Although ripple carry adders offer minimal area and straightforward design, they suffer from limited parallelism and a longer delay. Conversely, prefix adders such as Kogge-Stone and Brent-Kung enable higher parallelism at the cost of increased area and wiring complexity. Our analysis aims to assess these trade-offs within MAC-oriented in-memory computing implementations and identify the most efficient designs for scalable, high-performance applications.

The rest of the paper is organized as follows. Section II presents the required background, including the RRAM crossbar structure and MAC operation. Section III introduces the MAC-based mapping methodology. Section IV discusses the evaluation results, and Section V concludes the paper and outlines directions for future work.

II. Background

A. RRAM Crossbar

With the growing demand for high-density, low-power, high-speed, and cost-effective non-volatile memory (NVM) solutions, various emerging memory technologies have been extensively explored. Among these, Resistive Random-Access Memory (RRAM) has gained significant attention as a strong

candidate for next-generation NVM due to its excellent scalability, rapid switching speed, simple device structure, multi-level storage capability, compatibility with CMOS technology, and suitability for crossbar array integration. RRAM is a two-terminal memristive device whose resistance can be modulated between a Low Resistance State (LRS, representing logic 1) and a High Resistance State (HRS, representing logic 0) by applying voltage pulses of appropriate magnitude and polarity. These devices are typically implemented in crossbar architectures, where each memory cell is formed at the junction of orthogonally arranged nanowires, bitlines and wordlines, enabling dense and efficient memory arrays [6]–[10].

B. MAC Operation

In addition to all the advantages mentioned above, RRAM's unique analog computation capability is widely recognized for enabling efficient MAC operations, which are fundamental to matrix-vector multiplication. This feature makes RRAM particularly attractive for in-memory computing and neural network acceleration. In a typical crossbar configuration, Fig. 1, the conductance values of RRAM cells are programmed to represent the inverse weights $a_{j,k}^{-1}$.

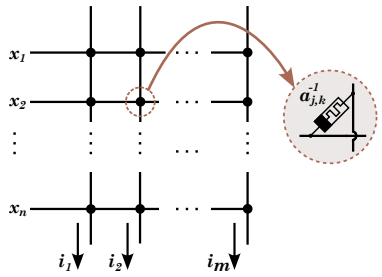

Fig. 1. MAC computation through RRAM crossbar structure

When input voltages $x_1, \ldots, x_n$ are applied to the horizontal rows, the resulting currents along the vertical columns naturally perform MAC operations in parallel. Specifically, each column output i_j corresponds to a weighted sum of inputs, expressed as $i_j = \sum_{k=1}^{n} x_k \cdot a_{j,k}$, or more compactly, $I = XA$, where $I = (i_1, \ldots, i_m)$, $x = (x_1, \ldots, x_n)$, and the weight matrix A is defined as:

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \ldots & a_{1,m} \\ a_{2,1} & a_{2,2} & \ldots & \ldots & a_{2,m} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & \ldots & a_{n,m} \end{bmatrix} \tag{5}$$

This structure enables the parallel execution of m MAC operations, each involving n multiplications, in a single computational cycle. While such analog operations have been extensively explored in neuromorphic applications, their use for digital logic remains largely untapped. In this work, we leverage RRAM-based MAC operations to implement different adder structures through a crossbar mapping approach.

III. MAC-BASED MAPPING METHODOLOGY

The MAC-based mapping methodology enables efficient logic computation directly on RRAM crossbars by expressing Boolean functions as sequences of MAC micro-operations [11]. These operations exploit the analog current summation capability of RRAM devices, allowing parallel computation and reduced evaluation cycles. Unlike traditional gate-based approaches such as IMPLY, MAJ, or MAGIC, our method encodes Boolean `OR` functions into MAC-based micro-operations mapped to the crossbar.

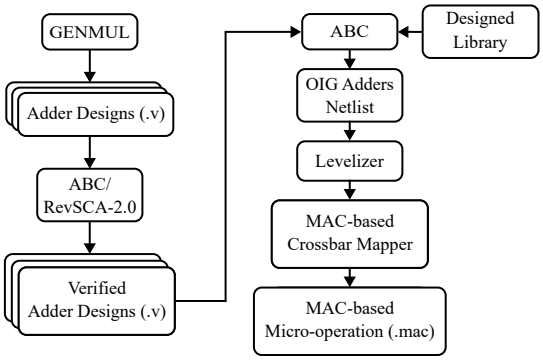

Fig. 2. The overall flow of analysis of adder architectures

The evaluation process is illustrated in Fig. 2. We utilized the GENMUL framework to generate adder designs with bit widths ranging from 8 to 64 [4]. These were converted to (.aig) format using Yosys and formally verified using the RevSCA-2.0 framework. Combinational equivalence checking was then performed using the ABC tool [12]. The verified adder files were subsequently input into the MAC-based synthesis and mapping flow. In the first step of this flow, the input function was converted into an OR-Inverter Graph (OIG) using the ABC tool along with a designed library. The OIG is a directed acyclic graph composed of three node types: (i) primary input nodes without incoming edges, (ii) primary output nodes without outgoing edges, and (iii) internal nodes representing logical `OR` gates, each with at most ten inputs and a single output.

Edges in the OIG may be regular or complemented, representing direct or negated connections, respectively. Formally, an OIG over input variables $X = \{x_1, x_2, \ldots, x_n\}$ and output variables $Y = \{y_1, y_2, \ldots, y_m\}$ is defined as a DAG $H = (V, E)$, where:

- $V = V_X \cup V_H \cup V_Y$, with V_X and V_Y denoting primary input and output nodes, and $V_H = \{v_{h1}, v_{h2}, \ldots, v_{hk}\}$ representing internal `OR` nodes;
- Each edge $e \in E$ is defined as $(u, (v \times p))$, where $u \notin V_Y$, $v \notin V_X$, and $p = 1$ for a regular edge or $p = 0$ for a complemented edge.

The OIG is levelized by assigning each node to a logic level based on its topological depth from the primary inputs. This hierarchical organization defines the evaluation sequence for MAC operations, ensuring that all gates at the same level can be computed concurrently before proceeding to the next.

Each internal `OR` gate is mapped to a unique crossbar column, while each input variable and its complement are assigned to separate rows. Intermediate results between levels are stored and propagated through column-to-row mappings.

TABLE I
MAC-BASED EVALUATION OF 8-BIT ADDERS

Designs	#PI/PO	#OR	#L	CBS	C_I	C_E	C_T
BK_8	8/8	52	16	134 x 60	60	31	91
CK_8	8/8	59	18	134 x 60	60	35	95
CL_8	8/8	52	16	134 x 60	60	31	91
KS_8	8/8	53	15	134 x 60	60	29	89
LF_8	8/8	53	16	136 x 61	61	31	92
RC_8	8/8	59	18	134 x 60	60	35	95
SE_8	8/8	52	16	134 x 60	60	31	91

TABLE II
MAC-BASED EVALUATION OF 16-BIT ADDERS

Designs	#PI/PO	#OR	#L	CBS	C_I	C_E	C_T
BK_16	16/16	109	32	280 x 125	125	63	188
CK_16	16/16	123	34	278 x 124	124	67	191
CL_16	16/16	115	32	282 x 126	126	63	189
KS_16	16/16	121	27	304 x 137	137	53	190
LF_16	16/16	108	32	278 x 124	124	63	187
RC_16	16/16	123	34	278 x 124	124	67	191
SE_16	16/16	108	32	278 x 124	124	63	187

TABLE III
MAC-BASED EVALUATION OF 32-BIT ADDERS

Designs	#PI/PO	#OR	#L	CBS	C_I	C_E	C_T
BK_32	32/32	226	64	566 x 252	252	127	379
CK_32	32/32	251	66	566 x 252	252	131	383
CL_32	32/32	266	64	632 x 285	285	127	412
KS_32	32/32	252	53	630 x 284	284	105	389
LF_32	32/32	247	43	602 x 270	270	85	355
RC_32	32/32	251	66	566 x 252	252	131	383
SE_32	32/32	220	64	566 x 252	252	127	379

TABLE IV
MAC-BASED EVALUATION OF 64-BIT ADDERS

Designs	#PI/PO	#OR	#L	CBS	C_I	C_E	C_T
BK_64	64/64	498	84	1222 x 548	548	167	715
CK_64	64/64	507	130	1142 x 508	508	259	767
CL_64	64/64	561	128	1328 x 601	601	255	856
KS_64	64/64	551	80	1282 x 578	578	159	737
LF_64	64/64	549	57	1328 x 601	601	113	714
RC_64	64/64	507	130	1142 x 508	508	259	767
SE_64	64/64	444	128	1142 x 508	508	255	763

Before evaluation, initialization instructions are generated to configure the RRAM cells associated with each gate input in a low-resistance state.

The number of initialization cycles equals the number of `OR` gates, denoted as N. Each level requires one cycle for the MAC operation, followed by a feedback cycle for data propagation. Thus, the total number of MAC cycles is:

$$\#\text{MAC Cycles} = (2 \times L) - 1, \tag{6}$$

where L is the number of logic levels. Hence, the total number of cycles required to evaluate the Boolean function is:

$$\#\text{Total Evaluation Cycles} = N + (2 \times L) - 1. \tag{7}$$

The size of the RRAM crossbar varies depending on the structure of the Boolean function. In the case of a comparison function with an equal number of primary inputs and outputs, the number of crossbar rows is proportional to the number of gates N, while the number of columns corresponds to the number of levels L.

The complete mapping process generates a `.mac` file, which serves as a low-level executable specification for MAC-based logic execution on RRAM crossbars.

By leveraging the analog computation capabilities of RRAM, the proposed methodology achieves scalable and parallel logic evaluation with minimal memory reconfiguration.

IV. RESULTS AND DISCUSSION

In this section, we present the results of mapping various adders in the memristor crossbars. Tables I to IV summarize the evaluation results for seven different adder architectures with bit-widths ranging from 8 to 64 bits. The evaluated designs include *Ripple Carry (RC)*, *Carry Look-Ahead (CL)*, *Ladner-Fischer (LF)*, *Kogge-Stone (KS)*, *Brent-Kung (BK)*, *Carry-Skip (CK)*, and *Serial Prefix (SE)*. The first two columns in each table list the adder name along with the number of Primary Inputs (`PI`) and Primary Outputs (`PO`). The subsequent six columns report synthesis and mapping results, including the number of OR gates (`#OR`), number of logic levels (`#L`), crossbar size (`CBS`), initialization cycles (C_I), evaluation cycles (C_E), and total execution cycles ($C_T = C_I + C_E$). For logic synthesis and mapping process, we use the ABC tool in conjunction with a state-of-the-art MAC-based mapping tool as proposed in [11]. All experiments were carried out on a machine equipped with an Intel® Core™ i7 2.10 GHz processor and 32GB of main memory.

A. 8-bit Adders

The results for the 8-bit adders are summarized in Table I. All designs, except for LF_8, utilize a crossbar of size 134×60, while LF_8 requires a slightly larger crossbar of 136×61, indicating a higher hardware demand. In terms of total cycles, the latency ranges from 89 cycles (KS_8) to 95 cycles (CK_8 and RC_8). The KS_8 adder demonstrates the lowest total latency, outperforming the RC_8 and CK_8 adders by approximately 6.3% in total cycles. Meanwhile, the LF_8 adder incurs a slight increase in both crossbar size and latency, suggesting a trade-off between area and speed. Overall, KS_8 is the most efficient in terms of latency, while SE_8, CL_8, and BK_8 offer balanced performance with moderate crossbar size and total cycles.

B. 16-bit Adders

The results for the 16-bit adders are shown in Table II. It can be seen that the crossbar sizes range from 278×124 to 304×137, while total cycles vary from 187 (LF_16, SE_16) to 191 (CK_16, RC_16). The LF_16 and SE_16 adders demonstrate the best performance, achieving the lowest total latency of 187 cycles, outperforming BK_16 (188) and CL_16 (189). In contrast, KS_16 exhibits the highest latency and largest crossbar size, making it the least efficient option for MAC-based implementations.

C. 32-bit Adders

The results for the 32-bit adders are shown in Table III. We can see that the crossbar sizes range from 566×252 to

632×285, while total cycles vary from 355 (LF_32) to 412 (CL_32). The LF_32 adder demonstrates the best performance, achieving the lowest total latency of 355 cycles, outperforming SE_32 and BK_32, both at 379 cycles. In contrast, CL_32 and KS_32 exhibit the highest latency and largest crossbar sizes, making them the least efficient options for MAC-based implementations.

D. 64-bit Adders

The results for the 64-bit adders are presented in Table IV. It can be observed that the crossbar sizes range from 1142×508 to 1328×601, and total cycles range from 714 (LF_64) to 856 (CI_64). The LF_64 adder shows the best performance, achieving the lowest total latency of 714 cycles, closely followed by BK_64 with 715 cycles. In contrast, the CI_64 and CK_64 designs incur the highest latency and require the largest crossbar resources, making them the least suitable for efficient MAC-based execution.

E. Overall Discussion

Table I through Table IV present a comprehensive MAC-based evaluation of various adder architectures across 8-, 16-, 32-, and 64-bit implementations. The evaluation focuses on three main criteria: *parallelization capability*, *area overhead*, and *logical complexity*.

Parallelization capability: The total cycle count $C_T = C_I + C_E$ reflects the effective execution time on a MAC-based architecture. Across all bit-widths, the Kogge-Stone (KS) and Ladner-Fischer (LF) adders consistently achieve lower C_T values. For instance, at 64 bits, LF_64 requires only 714 cycles compared to 856 for CL_64 and 767 for CK_64 and RC_64. The lower cycle counts of LF and KS stem from their balanced carry propagation strategies, which allow better exploitation of MAC parallelism.

Area overhead: The Crossbar Size (CBS) metric indicates the required area. While KS and LF adders show good cycle performance, they also tend to require larger crossbar sizes, particularly CL_64 and LF_64 with 1328×601. In contrast, RC and CK adders typically show more compact layouts but at the cost of higher execution cycles.

Logical complexity: The number of OR gates and logic levels (#OR, #L) capture the logic depth and gate usage. Ripple-Carry (RC) and Brent-Kung (BK) adders offer relatively low logic depth for smaller widths, but their complexity grows rapidly with bit-width. LF consistently maintains a low number of logic levels (e.g., only 57 for LF_64), supporting fast signal propagation in MAC.

Scalability and trade-offs: As the bit-width increases, the difference in C_T becomes more pronounced. LF and KS maintain better scalability in execution cycles, while architectures like CK and RC show a sharp increase in both area and latency. Notably, LF_64 offers the best trade-off between parallelization and logic complexity, making it a strong candidate for energy-efficient MAC-based mapping in high-bit-width operations.

In summary, Ladner-Fischer and Kogge-Stone adders demonstrate superior parallelization performance with manageable complexity, while RC and CK offer simpler hardware at the cost of higher execution latency. The optimal choice depends on the target system's priorities, throughput versus area.

V. Conclusion

This work presents an evaluation of seven adder architectures with bit widths from 8 to 64, mapped to MAC-based in-memory computing structures. Using a state-of-the-art MAC mapping tool, we analyze each design in terms of execution cycles and crossbar area. The Ladner-Fischer (LF) adder consistently achieves the best performance for 16, 32, and 64-bit designs, while the Kogge-Stone (KS) adder proves most efficient for 8-bit implementations. The results provide valuable insights into the effective design of arithmetic units in MAC-based in-memory systems.

Acknowledgement

This work is partly supported by the German Research Foundation (DFG) within the Project PLiM (DR 287/35-1, DR 287/35-2 and SH 1917/1-2), the German Ministry for Research, Technology and Space (BMFTR) with project ExaVerse (grant number 01IW25003), and the Data Science Center of the University of Bremen (DSC@UB) funded by the State of Bremen.

References

[1] 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.

[2] K. Sun, J. Chen, and X. Yan, "The future of memristors: materials engineering and neural networks," *Advanced Functional Materials*, vol. 31, no. 8, p. 2006773, 2021.

[3] 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.

[4] C. K. Jha, A. Mahzoon, and R. Drechsler, "Investigating various adder architectures for digital in-memory computing using magic-based memristor design style," in *2022 IEEE International Conference on Emerging Electronics (ICEE)*. IEEE, 2022, pp. 1–4.

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

[6] Y. Deng, P. Huang, B. Chen, X. Yang, B. Gao, J. Wang, L. Zeng, G. Du, J. Kang, and X. Liu, "Rram crossbar array with cell selection device: A device and circuit interaction study," *IEEE Transactions on Electron Devices*, vol. 60, no. 2, pp. 719–726, 2013.

[7] L. O. Chua and M. K. Sung, "Memristive devices and systems," *Proceedings of the IEEE*, vol. 64, no. 2, pp. 209–223, February 1976.

[8] J. Liu, B. Yu, and M. P. Anantram, "Scaling analysis of nanowire phase-change memory," *IEEE Electron Device Letters*, vol. 32, no. 10, pp. 1340–1342, 2011.

[9] M. Prezioso *et al.*, "Training and operation of an integrated neuromorphic network based on metal-oxide memristors," *Nature*, vol. 521, no. 7550, pp. 61–64, 2015.

[10] C. Nauenheim, C. Kugeler, A. Rudiger, R. Waser, A. Flocke, and T. G. Noll, "Nano-crossbar arrays for nonvolatile resistive ram (rram) applications," in *2008 8th IEEE Conference on Nanotechnology*, 2008, pp. 464–467.

[11] F. Shirinzadeh, A. Kole, K. Datta, S. Shirinzadeh, and R. Drechsler, "A comprehensive synthesis and verification approach for rram-based neuromorphic computing," in *Proceedings of the Euromicro Conference Series on Digital System Design (DSD)*, Italy, 2025, to appear.

[12] R. Brayton and A. Mishchenko, "Abc: An academic industrial-strength verification tool," in *International Conference on Computer Aided Verification*. Springer, 2010, pp. 24–40.