Fan-In Aware Graph-Based Optimization for MAC-Based In-Memory Computing

Fatemeh Shirinzadeh* Abhoy Kole* Kamalika Datta*‡
fatemeh.shirinzadeh@dfki.de abhoy.kole@dfki.de kdatta@uni-bremen.de

Saeideh Shirinzadeh*† Rolf Drechsler*‡
saeideh.shirinzadeh@dfki.de drechsler@uni-bremen.de

*German Research Centre for Artificial Intelligence (DFKI), Bremen, Germany
†Fraunhofer Institute for Systems and Innovation Research (ISI), Karlsruhe, Germany
‡Institute of Computer Science, University of Bremen, Germany

***Abstract*—Resistive RAM (RRAM) has emerged as a promising technology for in-memory computing, allowing both storage and computation within the same physical substrate. Although its ability to perform analog computations, especially multiply-accumulate (MAC) operations, has been effectively utilized in neuromorphic systems, there has been limited research on its applicability to Boolean logic synthesis. Existing approaches typically rely on graph-based representations of Boolean functions that are mapped to column-wise MAC operations on standard RRAM crossbars. However, these representations largely inherit binary fan-in constraints from conventional logic synthesis flows, resulting in limited exploitation of MAC-level parallelism and underutilization of available crossbar resources. In this work, we address this limitation by introducing the concept of multi-input OR–Inverter Graphs (m-OIGs), which allow OR nodes with fan-in greater than two to better match the accumulation semantics of MAC operations. Experimental results on standard benchmark suites demonstrate that increasing OR fan-in consistently reduces both crossbar area and total evaluation cycles, leading to improved performance and more efficient use of RRAM crossbar resources, highlighting the importance of fan-in–aware logic representations.**

***Index Terms*—In-Memory Computing, MAC Operation, OR-Inverter Graph**

I. Introduction

Rising performance and energy constraints in modern computing systems have motivated increasing interest in in-memory computing approaches based on memristive technologies such as *Resistive Random Access Memory (RRAM)*. By enabling computation to be carried out directly inside memory arrays, these technologies reduce communication overhead and improve overall system efficiency. RRAM devices are especially attractive for logic realization because they can retain state while actively participating in computation. Their non-volatile behavior, fast resistive switching, and low energy requirements make them well-suited for dense integration. Moreover, their compact structure and uniform fabrication process allow large-scale deployment in crossbar architecture [1], [2]. To enable logic-in-memory computation with RRAM devices, a variety of automated mapping approaches have been proposed. These methods typically rely on universal logic primitives that can be directly executed within RRAM arrays, such as *material implication (IMPLY)* [3], resistive majority-of-three *(MAJ)* operations [4], [5], and *Memristor-Aided LoGIC (MAGIC)* [6]. Beyond Boolean logic evaluation, RRAM crossbars naturally support parallel *Multiply-And-Accumulate (MAC)* operations by summing currents along memory columns, where device conductances are tuned to represent weights. This enables efficient matrix-vector multiplication, a core operation in neuromorphic processing [7]–[9].

More recently, it has been observed that the same MAC mechanisms can be repurposed for Boolean logic evaluation, enabling *on-demand logical computation* on standard neuromorphic RRAM hardware without additional circuitry [10]–[13]. This opens the possibility of *hybrid logical–neuromorphic* computing systems, where inference-oriented MAC hardware can also execute control logic, verification tasks, or Boolean reasoning workloads within the same memory fabric [10]. In this context, MAC-based logic execution offers a fundamentally different execution model compared to IMPLY-, MAJ-, or MAGIC-based approaches, as it allows multiple literals or intermediate signals to be accumulated and evaluated in parallel within a single cycle.

In this direction, Fröhlich *et al.* introduced *multi-input And–Inverter Graphs (m-AIGs)* as a logic representation for parallel evaluation on RRAM crossbars [14]. By allowing AND nodes with fan-in greater than two, m-AIGs reduce logic depth and expose structural parallelism. The proposed mapping targets MAJ-based logic-in-memory operations, exploiting the ability of RRAM crossbars to evaluate multiple MAJ functions in parallel. This work highlights the importance of adapting logic representations to the underlying execution primitive of memory technology.

Despite these advantages, the potential of MAC operations for logic synthesis and execution has not yet been fully explored. In particular, existing logic representations and synthesis flows are largely inherited from gate-level or binary-fan-in graph models, which are not explicitly optimized for accumulation-based execution. Fan-in restrictions in conventional logic graphs can lead to increased depth, redundant intermediate nodes, and underutilization of the intrinsic parallelism offered by MAC-capable crossbars. As a result, execution latency and crossbar resource usage may remain suboptimal, limiting the benefits of MAC-based logic-in-memory computing.

To address this challenge, this work investigates the use of *Multi-input OR–Inverter Graphs (m-OIGs)* for MAC-based synthesis, as they provide a native logic representation in which each graph node can be mapped directly to a crossbar column. By allowing OR nodes with higher fan-in, the proposed approach better aligns logic structure with the accumulation

979-8-3315-8229-6/26/$31.00

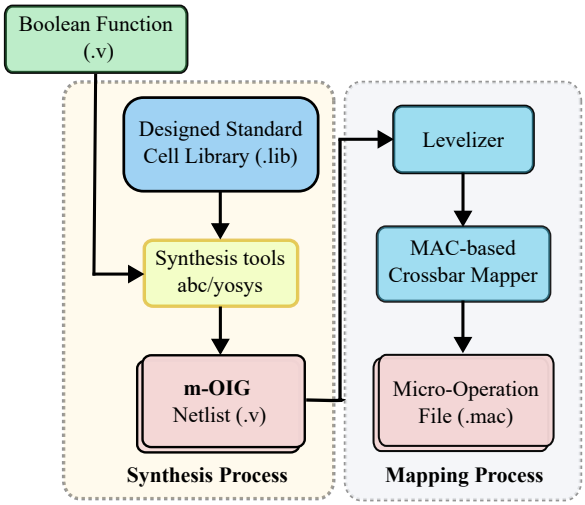


Fig. 1: Overall flow of MAC-based mapping with m-OIG

semantics of MAC operations. This reduces graph depth, simplifies execution scheduling, and exposes higher degrees of parallelism during evaluation. As a result, Boolean functions can be translated into MAC micro-operations more efficiently, leading to fewer execution cycles and improved crossbar utilization while preserving full compatibility with standard RRAM crossbar architectures.

II. Synthesis and Mapping for MAC-Based RRAM Computing

A. MAC-Based Design Synthesis Using m-OIG

The proposed methodology maps Boolean functions onto RRAM crossbars using m-OIG representations, as illustrated in Fig. 1. Starting from a Verilog description, synthesis tools such as ABC [15] or Yosys [16] generate an m-OIG that exposes multi-fan-in OR structures for efficient accumulation-based evaluation.

An *OIG* is a directed acyclic graph used to represent Boolean functions by combining OR operations with edge-level inversion.

Definition 1: An *OR Inverter Graph (OIG)* over the primary input variables $X = \{x_1, x_2, \ldots, x_n\}$ and the primary output variables $Y = \{y_1, y_2, \ldots, y_m\}$ is a directed acyclic graph $H = (V, E)$ with the following characteristics:

- A finite set of nodes $V = V_X \cup V_H \cup V_Y$, where V_X and V_Y denote the sets of primary input and primary output nodes, respectively, and $V_H = \{v_{h1}, v_{h2}, \ldots, v_{hk}\}$ represents non-terminal nodes implementing Boolean OR operations.
- Each edge $e \in E$ connects a source node $u \in V$ to a target node $v \in V$, with $u \notin V_Y$ and $v \notin V_X$. An edge is represented as $(u, (v \times p))$, where $p = 1$ denotes a regular edge and $p = 0$ denotes a complemented edge.

The depth of an OIG corresponds to the number of logic levels from primary inputs to primary outputs.

Multi-Input OR-Inverter Graphs (m-OIG) In this work, we extend OIGs to *Multi-input OR–Inverter Graphs (m-OIGs)* by allowing OR nodes with fan-in greater than two. For a parameter $m \geq 2$, each OR node may have a fan-in between 2 and m, enabling a heterogeneous mix of node sizes. This flexibility preserves wide disjunctions that would otherwise be decomposed into cascaded trees, reducing logic depth and increasing parallelism, which is well-suited to MAC-based execution.

Once the m-OIG is generated, a levelized intermediate representation is derived. During this step, all nodes are analyzed based on their input dependencies and assigned to discrete logic levels. Each level contains OR nodes whose outputs depend only on signals produced in previous levels. This levelization step is essential for scheduling MAC operations and determining the execution order on the crossbar.

Based on the intermediate levelized netlist, the MAC-based mapper starts generating micro-operations that can be evaluated on the crossbar. First, all primary inputs and their complements are written onto dedicated crossbar rows. This step corresponds to establishing physical connections between the primary input signals and the rows of the crossbar.

In the proposed mapping, each OR gate or logic node of the graph is assigned to a separate column of the RRAM crossbar, while its input literals are mapped to distinct rows. The RRAM devices located at the intersections of the input rows and the gate's column form the physical realization of the logic node.

Initialization is performed in a sequential, row-wise procedure. For each logic node in the graph, all of its input literals are initialized simultaneously within a single initialization cycle. To this end, the crossbar rows associated with the node are programmed to represent the logical values of the node's input variables by setting the resistive states of the corresponding RRAM devices to low or high resistance, encoding logic one and logic zero, respectively.

Once initialization is completed, all logic nodes at the same logic level are evaluated simultaneously and in parallel. For this, a zero voltage is applied to the selected column, while a logic-one voltage is applied to the rows connected to the gate's input literals, with all other rows held at zero voltage. The accumulated current sensed at each column represents the output logic state of the corresponding OR gate, enabling its evaluation within a single MAC cycle.

Based on the crossbar mapping and execution model described above, the execution cost of a Boolean function can be quantified in terms of initialization and evaluation cycles. For a Boolean function with n primary inputs, m primary outputs, N logic nodes, and L logic levels, the total number of initialization (write) cycles is equal to the number of logic nodes, since each node is mapped to a dedicated crossbar column that must be programmed once:

$$\#\text{Initialization Cycles} = N \tag{1}$$

Initialization follows a sequential, row-wise procedure in which, for each logic gate, all corresponding input variables are programmed simultaneously within a single cycle. Therefore, initialization cycles are counted per gate rather than per individual variable, as reflected in Eq. 1.

During evaluation, all OR nodes belonging to the same logic level are computed in parallel using a single MAC cycle. As a result, the total number of MAC cycles is given by:

$$\#\text{MAC Cycles} = L \tag{2}$$

Consequently, the overall evaluation latency of the Boolean function is:

$$\#\text{Total Evaluation Cycles} = N + L \tag{3}$$

The required size of the RRAM crossbar depends on the number of primary inputs, intermediate signals, and logic gates in the synthesized design. Each Boolean variable is represented

by two separate rows corresponding to its true and complemented forms. This dual-row representation enables direct access to both signal polarities during computation, eliminating the need for explicit inverter operations.

Each logic OR gate in the synthesized representation is evaluated in a dedicated crossbar column. Primary outputs are not mapped back into rows; instead, they are sensed directly from the corresponding output columns after evaluation. As a result, the total number of columns equals the number of logic gates.

B. Design Rationale and Benefits of m-OIG

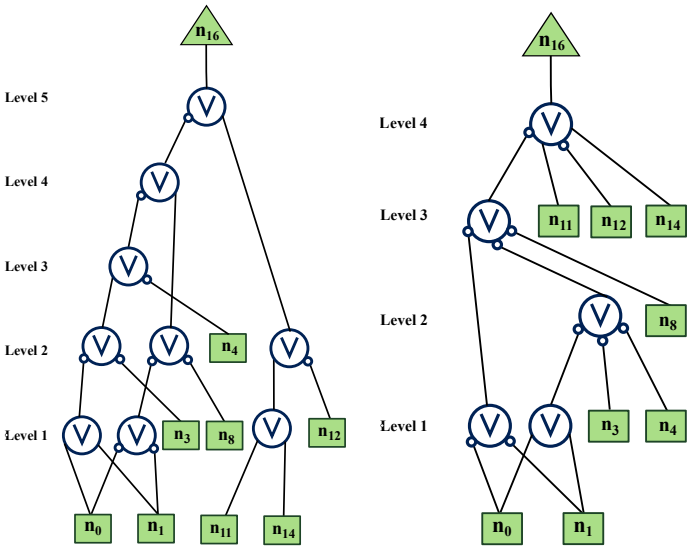
The use of m-OIGs is motivated by the execution characteristics of accumulation-based architectures and the desire to make MAC execution more efficient. Because MAC-based architectures can accumulate multiple inputs in parallel within a column, the cost of evaluating an OR operation is dominated by the number of sequential accumulation steps rather than by its fan-in. As a result, the evaluation cost of an OR operation is largely independent of the number of inputs, provided that all operands can be accumulated concurrently. In a crossbar realization, the accumulated current of each column is sensed and converted by an ADC, which can serve either as a final output or as an intermediate signal for subsequent computation stages.

By allowing OR nodes with fan-in greater than two, m-OIGs preserve wide logic structures that would otherwise be decomposed into cascaded binary trees. This enables more devices (rows) to actively participate in each column during a single MAC operation, effectively increasing intra-column parallelism. As a result, multiple logical inputs are evaluated simultaneously, reducing the need for sequential accumulation across multiple MAC cycles. From a logic perspective, this is analogous to increasing the fan-in of OR gates, leading to denser graph representations with fewer nodes and reduced logic depth.

From a synthesis standpoint, m-OIGs trade increased fan-in for reduced logic depth while maintaining a structured and deterministic mapping flow. Each OR node is mapped to a single MAC evaluation step, and edge-level inversion preserves signal polarity without introducing additional operations or hardware overhead. This alignment between graph structure and MAC execution enables the sensing and ADC stages to naturally capture the result of wide logical accumulations in a single cycle.

These structural properties have two important implications for crossbar-based execution. First, reducing logic depth lowers the number of intermediate signals that must be generated and stored, which in turn reduces crossbar resource utilization and improves scalability. Second, by involving more devices per column and reducing the number of sequential MAC evaluations, the total number of MAC cycles is lowered, which directly decreases switching activity and reduces energy consumption and latency. It should be noted that the effectiveness of this approach depends on the structure of the target Boolean function. While some functions naturally expose wide disjunctions, others may offer limited opportunities for fan-in expansion. Accordingly, this work aims to explore a more efficient logical perspective, this is analogous to increasing the fan-in of OR gates, resulting in enabling the synthesis flow to better exploit the parallel accumulation capabilities of MAC-based in-memory architectures.

To illustrate the structural and execution-level implications of increasing OR fan-in, Fig. 2 presents two alternative OIG representations of the same arbitrary Boolean function.



(a) 2-OIG representation (b) 4-OIG representation

Fig. 2: Structural comparison of OIG representations for the same Boolean function.

Listing 1: MAC-based micro-operation sequences for different OIG representations

```
(a) 2-OIG micro-operations
--------------------------------
0 False 2 True 0 True
1 False 3 True 1 True
2 False 14 True 10 True
3 False 17 True 5 True
4 False 19 True 9 True
5 False 13 True 20 True
6 False 7 True 22 True
7 False 29 True 24 True
8 False 31 True 26 True

Outputs placed at columns:
n16 -> col. 8
Metrics:
Levels : 5
Crossbar size : 32 x 9
Initialisation cycles : 9
Evaluation cycles : 5

(b) 4-OIG micro-operations
--------------------------------
0 False 2 True 0 True
1 False 3 True 1 True
2 False 17 True 7 True 5 True
3 False 19 True 21 True 9 True
4 False 14 True 10 True 23 True 13 True

Outputs placed at columns:
n16 -> col. 4
Metrics:
Levels : 4
Crossbar size : 24 x 5
Initialisation cycles : 5
Evaluation cycles : 4
```

The first representation is restricted to 2-input OR nodes (2-OIG), whereas the second allows OR nodes with fan-in up to four (4-OIG). The OIGs comprise eight input bits, namely n_0, n_1, n_3, n_4, n_8, n_{11}, n_{12}, and n_{14}, and produce a single output bit, n_{16}. Circular nodes denote OR gates, while bubbles on the edges indicate input signal inversion. Both graphs implement identical logic functionality; however, they differ in graph depth, node count, and the degree of parallelism exposed during MAC-based execution.

While the two OIG representations shown in Fig. 2 are functionally equivalent, their structural differences lead to distinct

sequences of MAC-based micro-operations during execution.

To make this distinction explicit, Listing 1 presents the corresponding micro-operation schedules generated for the 2-OIG and 4-OIG representations. The listings highlight how increased OR fan-in reduces the number of intermediate accumulation steps and exposes greater parallelism at the micro-operation level. Compared to the 2-OIG implementation, the 4-OIG representation achieves lower logic depth, fewer MAC cycles, and a substantially smaller crossbar, with crossbar size and initialization overhead reduced by nearly 45%.

III. Experimental Results

This section presents an experimental evaluation of the proposed fan-in–aware m-OIG synthesis flow from two complementary perspectives. First, we analyze the impact of increasing OR fan-in on latency and area when mapping Boolean logic to RRAM-based crossbar architectures. Second, we compare the proposed m-OIG approach with the m-AIG baseline.

All experiments were conducted using the ISCAS'85 and IWLS 2005 benchmark suites [17], [18]. The experiments were performed on a machine equipped with an Intel® Core™ i7 processor running at 2.10 GHz with 8 GB of main memory. The proposed synthesis and mapping flow was applied uniformly across all benchmarks to ensure a fair and consistent comparison.

A. Impact of OR Fan-in on Performance and Area

Tables I presents the experimental results of the proposed m-OIG synthesis flow on the ISCAS'85 and IWLS 2005 benchmarks, respectively. The results are organized to highlight how increasing the maximum allowed OR fan-in affects both the structural characteristics of the synthesized graphs and their execution cost when mapped onto RRAM-based crossbar architectures. For each benchmark, the tables report the number of primary inputs and outputs (PI/PO), reflecting the functional interface complexity of the circuit. Synthesis results are shown for 2-OIG, 3-OIG, and 4-OIG configurations. In each case, the column labeled *W.* denotes the number of write (initialization) cycles required to configure the resistive states of the crossbar prior to computation, while *M.* represents the number of MAC cycles needed to evaluate the logic function. The total delay is defined as the sum of write and MAC cycles and captures the end-to-end latency required to produce the output for a given Boolean function. The crossbar size (CBS), expressed as rows $\times$ columns, reflects the physical area footprint of the mapping.

Across both benchmark suites, a consistent trend can be observed: increasing the OR fan-in reduces the logic depth of the graph. Since OR evaluation in MAC-oriented crossbar architectures can be performed in parallel with a cost that is largely independent of fan-in, this reduction in depth directly translates into lower execution latency. Consequently, the total delay decreases when moving from 2-OIG to 3-OIG and further to 4-OIG for the majority of benchmarks. This effect is particularly pronounced for structurally complex circuits such as `c3540`, `c5315`, and `c2670`, where wide OR structures naturally arise and can be preserved rather than decomposed into binary trees.

In addition to latency improvements, higher OR fan-in also leads to substantial reductions in crossbar size. By consolidating multiple OR operations into a single node, the number of intermediate signals that must be stored and propagated is reduced, which in turn lowers the required number of crossbar rows and columns. On average, transitioning from 2-OIG to 4-OIG achieves a crossbar size reduction of **19.79%** for the ISCAS'85 benchmarks and **30.53%** for the IWLS 2005 benchmarks, while simultaneously reducing the total delay by **11.33%** and **18.90%**, respectively.

For small or highly regular circuits, such as `c17` or `xor5`, the impact of increasing OR fan-in is naturally limited. In these cases, the Boolean structure does not expose sufficiently wide OR operations, and the synthesized graphs are already close to optimal under the 2-OIG representation. Importantly, no performance degradation is observed, indicating that the proposed approach does not introduce unnecessary overhead when higher fan-in cannot be exploited.

B. Comparison with m-AIG Baseline

lTable II compares the proposed m-OIG approach with the m-AIG baseline [14] using a fixed fan-in parameter of $m = 4$. The table reports the decomposition of execution cost into MAC and initialization cycles for m-OIG, and read and computation cycles for m-AIG.

Evaluation on benchmarks provided with EPFL shows that the proposed m-OIG method achieves an average reduction of **32.9%** in total execution cycles. The improvements are particularly pronounced for control- and arithmetic-intensive circuits such as `sqrt32`, `usb_phy`, and `tv80`, where reductions in logic depth directly shorten the critical execution path.

Overall, these results highlight a fundamental complementarity between m-OIGs and m-AIGs. While m-AIGs primarily optimize AND-dominated structures, the proposed m-OIG framework targets OR-dominated accumulation patterns that are central to MAC-based in-memory computing. By aligning the graph representation with the execution semantics of RRAM crossbars, m-OIGs expose additional parallelism and reduce unnecessary serialization, resulting in consistent improvements in both area and execution latency.

IV. Conclusion

This paper studies the impact of OR fan-in on MAC-based logic execution in RRAM crossbar architectures using Multi-input OR–Inverter Graphs (m-OIGs). The key idea is to better match logic representation to the accumulation-based execution model by allowing OR nodes with fan-in greater than two. Since OR evaluation in MAC-oriented crossbars can be performed in parallel with little dependence on fan-in, preserving wide disjunctions avoids unnecessary decomposition and reduces graph depth . A synthesis and mapping flow based on m-OIGs is presented, along with an execution model capturing initialization cycles, MAC cycles, and crossbar size. Experimental results on standard benchmarks show that increasing OR fan-in reduces execution latency and crossbar resources, especially for circuits with wide OR structures, while introducing no overhead when such opportunities are limited.

Compared to m-AIG, m-OIG targets a complementary optimization space focused on OR-dominated logic suited to accumulation-based execution.

In general, fan-in–aware representations improve the efficiency of MAC-based in-memory logic execution without requiring modifications to the underlying hardware. While the proposed approach demonstrates the advantages of increasing OR fan-in, the current implementation is limited by the capabilities of existing synthesis tools, restricting the maximum fan-in to four inputs. Extending the synthesis framework to support higher fan-in is a promising direction for future work.

TABLE I: Impact of fan-in on crossbar size (CBS) and total delay on ISCAS'85 and IWLS 2005 benchmarks

	Benchmark		2-OIG				3-OIG				4-OIG				3-OIG vs 2-OIG		4-OIG vs 2-OIG	
	Name	#PI/PO	W.	M.	Delay	CBS	W.	M.	Delay	CBS	W.	M.	Delay	CBS	CBS (%)	Delay (%)	CBS (%)	Delay (%)
ISCAS-85	c17	5/2	6	3	9	16×6	6	3	9	16×6	6	3	9	16×6	0.00	0.00	0.00	0.00
	c432	36/7	126	22	148	318×126	126	22	148	318×126	107	17	124	280×107	0.00	0.00	25.22	16.22
	c499	41/32	398	17	415	814×398	398	17	415	814×398	382	16	398	782×382	0.00	0.00	7.51	4.10
	c880	60/26	273	23	296	614×273	273	23	296	614×273	260	22	282	588×260	0.00	0.00	8.74	4.73
	c1355	41/32	398	17	415	814×398	398	17	415	814×398	382	16	398	782×382	0.00	0.00	7.51	4.10
	c1908	33/25	390	28	418	796×390	353	23	376	722×353	340	22	362	696×340	17.71	10.05	23.63	13.40
	c2670	233/140	610	24	634	1466×610	522	16	538	1292×522	498	16	514	1242×498	24.53	15.14	30.85	18.92
	c3540	50/22	986	35	1021	2048×986	807	29	836	1690×807	752	27	779	1580×752	32.35	18.12	41.10	23.70
	c5315	178/123	1505	28	1533	3168×1505	1237	23	1260	2634×1237	1222	23	1245	2598×1222	31.59	17.81	33.52	18.78
	c6288	32/32	2113	90	2203	4226×2113	1874	88	1962	3748×1874	1873	88	1961	3746×1873	21.59	10.94	21.63	10.98
	c7552	207/108	1503	37	1540	3318×1503	1388	30	1418	3088×1388	1360	30	1390	3028×1360	14.13	7.92	17.24	9.74
AVG															**12.91**	**7.27**	**19.79**	**11.33**
IWLS-2005	con1f1	7/2	17	5	22	44×17	14	4	18	38×14	14	4	18	38×14	28.48	18.18	28.48	18.18
	exam1_d	3/1	6	4	10	16×6	6	4	10	16×6	6	4	10	16×6	0.00	0.00	0.00	0.00
	exam3_d	4/1	8	5	13	22×8	7	4	11	20×7	7	4	11	20×7	20.45	15.38	20.45	15.38
	max46_d	9/1	164	14	178	344×164	124	11	135	264×124	118	11	129	252×118	42.09	24.16	47.39	27.53
	newill_d	8/1	19	8	27	52×19	17	6	23	48×17	17	6	23	48×17	17.48	14.81	17.48	14.81
	newtag_d	8/1	9	5	14	32×9	6	4	10	26×6	5	4	9	24×5	45.83	28.57	58.33	35.71
	rd53f1	5/3	42	7	49	88×42	35	6	41	74×35	33	6	39	70×33	29.85	16.33	37.50	20.41
	rd73f1	7/3	112	11	123	232×112	100	9	109	208×100	94	9	103	196×94	19.83	11.38	27.10	16.26
	rd84f1	8/1	79	12	91	172×79	67	10	77	148×67	65	9	74	144×65	27.09	15.38	31.29	18.68
	sao2f1	10/1	36	9	45	90×36	29	8	37	76×29	24	7	31	66×24	31.96	17.78	51.11	31.11
	sym10_d	10/1	69	16	85	156×69	59	12	71	136×59	57	11	68	132×57	25.46	16.47	30.07	20.00
	t481_d	16/1	25	6	31	80×25	23	6	29	76×23	23	6	29	76×23	12.60	6.45	12.60	6.45
	xor5	5/1	13	7	20	34×13	13	7	20	34×13	13	7	20	34×13	0.00	0.00	0.00	0.00
AVG															**23.98**	**14.83**	**30.53**	**18.90**

TABLE II: Comparison between m-OIG and m-AIG ($m = 4$)

Bench	m-OIG			m-AIG [14]			Imp. (%)
	MAC	Init	Total	Read	Compute	Total	
i2c	14	967	981	627	289	916	-7.1
max	166	2653	2819	1610	1138	2748	-2.6
simple_spi	8	246	254	441	234	675	62.4
spi	24	2282	2306	1757	837	2594	11.1
sqrt32	53	850	903	1639	1401	3040	70.3
ss_pcm	5	141	146	107	107	214	31.8
square	246	16635	16881	17114	5150	22264	24.2
tv80	51	5323	5374	6175	2137	8312	35.3
usb_phy	7	106	113	271	118	389	71.0
Avg	63.8	3245	3309	3305	1268	4572	**32.9**

Benchmarks are provided by EPFL: http://lsi.epfl.ch/mig.

ACKNOWLEDGMENTS

This work was supported by the German Research Foundation (DFG) with project PLiM (DR 287/35-1, DR 287/35-2, and SH 1917/1-2), and by the German Ministry for Research, Technology, and Space (BMFTR) with project ExaVerse (grant number 01IW25003).

REFERENCES

[1] A. Sebastian, M. L. Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory devices and applications for in-memory computing," *Nature Nanotechnology*, vol. 15, pp. 529–544, 2020.

[2] A. Sinha, M. S. Kulkarni, and C. Teuscher, "Evolving nanoscale associative memories with memristors," in *2011 11th IEEE International Conference on Nanotechnology*, 2011, pp. 860–864.

[3] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams, "'memristive' switches enable 'stateful' logic operations via material implication," *Nature*, vol. 464, no. 7290, pp. 873–876, 2010.

[4] P.-E. Gaillardon, L. Amarú, A. Siemon, E. Linn, R. Waser, A. Chattopadhyay, and G. De Micheli, "The programmable logic-in-memory (PLiM) computer," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2016, pp. 427–432.

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

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

[7] 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.

[8] O. Krestinskaya, K. N. Salama, and A. P. James, "Learning in memristive neural network architectures using analog backpropagation circuits," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 2, pp. 719–732, 2018.

[9] L. Xia *et al.*, "Technological exploration of RRAM crossbar array for matrix-vector multiplication," *Journal of Computer Science and Technology*, vol. 31, no. 1, pp. 3–19, Jan 2016.

[10] F. Shirinzadeh, A. Kole, K. Datta, S. Shirinzadeh, and R. Drechsler, "A comprehensive synthesis and verification approach for RRAM-based neuromorphic computing," in *2025 28th Euromicro Conference on Digital System Design (DSD)*, 2025, pp. 531–538.

[11] F. Shirinzadeh, K. Datta, S. Shirinzadeh, A. Kole, and R. Drechsler, "Towards formal verification for MAC-based in-memory computing," in *2024 IEEE 33rd Asian Test Symposium (ATS)*, 2024, pp. 1–6.

[12] F. Shirinzadeh, A. Kole, K. Datta, S. Shirinzadeh, and R. Drechsler, "Fault-tolerant character recognition in neuromorphic systems using RRAM crossbar arrays," in *2025 IEEE Nordic Circuits and Systems Conference (NorCAS)*, 2025, pp. 1–7.

[13] F. Shirinzadeh, S. Nabipour, K. Datta, C. K. Jha, S. Shirinzadeh, and R. Drechsler, "Mac-based mapping of adder architectures to rram crossbars," in *2025 IEEE 7th International Conference on Emerging Electronics (ICEE)*, 2025, pp. 1–4.

[14] S. Froehlich, S. Shirinzadeh, and R. Drechsler, "Parallel computing of graph-based functions in ReRAM," *J. Emerg. Technol. Comput. Syst.*, vol. 18, no. 2, Jan. 2022.

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

[16] C. Wolf, "Yosys—a free verilog synthesis suite," in *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*, 2013.

[17] M. Hansen, H. Yalcin, and J. Hayes, "Unveiling the ISCAS-85 benchmarks: a case study in reverse engineering," *IEEE Design Test of Computers*, vol. 16, no. 3, pp. 72–80, 1999.

[18] C. Albrecht, "IWLS 2005 benchmarks," Tech. Rep., Jun. 2005.