Q-OIG: Quality-Constrained Approximate OIG Synthesis for MAC-Based Logic-in-Memory

Fatemeh Shirinzadeh* fatemeh.shirinzadeh@dfki.de
Saeideh Nabipour* saeideh.nabipour@dfki.de
Abhoy Kole* abhoy.kole@dfki.de
Kamalika Datta*‡ kdatta@uni-bremen.de
Saeideh Shirinzadeh*† saeideh.shirinzadeh@dfki.de
Rolf Drechsler*‡ drechsler@uni-bremen.de

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

***Abstract*—Approximate computing improves the efficiency of error-tolerant systems by trading bounded accuracy for reduced implementation cost. However, existing approximate logic synthesis techniques primarily target binary graph representations, leaving native multi-input OR-Inverter Graphs (OIGs) relatively unexplored for MAC-based logic-in-memory execution.**

This paper presents *Q-OIG*, a quality-constrained approximate synthesis framework that operates directly on native OIGs. Q-OIG combines lightweight local transformations with mapping-aware archive-based evolutionary search to minimize MAC execution cycles under prescribed quality constraints. Candidate circuits are optimized using stochastic simulation and independently validated. Experiments on ISCAS-85 and IWLS benchmarks demonstrate substantial cycle reductions under BER constraints. An RCA-8 case study further shows competitive cycle–error trade-offs against a recent MAGIC-based approximate adder in terms of MAE and MSE.

***Index Terms*—Approximate logic synthesis, OR-Inverter Graph, RRAM crossbar, MAC-based synthesis, evolutionary optimization**

I. Introduction

Approximate computing has become an effective design paradigm for error-tolerant digital systems, where bounded loss in computational accuracy can be traded for reductions in hardware cost, power consumption, and execution latency [1]. At the same time, *Resistive RAM (RRAM)*-based logic-in-memory architectures have attracted significant attention as a way to mitigate the von Neumann bottleneck by performing computation directly inside memory arrays. Besides established memristive logic styles such as *IMPLY*, *MAJ*, and *MAGIC* [2]–[5], RRAM crossbars naturally support *Multiply-and-Accumulate (MAC)* operations through current accumulation, enabling highly parallel Boolean computation.

Recent MAC-based logic-in-memory approaches have shown that Boolean functions can be evaluated with significantly lower execution latency than MAGIC- and MAJ-based implementations while preserving correctness through formal verification [6]–[9]. This execution model is particularly well matched to *OR-Inverter Graphs (OIGs)*, since multi-input OR nodes can be directly mapped to accumulation-oriented MAC operations. Preserving native higher fan-in OIG structures can reduce intermediate nodes, logic depth, and total MAC execution cycles. In contrast, conventional *AND-Inverter Graphs (AIGs)* require binary decomposition, which may introduce additional intermediate nodes and hide optimization opportunities that are visible in the original OIG representation.

Approximate Logic Synthesis (ALS) has been widely studied for conventional Boolean networks using AIG rewriting, approximate resubstitution, Boolean matrix factorization, and library-based replacement [1], [10]–[14]. While effective in CMOS-oriented flows, these techniques primarily rely on binary graph representations and do not directly exploit native multi-input OIG structures. Approximation has also been explored for memristive logic-in-memory, particularly for MAGIC-based approximate arithmetic using latency and numerical error metrics such as MAE and MSE [15], [16]. However, these works mainly target arithmetic designs rather than approximation-aware synthesis of general logic networks for MAC-based execution.

To address this gap, we propose *Q-OIG*, a quality-constrained approximate synthesis framework that operates directly on native multi-input OIGs. Q-OIG combines lightweight local OIG transformations with mapping-aware evolutionary search to reduce MAC execution cycles under prescribed quality constraints, with independent validation of the final candidates. The main contributions of this work are summarized as follows:

- We propose Q-OIG, a quality-constrained approximate synthesis framework that performs approximation directly on native multi-input OIGs, preserving structural opportunities relevant to MAC-based logic-in-memory execution.
- We develop a mapping-aware optimization flow that combines lightweight local OIG transformations with archive-based evolutionary search to minimize mapped MAC execution cycles under prescribed quality constraints, followed by independent validation for robust final candidate

selection.

- We evaluate Q-OIG on ISCAS-85 and IWLS benchmark circuits across different quality constraints, and further assess its arithmetic cost–quality trade-offs on RCA designs against recent MAGIC-based approximate adders.

The remainder of this paper is organized as follows. Section II introduces the required background on OIGs and MAC-based logic design. Section III presents the proposed Q-OIG framework. Section IV reports the experimental results, and Section V concludes the paper.

II. Background

A. OR-Inverter Graphs (OIG)

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

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

- 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 internal nodes implementing Boolean OR operations.
- A set of directed edges E, where 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 $v \notin V_X$.
- Each edge is represented as (u, v, p), where $p \in \{0, 1\}$ specifies edge polarity. A value of $p = 1$ denotes a regular edge, while $p = 0$ denotes a complemented edge.

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

In contrast to conventional binary graph representations, the OIG structure used in this work allows OR nodes with variable fan-in sizes of up to six inputs. Consequently, different OR nodes within the same graph may contain different fan-in sizes depending on the logic structure being represented. For example, a single OIG may simultaneously contain 2-input, 3-input, and 6-input OR nodes.

B. MAC-Based Logic Design Style

For MAC-based logic design, we follow the OIG-to-MAC mapping methodology introduced in [7]. In this flow, the input Verilog design is transformed into an OIG representation using conventional synthesis tools such as Yosys [17]. The OIG structure enables efficient realization of sum-of-products expressions suitable for MAC computation. The generated OIG is then levelized so that all gates belonging to the same logic level can be evaluated in parallel within a single MAC cycle.

During mapping, each OR gate is assigned to an individual column on the crossbar, while both inputs and their complements are assigned to dedicated rows. The computation proceeds level by level, with outputs from one level written back to rows and reused as inputs for subsequent levels. The primary outputs can then be observed as output currents on the corresponding crossbar columns. Following [7], we use the resulting total number of MAC execution cycles as the main latency-oriented cost metric throughout this work.

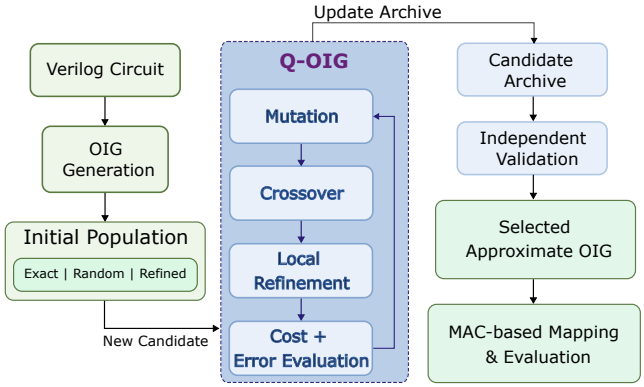


Fig. 1: Overview of the proposed Q-OIG synthesis framework

III. Q-OIG: Quality-Constrained Approximate OIG Synthesis

Approximate synthesis is performed directly on native multi-input OIGs, avoiding intermediate binary representations and preserving structural properties relevant to MAC-based logic-in-memory execution. Starting from an exact OIG, the proposed framework generates approximate candidates through lightweight graph transformations and explores the resulting design space using mapping-aware evolutionary search. Each candidate is evaluated by stochastic simulation and ranked according to its mapped total-cycle cost and approximation quality. An independent validation set is subsequently used to select the final implementation for each prescribed error bound. Fig. 1 presents an overview of the proposed Q-OIG synthesis flow, from initial OIG generation and evolutionary optimization to independent validation and MAC-based mapping.

A. Mapping-Aware Optimization Formulation

A central feature of the proposed framework is that approximation is guided by the execution cost of the mapped MAC-based implementation rather than by an intermediate graph metric alone. Given an exact OIG C_0, the framework searches for an approximate OIG C that minimizes mapped execution latency while satisfying a prescribed approximation bound. This formulation directly couples structural approximation with the execution characteristics of the target logic-in-memory architecture.

Let $\mathcal{C}(C_0)$ denote the design space reachable from C_0 through the proposed OIG rewriting operators. During design-space exploration, the optimization problem is formulated as

$$\min_{C \in \mathcal{C}(C_0)} \quad T_{\mathrm{cyc}}(C) \quad \text{s.t.} \quad \mathcal{E}(C_0, C; S_{\mathrm{opt}}) \leq \varepsilon, \tag{1}$$

where $T_{\mathrm{cyc}}(C)$ denotes the mapped execution cost in cycles, $\mathcal{E}(C_0, C; S_{\mathrm{opt}})$ is the approximation error estimated over the optimization sample set S_{opt}, and ε is the prescribed error

bound. Unless stated otherwise, $\mathcal{E}$ denotes the *Bit Error Rate* (BER), which is defined later in this section.

Following the micro-operation-level latency model of [7], the total-cycle cost of an OIG containing $N(C)$ internal nodes and having logic depth $D(C)$ is

$$T_{\mathrm{cyc}}(C) = N(C) + \begin{cases} 2D(C) - 1, & D(C) > 0, \\ 0, & D(C) = 0. \end{cases} \tag{2}$$

The term $N(C)$ captures the initialization cost associated with the internal OIG nodes, whereas $2D(C) - 1$ captures level-wise MAC evaluation and inter-level write-back. Eq. (2) highlights an important property of the proposed approach: a rewrite is beneficial not only when it removes internal nodes, but also when it shortens the OIG depth. This allows the mapping-aware optimizer to simultaneously reduce implementation cost and the number of sequential MAC operations, thereby improving both mapped structural cost and in-memory execution latency. Because different OIG structures with identical total-cycle cost may still require different hardware resources after mapping, we also estimate implementation cost using a fan-in-based cell proxy. When two candidates have the same total-cycle cost, the tie is broken using the fan-in-based cell proxy

$$\mathrm{Cell}(C) = \sum_{v \in V_H(C)} |\mathrm{fanin}(v)|\,, \tag{3}$$

which serves as a secondary objective. It counts the internal node-input incidences and provides a lightweight estimate of the node-input connections after mapping.

$$\mathrm{BER}(C_0, C; S) = \frac{1}{|S||Y|} \sum_{\mathbf{x} \in S} \sum_{j=1}^{|Y|} \mathbb{I}\left[y_j(\mathbf{x}) \neq \hat{y}_j(\mathbf{x})\right], \tag{4}$$

where Y denotes the set of primary outputs, $y_j(\mathbf{x})$ and $\hat{y}_j(\mathbf{x})$ are the exact and approximate values of output j for input vector $\mathbf{x}$, respectively, and $\mathbb{I}[\cdot]$ is the indicator function. Unlike an output-vector mismatch metric, BER measures the fraction of erroneous output bits and is therefore informative for multi-output circuits.

For arithmetic circuits, the framework additionally reports MAE and MSE:

$$\mathrm{MAE}(C_0, C; S) = \frac{1}{|S|} \sum_{\mathbf{x} \in S} \left|V(\mathbf{y}(\mathbf{x})) - V(\hat{\mathbf{y}}(\mathbf{x}))\right|, \tag{5}$$

$$\mathrm{MSE}(C_0, C; S) = \frac{1}{|S|} \sum_{\mathbf{x} \in S} \left(V(\mathbf{y}(\mathbf{x})) - V(\hat{\mathbf{y}}(\mathbf{x}))\right)^2, \tag{6}$$

where $V(\cdot)$ interprets an output bit vector as an unsigned integer. These metrics capture the numerical significance of output deviations, which cannot be represented by BER alone.

B. Local OIG Transformations

Approximate candidates are generated by applying lightweight local transformations directly to the OIG structure. Each transformation modifies a limited structural region, enabling efficient exploration without global Boolean reconstruction. The implemented transformations are:

- *Fan-in pruning*: removes one incoming edge from a selected multi-input OR node.
- *Node bypass*: removes a selected internal node and redirects its fan-outs to one of its fan-in nodes.
- *Input-based replacement*: replaces the fan-in set of a selected internal node with one or two primary-input literals, which may appear in regular or complemented form.
- *Polarity mutation*: complements the polarity of a selected incoming edge.
- *Constant replacement*: when enabled, replaces a selected internal node with logic zero.

After each transformation, a structural cleanup step removes logic that is no longer reachable from any primary output. This prevents obsolete nodes from contributing to the mapped cost and ensures that candidate evaluation reflects only the functionally active portion of the OIG. Candidate generation relies exclusively on localized graph transformations and avoids computationally expensive exact optimization procedures during the approximation search.

A mutation may comprise multiple transformations applied consecutively:

$$C' = \mathcal{M}_k \circ \mathcal{M}_{k-1} \circ \cdots \circ \mathcal{M}_1(C), \tag{7}$$

where $\mathcal{M}_i$ denotes a randomly selected local transformation and k is the number of transformations applied in the mutation. Compound mutation allows the search to move beyond the immediate one-transformation neighborhood while retaining the low computational cost of local OIG modification. Because these transformations intentionally modify circuit functionality, their acceptability is determined by subsequent error evaluation rather than by local equivalence checking.

C. Mapping-Aware Evolutionary OIG Search

The proposed search combines population-based evolutionary exploration with local structural refinement, following the general memetic optimization paradigm [18].

The initial candidate pool contains the exact OIG, a locally refined seed, and randomly transformed candidates derived from the exact network. This initialization provides both a competitive starting point and structural diversity without restricting subsequent population-based exploration. The exact OIG is also retained in the archive as a zero-error reference.

At each generation, candidates are classified according to their feasibility with respect to the prescribed approximation bound. A candidate C is considered optimization-feasible if

$$\mathrm{Feasible}_{\mathrm{opt}}(C) \iff \mathcal{E}(C_0, C; S_{\mathrm{opt}}) \leq \varepsilon. \tag{8}$$

Optimization-feasible candidates are always preferred over infeasible candidates. Among feasible candidates, lexicographic ranking is performed according to

$$R_{\mathrm{f}}(C) = \left(T_{\mathrm{cyc}}(C), \mathrm{Cell}(C), \mathcal{E}(C_0, C; S_{\mathrm{opt}})\right). \tag{9}$$

The mapped total-cycle cost is therefore the primary search objective, followed by the cell proxy and optimization error. Infeasible candidates may remain in the population for exploration, but they are never preferred over feasible candidates.

The best-ranked candidates are preserved through elitism. The remaining positions in the next population are filled using three complementary offspring-generation mechanisms:

1) mutation of a selected population member;
2) crossover between two parent candidates followed by mutation; and
3) restart mutation from the exact OIG to introduce candidates outside the neighborhood of the current population.

Mutation applies one or more of the local transformations introduced in Section III-B. Restart mutation generates a new candidate from the exact OIG rather than from the current population, periodically introducing structural characteristics that may have been lost during earlier generations.

Crossover transfers compatible node definitions between two parent OIGs. Because all candidates originate from the same exact network, surviving nodes retain their original identifiers. A node definition is transferred only when its required fan-in nodes remain available in the receiving candidate. After crossover, unreachable logic is removed and the offspring is further modified through mutation.

Each offspring undergoes bounded local refinement using the OIG transformations and a fixed number of neighboring candidates are generated sequentially. An evaluated neighbor replaces the current candidate if it improves the feasibility-cost ranking. This problem-specific refinement constitutes the local-search component of the memetic framework.

All evaluated candidates, including generated offspring and locally evaluated neighbors, are inserted into an archive $\mathcal{A}$. Unlike the active population, which changes across generations, the archive preserves candidates encountered throughout the search and supplies the candidate pool for the independent validation stage.

Algorithm 1 summarizes the interaction between population-based exploration, local refinement, and archive maintenance.

Algorithm 1: Mapping-Aware Evolutionary OIG Search

Input: Exact OIG C_0; error bound ε; optimization samples S_{opt}; population size P; offspring budget B
Output: Candidate archive $\mathcal{A}$

Initialize $\mathcal{Q}$ with C_0, a locally refined seed, and randomly transformed candidates
Evaluate $\mathcal{Q}$ on S_{opt} and set $\mathcal{A} \leftarrow \mathcal{Q}$, $\mathcal{P} \leftarrow \mathrm{Best}_P(\mathcal{A})$
while B *is not exhausted* **do**
  Classify candidates using Eq. (8) and rank $\mathcal{P}$ using feasibility-first, mapping-aware ranking
  Preserve elite candidates
  Generate offspring by mutation, crossover with mutation, or restart mutation from C_0
  Remove unreachable logic and evaluate offspring
  Apply bounded local refinement and evaluate generated neighbors
  Insert all evaluated candidates into $\mathcal{A}$ and update $\mathcal{P}$ with the best-ranked candidates
return $\mathcal{A}$

D. Independent Validation and Final Selection

The optimization sample set S_{opt} guides the evolutionary search, but feasibility on this set alone is insufficient for final selection. Because Monte Carlo estimates obtained from a finite sample set may not generalize to previously unseen inputs, selected archive candidates are re-evaluated using an independently generated validation set S_{val}.

Define the optimization-error estimate of candidate C as

$$e_{\mathrm{opt}}(C) = \mathcal{E}(C_0, C; S_{\mathrm{opt}}). \tag{10}$$

To bound validation cost while retaining candidates with different cost–quality characteristics, three lexicographic archive rankings are used. The cost-first ranking and the error-first ranking are

$$R_{\mathrm{cost}}(C) = \left(\phi(C), T_{\mathrm{cyc}}(C), \mathrm{Cell}(C), e_{\mathrm{opt}}(C)\right), \tag{11}$$

$$R_{\mathrm{err}}(C) = \left(\phi(C), e_{\mathrm{opt}}(C), T_{\mathrm{cyc}}(C), \mathrm{Cell}(C)\right), \tag{12}$$

where

$$\phi(C) = \begin{cases} 0, & e_{\mathrm{opt}}(C) \le \varepsilon, \\ 1, & e_{\mathrm{opt}}(C) > \varepsilon. \end{cases} \tag{13}$$

Thus, optimization-feasible candidates are prioritized in both rankings.

A third boundary-oriented ranking is defined as

$$R_{\mathrm{bnd}}(C) = \left(T_{\mathrm{cyc}}(C), \left|e_{\mathrm{opt}}(C) - \varepsilon\right|, \mathrm{Cell}(C)\right). \tag{14}$$

This ranking is not a Pareto-front computation. It prioritizes low-cost candidates and, for equal cost, candidates whose estimated error lies close to the prescribed bound. It can therefore include candidates estimated as infeasible on S_{opt}, allowing independent validation to correct for sampling variation. Let $\mathrm{Top}_{K_{\mathrm{val}}}(\mathcal{A}; R)$ denote the first K_{val} candidates in archive $\mathcal{A}$ under ranking R. In addition, let $\mathcal{A}_{\mathrm{env}}$ denote the monotone cost-error envelope obtained by first retaining, for each distinct total-cycle value, the candidate with the smallest optimization error, and then keeping candidates that improve or match the best error encountered as total-cycle cost increases. The set submitted to validation is

$$\begin{aligned} \mathcal{A}_{\mathrm{val}} = \mathrm{Unique}\Big(& \mathrm{Top}_{K_{\mathrm{val}}}(\mathcal{A}; R_{\mathrm{cost}}) \\ & \cup \mathrm{Top}_{K_{\mathrm{val}}}(\mathcal{A}; R_{\mathrm{err}}) \cup \mathrm{Top}_{K_{\mathrm{val}}}(\mathcal{A}; R_{\mathrm{bnd}}) \\ & \cup \mathcal{A}_{\mathrm{env}} \Big), \end{aligned} \tag{15}$$

where $\mathrm{Unique}(\cdot)$ removes candidates with duplicate recorded cost-error signatures. Consequently, at most $3K_{\mathrm{val}} + |\mathcal{A}_{\mathrm{env}}|$ distinct candidates are submitted to validation. Both optimization-feasible and selected optimization-infeasible candidates may be included.

Each candidate in $\mathcal{A}_{\mathrm{val}}$ is evaluated on S_{val}. Validation feasibility is determined using the constraint in Eq. (1), with

TABLE I: Exact OIG characteristics and validated Q-OIG total-cycle reductions for ISCAS/IWLS benchmarks.

Benchmark	Exact OIG [7]				Q-OIG: BER $\leq 10\%$			Q-OIG: BER $\leq 20\%$		
	Nodes	Levels	Cell proxy	T_{cyc}	T_{cyc}	Red.	BER	T_{cyc}	Red.	BER
c432	91	17	245	124	110	11.3%	0.095	80	35.5%	0.198
c499	374	17	946	407	76	81.3%	0.085	43	89.4%	0.189
c880	249	22	635	292	184	37.0%	0.098	53	81.8%	0.197
c1355	374	15	970	403	93	76.9%	0.096	43	89.3%	0.190
max46	109	12	287	132	51	61.4%	0.095	5	96.2%	0.120
rd53	40	6	88	51	37	27.5%	0.095	27	47.1%	0.166
rd73	92	11	220	113	79	30.1%	0.093	42	62.8%	0.198
rd84	61	9	151	78	65	16.7%	0.090	53	32.1%	0.200
Average reduction						**42.8%**			**66.8%**	

S_{val} replacing S_{opt}.

Among all validation-feasible candidates, the implementation with the smallest mapped total-cycle cost is selected. When multiple candidates have the same total-cycle cost, the candidate with the smaller cell proxy is preferred. If both quantities are equal, the candidate with the lower validation BER is selected.

This separation reduces sensitivity to the optimization samples used during evolutionary exploration and provides a more reliable basis for final candidate selection. The reported implementation therefore minimizes the mapping-aware cost among the validation-feasible candidates while satisfying the prescribed approximation bound on the independent validation sample set.

IV. Experimental Results

A. Experimental Setup

Q-OIG was evaluated on eight ISCAS-85/IWLS [19], [20] combinational benchmarks and an 8-bit ripple-carry adder (RCA-8). The framework was implemented in Python 3.11 and executed on a Linux server equipped with Intel Xeon E5-2630 v3 processors. Input vectors were generated independently from a uniform random distribution.

For the logic benchmarks, Q-OIG used population size $P = 24$, four elite candidates, and an offspring budget of $B = 500$. Optimization and validation used 2,000 and 20,000 input vectors, respectively. BER bounds of 5%, 10%, and 20% were evaluated using three seeds. The reported operating point for each benchmark and bound is the lowest-cost validation-feasible candidate obtained across the seeds.

For RCA-8, separate searches under prescribed MAE and MSE bounds generated the corresponding cost–quality trade-offs, using $P = 24$, four elite candidates, an offspring budget of $B = 1200$, 5,000 optimization vectors, and 50,000 validation vectors. The selected candidates were subsequently re-evaluated on 100,000 independently generated vectors to obtain the reported MAE and MSE values. Total-cycle cost was computed using Eq. (2).

Three local-refinement attempts were applied per offspring, and $K_{\mathrm{val}} = 40$ candidates were retained from each archive ranking.

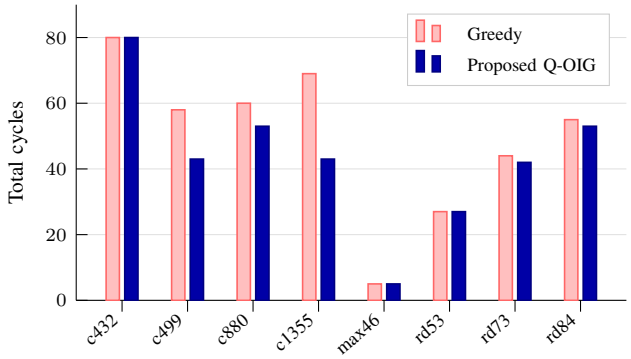


Fig. 2: Best validated total-cycle counts obtained across three seeds for the greedy baseline and Q-OIG under BER $\leq 20\%$

B. ISCAS/IWLS Logic Evaluation

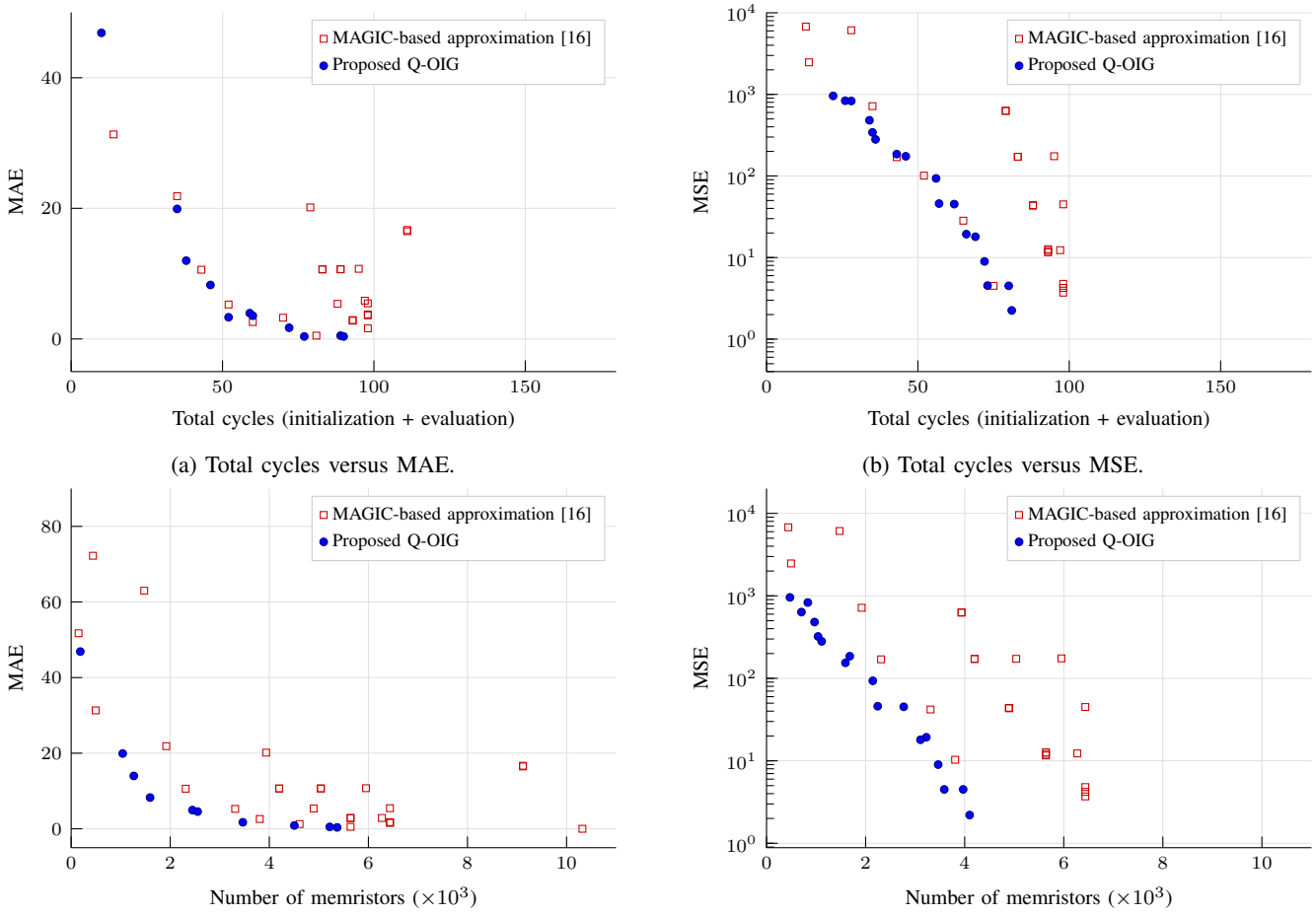
Table I reports the structural and mapping characteristics of the exact OIG benchmarks before approximation. The node count and logic level indicate the size and depth of the native OIG network, respectively. The cell proxy provides an implementation-cost indicator for the mapped MAC-based realization, while the exact T_{cyc} gives the total number of MAC execution cycles required by the exact design. These exact implementations serve as the reference point for the approximate designs generated by Q-OIG.

Table I reports the best validated approximate implementation under BER budgets of 10% and 20%. The proposed flow provides controllable cost-quality trade-offs: tighter budgets preserve more structure, while relaxed budgets enable larger T_{cyc} reductions. Average T_{cyc} reduction increases from 42.8% at BER $\leq$ 10% to 66.8% at BER $\leq$ 20%.

Fig. 2 compares the proposed method with the greedy baseline at the 20% BER operating point. The proposed search matches or improves the greedy solution for every benchmark. Across all 72 benchmark/budget/seed cases, covering BER bounds of 5%, 10%, and 20% and three random seeds, Q-OIG improves the validated T_{cyc} result in 53 cases and matches it in the remaining 19 cases.

C. RCA-8 Arithmetic Evaluation

The RCA-8 experiment compares Q-OIG with the approximate functions underlying the published MAGIC Pareto front [16]. For a common implementation basis, the baseline



(a) Total cycles versus MAE.

(b) Total cycles versus MSE.

(c) Number of memristors versus MAE.

(d) Number of memristors versus MSE.

Fig. 3: Comparison of Q-OIG and MAGIC-derived RCA-8 designs under the common OIG/MAC mapping: total cycles and memristor counts versus MAE and MSE.

functions were resynthesized into OIGs and evaluated using the same OIG/MAC mapping model as Q-OIG. Thus, Fig. 3 compares both total cycles and mapped memristor count against MAE and MSE under a common target representation. The exact OIG-based RCA-8 requires 95 MAC cycles [9].

Overall, Q-OIG provides competitive cost–error trade-offs in the plotted comparisons. For MAE, Q-OIG achieves 0.3761 at 77 cycles, compared with 0.5068 at 81 cycles for a plotted MAGIC-derived design. For MSE, Q-OIG achieves 2.2485 at 81 cycles, compared with 3.7117 at 98 cycles for a plotted MAGIC-derived design. These operating points improve both numerical error and cycle cost relative to the respective comparison designs. The memristor-count plots additionally illustrate resource–error trade-offs under the common OIG/MAC mapping.

V. Conclusion

This work introduced Q-OIG, a quality-constrained approximate synthesis framework that operates directly on native multi-input OIGs. Q-OIG combines local structural transformations with mapping-aware evolutionary search to reduce MAC execution cost under prescribed quality constraints, with independent validation for final candidate selection.

Across the evaluated ISCAS-85 and IWLS circuits, Q-OIG reduced the average total-cycle count by 42.8% under a 10% BER constraint and by 66.8% under a 20% BER constraint. For RCA-8, Q-OIG also demonstrated favorable cycle–error and mapped memristor-count–error trade-offs relative to the MAGIC-based approximation after both design sets were evaluated under a common OIG mapping basis. These results demonstrate the potential of approximate multi-input OIG synthesis for efficient MAC-based logic-in-memory execution.

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] D. Shin and S. K. Gupta, “Approximate logic synthesis for error tolerant applications,” in *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, 2010, pp. 957–960.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

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

[7] 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.

[8] F. Shirinzadeh, K. Datta, A. Kole, S. Shirinzadeh, and R. Drechsler, "An HDL-centric synthesis and verification approach for RRAM-based neuromorphic systems," *Microprocessors and Microsystems*, p. 105278, 2026.

[9] F. Shirinzadeh, S. Nabipour, K. Datta, C. Kumar 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.

[10] A. Chandrasekharan, M. Soeken, D. Große, and R. Drechsler, "Approximation-aware rewriting of AIGs for error tolerant applications," in *International Conference on Computer-Aided Design*, 2016, pp. 1–8.

[11] S. Venkataramani, K. Roy, and A. Raghunathan, "Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits," in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2013, pp. 1367–1372.

[12] S. Hashemi, H. Tann, and S. Reda, "BLASYS: Approximate logic synthesis using Boolean matrix factorization," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2018, pp. 1–6.

[13] M. Barbareschi, S. Barone, N. Mazzocca, and A. Moriconi, "A catalog-based aig-rewriting approach to the design of approximate components," *IEEE Transactions on Emerging Topics in Computing*, vol. 11, no. 1, pp. 70–81, 2023.

[14] C. Meng, A. Mishchenko, W. Qian, and G. De Micheli, "Efficient resubstitution-based approximate logic synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 44, no. 6, pp. 2040–2053, 2025.

[15] C. K. Jha, S. Ahmadi-Pour, and R. Drechsler, "Input distribution aware library of approximate adders based on memristor-aided logic," in *International Conference on VLSI Design and International Conference on Embedded Systems (VLSID)*, 2024.

[16] S. Nabipour, C. K. Jha, S. Shirinzadeh, and R. Drechsler, "Approximated MAGIC-ReRAM adder circuits for low-latency in-memory computing," in *International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, 2026.

[17] C. Wolf, "Yosys open synthesis suite," http://www.clifford.at/yosys/.

[18] P. Moscato, "On evolution, search, optimization, genetic algorithms and martial arts: Towards Memetic algorithms," California Institute of Technology, Pasadena, California, Tech. Rep. C3P 826, 1989.

[19] 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.

[20] C. Albrecht, "IWLS 2005 benchmarks," International Workshop on Logic Synthesis (IWLS), 2005, available: http://iwls.org/iwls2005/benchmarks.html.