Input Distribution Aware Library of Approximate Adders Based on Memristor-Aided Logic

Chandan Kumar Jha[ψ], Sallar Ahmadi-Pour[ψ], and Rolf Drechsler[ψ,†]
Institute of Computer Science, University of Bremen, Bremen, Germany[ψ]
German Research Center for Artificial Intelligence, DFKI GmbH, Bremen, Germany[†]
chajha@uni-bremen.de, sallar@uni-bremen.de, and drechsler@uni-bremen.de

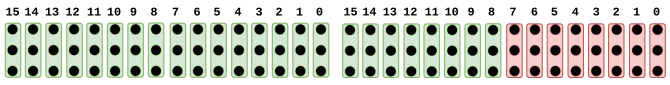
***Abstract*—Memristor-aided logic (MAGIC) is one of the most popular design styles for implementing Logic-in-Memory (LiM) using memristors. MAGIC-based LiM has been shown to be useful for high throughput applications, as the same design can be mapped to multiple rows of a crossbar. The computations can be performed simultaneously on all these rows on different inputs to achieve high throughput. In recent years, approximate circuits have been extensively explored to obtain benefits in power, performance, and area for traditional CMOS designs. While these approximate circuits produce erroneous outputs, several applications can produce an acceptable quality output even with these approximations. In this work, we propose approximate circuits for MAGIC-based LiM. We have used the Ripple Carry Adder (RCA) architecture and introduced functional approximation to generate the approximate RCAs. We obtained the mapping of these approximate RCAs on the memristor crossbar using the SIMPLER MAGIC tool. We generated the Pareto-optimal designs using Gate Count and Total Cycles as design metrics against Mean Square Error and Mean Absolute Error as error metrics. We generated separate Pareto-optimal designs for three different input data distributions namely uniform, exponential, and normal data distributions.**

***Index Terms*—Approximate computing, memristor-aided logic, logic-in-memory, pareto-optimal, input distribution, adders**

I. Introduction

Memristors are two terminal devices that can change their resistance in response to the applied voltage across their terminals [1], [2]. Memristors can be configured to a low resistance state ('1') or a high resistance state ('0') depending upon the magnitude of the applied voltage [3]. In addition to storage, these devices are capable of performing both analog and digital computations [2], [4]. In this work, we focus on using memristors to perform digital *Logic-In-Memory* (LiM) computations [3]. There are a number of different design styles to perform stateful LiM computations using memristors crossbars [5]–[7]. We have used one of the popular *Memristor-Aided Logic* (MAGIC) design styles in this work [6], [8]. MAGIC design style-based NOR and NOT operations can be mapped to a memristor crossbar. Since NOR is a universal gate, any Boolean function can be implemented using the MAGIC design style [9]. There have been recent works that have looked into comparing various arithmetic architectures for memristors crossbars [10]–[12].

In this work, our focus is on approximate computing for MAGIC-based LiM. Approximate computing deals with the introduction of approximation across the computing stack to improve design metrics such as power, performance, and area as the cost of acceptable errors in the result [13], [14]. Some applications have been shown to be resilient towards the introduction of such approximations, i.e., these applications produce acceptable quality output in the presence of these approximations [15]. In this work, we deal with the circuit-level approximation technique called functional approximation. Functional approximation means replacing the Boolean function of the exact circuit with another Boolean function to obtain benefits in the design metrics.



(a) Exact Adder **(b)** Approximate Adder

Figure 1: Approximation in Adder Designs (Exact Adders: Green; Approximated Adders: Red)

Owing to the benefits of approximate computing on traditional CMOS designs, some works have also investigated the benefits of using MAGIC-based LiM domain [16]–[18]. However, only one work exists that deals with the design of approximate adders LiM using the MAGIC design style [18]. In this work, the Pareto-optimal adder designs are obtained by mapping the design to the minimum number of memristors using MAGIC based LiM [19]. This method suffers from the limitation of not being able to use all the available memristors effectively. This happens because the mapping obtained focuses on minimizing the number of memristors, even though additional memristors are available. However, in our work, we use the SIMPLER MAGIC tool that focuses on efficiently mapping the design for a given number of memristors (not necessarily minimum) [20]. This becomes useful for cases where we want to utilize the available crossbar of a given size more effectively, thus alleviating the limitation of the prior work [18], [19]. In addition to it, we also wanted to generate the Pareto-optimal design for varying input data distributions. Our motivation behind this was to generate tailored approximate circuits that can benefit from the information about the input data distribution of the application.

Following are the contributions of our work:

1) We propose the first input-aware library of approximate adders based on the MAGIC-based design style.
2) We obtained the Pareto-optimal designs from among

millions of functional approximate adder designs for each distribution.

3) We also show how the Pareto-optimal designs vary with the input-data distributions and can be exploited in applications where the input data distribution is known.

II. Background

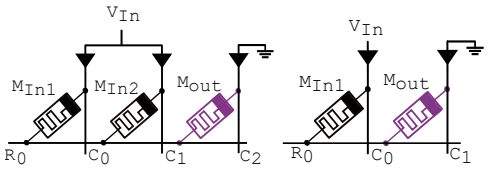
In this section, we will discuss functional approximation, NOR gate and NOT gate implementation using the MAGIC design style, and mapping of MAGIC-based gates to the memristor crossbar for *Single Instruction Multiple Data* (SIMD) architecture.

A. Functional Approximation

In functional approximation, as the name suggests, the Boolean function of the exact circuit is replaced with another Boolean function, that reduces the overall design cost metrics with the introduction of error in the output. The replacement of the Boolean function can be done depending on the required constraints in the design metric as well as the error metric. An example of a functional approximation in a *Ripple Carry Adder* (RCA) will be to replace the Boolean function of $SUM = A \oplus B \oplus C$ with Boolean function $SUM = A \oplus B$ of some the full adders. The approximated Boolean function requires only one XOR operation as compared to the two required in the original Boolean function which helps in reducing the design metric costs. However, since the approximated Boolean function is independent of the input C, for the cases when the output is dependent on C, we will get erroneous output. The entire goal of functional approximation is to identify the approximate Boolean functions that will give the maximum benefits in the design metrics for a given error constraint. An example showing a 16-bit exact adder and the approximate adder with the least significant 8-bit approximated is shown in Fig. 1.

B. MAGIC Design Style

MAGIC design style uses only memristors to perform computations using memristors. MAGIC design style is stateful as the inputs are represented as resistance values and the outputs after the computations are also represented as resistance values. MAGIC design style-based NOT and NOR gates can be mapped to a memristor crossbar as shown in Fig. 2. The NOR gate and the NOT gate require three and two memristors respectively. For the NOR operation, the output memristor is initialized to a low resistance state, and depending on the inputs the two input memristors are initialized. To perform the operation the input memristors are connected to V_{In} and the output memristor is connected to the ground as shown in Fig. 2. If at least one of the input memristors is in the low resistance state (‘1’), the output memristor switches to the high resistance state (‘0’). In the case, where both the input memristors are in a high resistance state (‘0’) the output memristor remains in the low resistance state (‘1’). Hence, the NOR operation is achieved using the three memristors. The NOT operation is similar to the NOR operation with the exception that only one input memristor is required.



(a) NOR gate **(b)** NOT gate

Figure 2: MAGIC Design Style Based gates

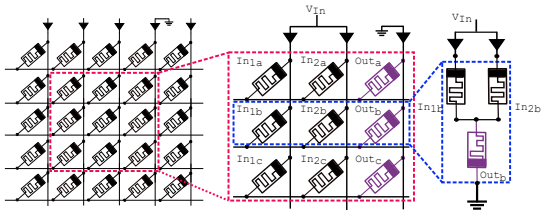


Figure 3: SIMD Implementation Using MAGIC

C. SIMD Operations

The individual MAGIC NOR and MAGIC NOT gates were discussed in Section II-B. The same design is mapped to all the rows of the crossbar to increase the throughput. Each design in the different rows of the crossbar can operate on different input data. For example, let us assume we have a crossbar of size 256×256, and an adder can be mapped to one row of the memristor crossbar. This adder can be replicated across the 256 rows of the crossbar. Hence at a time, 256 additions can be done using the 256 rows. Thus, it is useful in applications that can exploit the SIMD architecture. In Fig. 3, we show the snippet of three NOR operations being executed on the three rows of the crossbar.

III. Methodology

The overall library generation methodology is shown in Fig. 4. It consists of four main stages. We will discuss each of these stages in detail in the following subsections.

A. Approximate Adder Generation

We have used functional approximation to generate the approximate RCA design. The RCA consists of full adders which have three inputs and two outputs. The Boolean function for SUM and $CARRY$ of each full adder can be computed as $A \oplus B \oplus C$ and $AB + BC + CA$ respectively. In functional approximation, these Boolean functions are replaced with another Boolean function to generate the approximate adder designs. Rather than evaluating a subset of the approximate Boolean function, we exhaustively generated all the possible Boolean functions that can be generated for the SUM and $CARRY$ similar to [18]. There are three inputs hence the SUM and $CARRY$ can be each generated in 256 ways, so in combination we generate a total of 65536 designs single-bit approximate adder designs. Since we use 8-bit and 16-bit ripple carry adder designs, another dimension of approximation deals with how many adders will be replaced with approximate adders. In this work for the 8-bit RCA, we start with the replacement from the least significant bit in steps of one, i.e.,

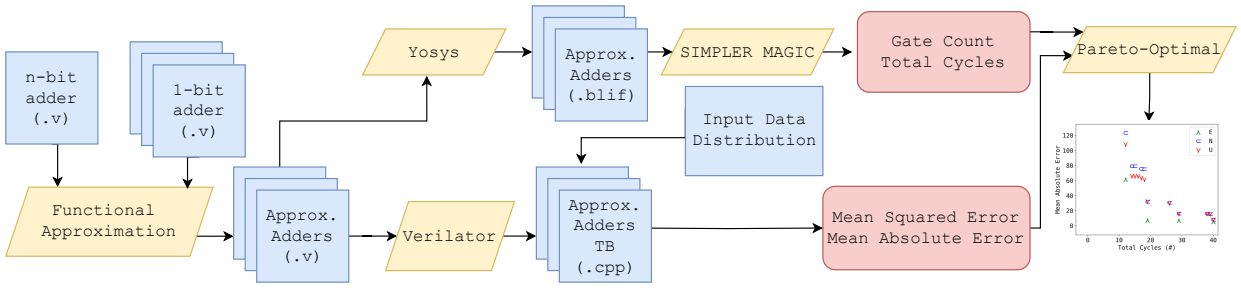


Figure 4: Overall Methodology for Generation of Library

1, 2, . . . 7, and similarly for the 16-bit from 1, 2, . . . 15. For every bit of approximation we have 65536 designs, i.e., for the example shown in Fig. 1, the 8 approximated least significant bit adders can be any of the 65536 approximate designs. We want to highlight to limit the design space from exploding, all the approximated adders are of the same type. We generated 458752 and 983040 approximate RCA designs for 8-bit and 16-bit respectively.

B. Mapping to Memristor Crossbars

To obtain the mapping of the approximate adder designs the open source SIMPLER MAGIC tool was used [20]. The Verilog designs were first converted to the *Berkeley Logic Interchange Format* (blif) format as the SIMPLER MAGIC tool requires it to be in this format. The blif files were used as input designs for the SIMPLER MAGIC tool. In [10], it was shown that the exact 8-bit and 16-bit RCA designs require 50 and 75 memristors respectively. We wanted to use typical sizes in which the memristor crossbars are fabricated. Hence, we selected 64 memristor count for 8-bit RCA and 128 for 16-bit RCA. The SIMPLER MAGIC tool gives the Gate Count, i.e., the number of NOR and NOT gates, and Total Cycles, i.e., the number of cycles required to obtain the final output. We have used these two as the design metrics in this work. SIMPLER was not able to correctly map some approximate adder designs we have removed such designs from the comparisons.

C. Error Computation

As discussed in Section III-A, millions of adder designs exist in the entire design space. The error computations require simulation of the design which is quite slow using Verilog designs. To alleviate this we used the Verilator tool to convert the Verilog designs to the C++ designs. This greatly improves the simulation time. This benefit adds up as we use three input data distributions namely normal, uniform, and exponential. We have used the *Mean Absolute Error* (MAE) and the *Mean Square Error* (MSE) as the error metrics in our work as shown in Equation (1).

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - x_i|; \quad MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - x_i)^2 \quad (1)$$

D. Pareto-Optimal Designs

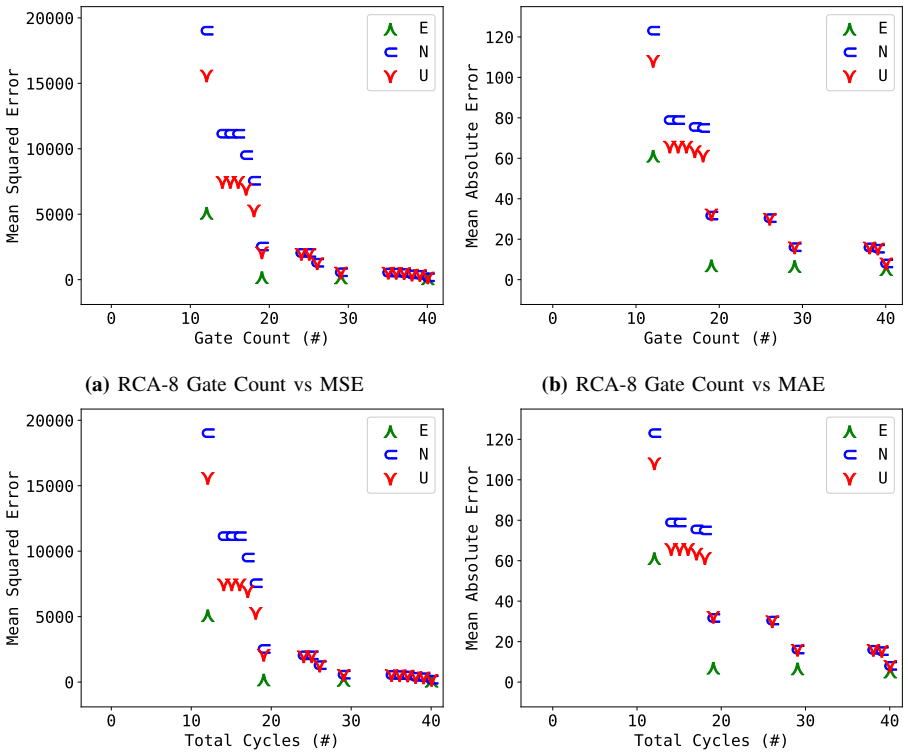
Lastly, we need to obtain the set of the Pareto-optmial designs. These designs can be explained as the RCA designs having the best design metric for a given error metric or the RCA design having the best error metric for a given design metric. We take the design metrics namely Gate Count and Total Cycles, and the error metrics namely MAE and MSE for all the adder designs. We use the Python Numpy library and the py-paretoarchive library to obtain the set of Pareto-optimal designs [21], [22]. We perform this analysis for each of the input data distributions to highlight its impact on the obtained Pareto-optimal designs. Since we have two design and two error metrics we perform four such analyses each for 8-bit and 16-bit RCA.

IV. Results and Discussion

In this section, we discuss the results in detail of the design space exploration of the functional approximate RCA designs using the methodology discussed in Section III.

In Fig. 5 and Fig. 6 the obtained Pareto-optimal designs for 8-bit and 16-bit RCA designs over the three input distributions, namely exponential, normal and uniform, are shown, respectively. The figures are structured such that, rows contain a specific design metric (Gate Count in the first row, Total Cycles in the second row), while columns contain a specific error metric (MSE in the first column, MAE in the second column). Thus, Figs. 5(a), 5(b), 6(a) and 6(b) show the Gate Count vs error metric, while Figs. 5(c), 5(d), 6(c) and 6(d) show the Total Cycles vs error metric. To further break down the illustration, each figure's caption contains an identifier for the circuit, the design metric, and the error metric. For example, Fig. 5(b) contains the Pareto-optimal designs that were obtained, optimizing the Gate Count against the MAE. In particular, each plot shows the Pareto-optimal designs determined for a given metric pair (e.g. Gate Count vs MAE in Fig. 5(b)) for the three cases, in which input samples for error calculation are sampled from exponential, normal and uniform distributions respectively. Each distribution is shown by its own respective marker in the plot, with ⅄ (green) marking the exponential, ⊂ (blue) the normal and ϒ (red) the uniform distribution respectively. The x-axis shows the design metric (Gate Count, Total Cycles) while the y-axis shows the error metric (MSE, MAE).

Section IV-1 discusses the Pareto-optimal designs shown in Fig. 5 and Fig. 6 further, while IV-2 examines the commonly and uniquely found Pareto-optimal designs across the different input distributions in this work's context.



(a) RCA-8 Gate Count vs MSE

(b) RCA-8 Gate Count vs MAE

(c) RCA-8 Total Cycles vs MSE

(d) RCA-8 Total Cycles vs MAE

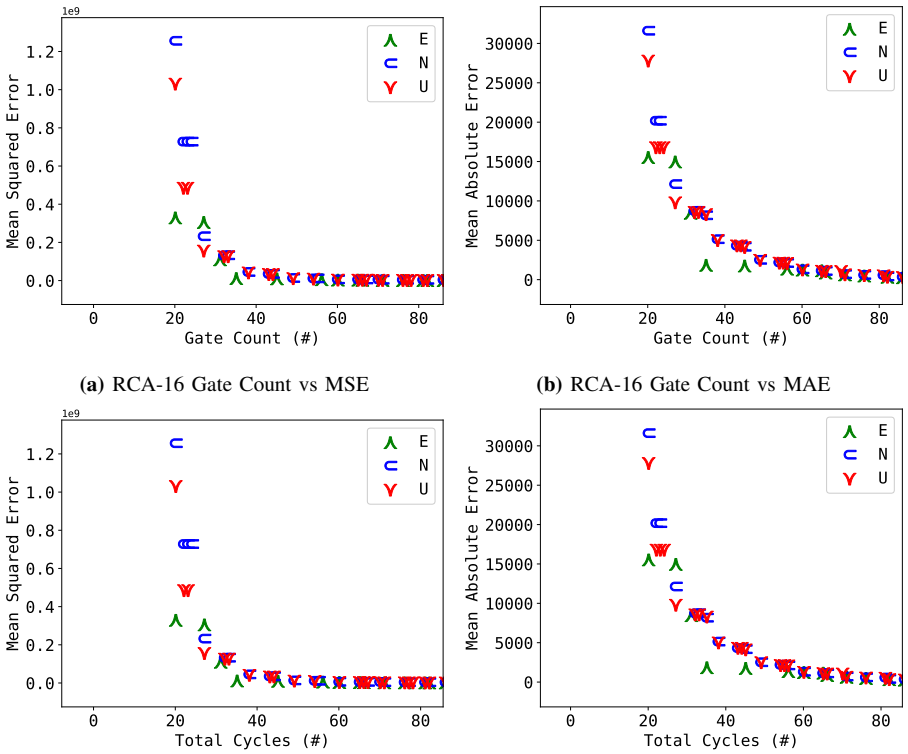
Figure 5: Comparison of all Pareto optimal solutions for each input distribution for 8-bit Ripple Carry Adder circuits. Each row shows the designs for both error metrics (MSE and MAE) for a given design metric, respectively.

1) Pareto-optimal designs: This section discusses the details of the determined Pareto-optimal results as shown in Fig. 5 and Fig. 6. As the methodology also compares the impact of the chosen distribution to sample the inputs for the simulations, a discussion on the effect on the obtained Pareto-optimal set will also be discussed. First, we provide a general overview of the 8 plots containing the Pareto-optimal sets.

Across all plots for the 8-bit RCA, it can be seen that more aggressive optimization for the design metric (e.g. lower amount of Total Cycles) with the exponential distribution yields designs with lesser error. This can be attributed to the fact that most of the sampled inputs in the exponential distribution are smaller, thus the calculated results are smaller, also resulting in lower error values. Hence, through the exponential distribution, the design space is explored on lower errors, yielding other designs.

In Figs. 5(a) and 5(b), below a Gate Count of 40 exponential distribution shows a notable difference in error compared with normal and uniform respectively. With Gate Count below 35, differences between the normal and uniform distribution become evident too, with uniform distribution highlighting different designs with less error. In Fig. 5(c), with a total cycle amount of less than 30, exponential distribution also shows a notable difference in error compared with normal and uniform, respectively. With Total Cycles less than 20, differences between normal and uniform distribution become more evident as well, with uniform distribution highlighting designs with less error again.

The observations made for 8-bit RCA are similar on 16-bit RCA with minor differences and a few exceptions. Inspecting Fig. 6(a), it can be seen for Gate Count less than 50 that the errors of the designs are notably different across distributions. For most design points, the exponential distribution presents designs with less error than the normal and uniform distribution, respectively. With an exception at around a Gate Count of 25, at which the exponential distribution highlights a design with a higher error than normal and uniform distributions. At a Gate Count less than 30, the differences between the designs provided by normal and uniform distribution become more evident, with uniform distribution providing designs with less error. Examining Fig. 6(c) differences in the distributions are notable for total cycle amounts of less than 40. Here too, most design points with less error are provided by the exponential distribution compared to the normal and uniform distribution. This too has an exception, at around a little less than 25 cycles, at which the exponential distribution provides a design with higher error than normal and uniform distribution respectively. At a total cycle number less than 30, differences between the normal and uniform distribution become notable, with the uniform distribution providing designs with less error. We want to emphasize that the Total Cycles are heavily related to Gate Count, hence similarities between the results are expected.



(a) RCA-16 Gate Count vs MSE

(b) RCA-16 Gate Count vs MAE

(c) RCA-16 Total Cycles vs MSE

(d) RCA-16 Total Cycles vs MAE

Figure 6: Comparison of all Pareto optimal solutions for each input distribution for 16-bit Ripple Carry Adder circuits. Each row shows the designs for both error metrics (MSE and MAE) for a given design metric, respectively.

It is important to highlight, that different distributions for the choice and sampling of inputs are not about which distribution performs better in terms of the error. The different input distributions highlight a particular set of circuits in the design space, which gives engineers a more faithful choice of circuits for the application the memristor-based approximate circuit is tailored for. Being able to assess the design space is fundamental in order to effectively trade-off between designs in the context of the required application.

2) Common and unique designs: Lastly, we examine the commonly and uniquely obtained circuit designs between different Pareto-sets for each metric pair (e.g. Gate Count vs MAE). This is of importance, as the Pareto-optimal sets discussed in IV-1 focus on qualitative differences in the design space. The listing and comparison of commonly and uniquely found designs across the input distributions allows for a quantitative assessment of our results. Hence, further differences are emphasized, supporting the case for the need for such a systematic methodology employing input distribution.

In Table I unique and common designs across input distribution are presented for each metric pair examined in this paper. The table is split into three parts. The leftmost part shows the circuit and the respective bit-width examined. The two columns right of that contain cells with the number of designs in the Pareto-optimal set that are common and unique to the input distribution for a design metric (from left to right: Gate Count, Total Cycles). For each column with a design metric, the table contains the error metrics (MSE, MAE). Each cell in the table contains the four respective amounts of (C)ommon designs found with each input distribution and those uniquely determined by (E)xponential, (N)ormal and (U)niform distribution respectively. Take the 8-bit RCA for Gate Count vs MAE as an example, with the cell containing C: 11, E: 1, N: 1, and U: 2. This means the Pareto-optimal sets have 11 designs in common across all distributions, 1 design uniquely found by exponential distribution, 1 design uniquely found by normal distribution, and 2 designs found by uniform distribution.

From the table, we can observe, that with each distribution unique designs are determined, while each Pareto-optimal set contains commonly found circuits in the design space. The case of RCA-16-GateCount-MSE and RCA-16-TotalCycles-MAE contains the most commonly found designs (C:35), while RCA-8-GateCount-MAE and RCA-8-TotalCycles-MAE have the lowest amount of commonly found designs (C:11). For the exponential distribution, RCA-16-GateCount-MSE and RCA-16-TotalCycles-MSE report the highest amount of unique designs (E:11), while all metrics pairs of RCA-8 show a single unique design (E:1), respectively. For the normal distribution, RCA-16-GateCount-MSE and RCA-16-TotalCycles-MSE have the highest number of unique designs (N:9), while RCA-8-GateCount-MAE and RCA-8-TotalCycles-MAE

Table I: Number of common (C) and unique design points for input samples from exponential (E), normal (N), uniform (U) distribution.

Circuit		Gate Count				Total Cycles			
		MSE		MAE		MSE		MAE	
RCA	8-bit	C: 13	E: 1	C: 11	E: 1	C: 13	E: 1	C: 11	E: 1
		N: 4	U: 5	N: 1	U: 2	N: 4	U: 5	N: 1	U: 2
	16-bit	C: 35	E: 11	C: 33	E: 2	C: 35	E: 11	C: 33	E: 2
		N: 9	U: 11	N: 2	U: 5	N: 9	U: 11	N: 2	U: 5

contain the lowest number of unique designs (N:1). For the uniform distribution, RCA-16-GateCount-MSE and RCA-16-TotalCycles-MSE report the highest number of unique designs (U:11), while RCA-8-GateCount-MAE and RCA-8-TotalCycles-MAE contain the lowest number of unique designs (U:2).

In general, the results can be summarized in two points 1) Section IV-1 shows and *emphasizes the different circuits* found in the design space through different input distributions, and 2) Section IV-2 highlights the found *differences quantitatively* by a comparison of common and unique designs found between input distributions.

V. Conclusion and Future Work

We proposed the first input data-aware library of approximate RCA that can be mapped to a memristor crossbar. We used functional approximation to generate several approximate RCA designs and explored this design space to identify the Pareto-optimal designs. Our library consists of 8-bit and 16-bit approximate RCA that have been tailored for three different distributions namely uniform, exponential, and normal data distributions. We showed how the Pareto-optimal design space is dependent on the input data distributions as well as the design and error metrics. We believe that this work will act as a baseline to be used in research directions to perform approximate computing using MAGIC-based LiM on memristor crossbars. Hence we made the Pareto-optimal designs available at https://github.com/agra-uni-bremen/vlsid2024-inputaware-approxadders-magic. In the future, we plan to extend this work to the design of approximate multipliers for LiM.

Acknowledgements

This work was supported in part by the German Research Foundation (DFG) within the Project PLiM (DR 287/35-1, DR 287/35-2).

References

[1] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *nature*, vol. 453, no. 7191, pp. 80–83, 2008.

[2] M. Di Ventra, Y. V. Pershin, and L. O. Chua, "Circuit elements with memory: memristors, memcapacitors, and meminductors," *Proceedings of the IEEE*, vol. 97, no. 10, pp. 1717–1724, 2009.

[3] I. Vourkas and G. C. Sirakoulis, "Emerging memristor-based logic circuit design approaches: A review," *IEEE circuits and systems magazine*, vol. 16, no. 3, pp. 15–30, 2016.

[4] J. Reuben, N. Talati, N. Wald, R. Ben-Hur, A. H. Ali, P.-E. Gaillardon, and S. Kvatinsky, "A taxonomy and evaluation framework for memristive logic," *Handbook of Memristor Networks*, pp. 1065–1099, 2019.

[5] S. Kvatinsky, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based material implication (IMPLY) logic: Design principles and methodologies," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 10, pp. 2054–2066, 2013.

[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 Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 11, pp. 895–899, 2014.

[7] S. Gupta, M. Imani, and T. Rosing, "Felix: Fast and energy-efficient logic in memory," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–7.

[8] S. Singh, C. K. Jha, A. Bende, P. L. Thangkhiew, 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*, pp. 1–1, 2023.

[9] N. Talati, S. Gupta, P. Mane, and S. Kvatinsky, "Logic design within memristive memories using memristor-aided logic (MAGIC)," *IEEE Transactions on Nanotechnology*, vol. 15, no. 4, pp. 635–650, 2016.

[10] 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.

[11] O. Leitersdorf, D. Leitersdorf, J. Gal, M. Dahan, R. Ronen, and S. Kvatinsky, "AritPIM: High-throughput in-memory arithmetic," *IEEE Transactions on Emerging Topics in Computing*, 2023.

[12] C. K. Jha and R. Drechsler, "Benchmarking multiplier architectures for MAGIC based in-memory computing," in *2023 21st IEEE Interregional NEWCAS Conference (NEWCAS)*. IEEE, 2023, pp. 1–5.

[13] H. Jiang, F. J. H. Santiago, H. Mo, L. Liu, and J. Han, "Approximate arithmetic circuits: A survey, characterization, and recent applications," *Proceedings of the IEEE*, vol. 108, no. 12, pp. 2108–2135, 2020.

[14] S. Amanollahi, M. Kamal, A. Afzali-Kusha, and M. Pedram, "Circuit-level techniques for logic and memory blocks in approximate computing systems," *Proceedings of the IEEE*, vol. 108, no. 12, pp. 2150–2177, 2020.

[15] H. Jiang, F. J. H. Santiago, H. Mo, L. Liu, and J. Han, "Approximate arithmetic circuits: A survey, characterization, and recent applications," *Proceedings of the IEEE*, vol. 108, no. 12, pp. 2108–2135, 2020.

[16] S. Froehlich and R. Drechsler, "Unlocking approximation for in-memory computing with cartesian genetic programming and computer algebra for arithmetic circuits," *it-Information Technology*, vol. 64, no. 3, pp. 99–107, 2022.

[17] C. K. Jha, S. Ahmadi-Pour, and R. Drechsler, "MARADIV: library of magic based approximate restoring array divider benchmark circuits for in-memory computing using memristors," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2023.

[18] C. K. Jha, P. L. Thangkhiew, K. Datta, and R. Drechsler, "Imagin: Library of imply and magic nor-based approximate adders for in-memory computing," *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, vol. 8, no. 2, pp. 68–76, 2022.

[19] P. L. Thangkhiew, R. Gharpinde, and K. Datta, "Efficient mapping of boolean functions to memristor crossbar using magic nor gates," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 8, pp. 2466–2476, 2018.

[20] 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.

[21] ehwFIT, "py-paretoarchive," https://github.com/ehw-fit/py-paretoarchive, 2023.

[22] T. Glasmachers, "A fast incremental bsp tree archive for non-dominated points," in *Evolutionary Multi-Criterion Optimization*. Cham: Springer International Publishing, 2017, pp. 252–266.