Investigating Various Adder Architectures for Digital In-Memory Computing Using MAGIC-based Memristor Design Style

Chandan Kumar Jha[1*], Alireza Mahzoon[2], and Rolf Drechsler[1,2]
German Research Center for Artificial Intelligence, DFKI GmbH, Bremen, Germany[1]
Institute of Computer Science, University of Bremen, Bremen, Germany[2]
Email: chandan.jha@dfki.de, alireza.mahzoon@uni-bremen.de, drechsler@uni-bremen.de

***Abstract*—Adders are implemented using a wide variety of architectures. These architectures have been extensively studied for digital IC-based implementations. In recent years, in-memory computing has gained interest owing to the benefits it provides in terms of both energy and performance as compared to conventional von Neumann computing. In this work, we for the first time investigate various adder architectures for in-memory computing using the memristor aided logic (MAGIC) design style for memristors. We analyze seven different adder architectures for bit-widths: 8-bit, 16-bit, 32-bit, and 64-bit. We have used the state-of-the-art SIMPLER tool for performing the mapping of these adders to memristor crossbars. We show that serial prefix adders are better suitable for IMC using the MAGIC design style as compared to the widely used ripple carry adder. The adder designs and the mapping will be made open source at https://github.com/agra-uni-bremen/icee2022-magic-adder-lib, to promote further research in the direction.**

I. Introduction

In-memory computing (IMC) using memristors has gained immense popularity in recent years [1]. A memristor is a two-terminal device capable of changing its resistance depending upon the applied voltage/current [2], [3]. Memristors can be configured to be in a high resistance state (logic 0) or a low resistance state (logic 1), which can then be used to perform computations [4]. Implementing arithmetic circuits using memristors has been explored using both digital and analog computations [5], [6]. In this work, we focus on using memristors for the implementation of adders using digital computations [7]–[9].

We have used one of the most popular logic design styles using memristors called memristor aided logic (MAGIC) for implementing adders [10], [11]. The NOR and NOT operations implemented using the MAGIC design style are shown in Fig. 1. These gates can be implemented using a memristor crossbar array as shown in Fig. 2. Before performing the NOR operation, the output memristor (M_{out}) is first set to low resistance, i.e., logic 1 state. When both the input memristors (M_{in1}, M_{in2}) are in logic 0 state, the output memristor remains in logic 1 state, as the current through M_{out} is lesser than the threshold current. In all the other cases, at least one of the input memristors is ON and the state of the output memristor changes to logic 0 as sufficient current flows through the memristor M_{out}. The NOT gate also operates similarly but requires only one input memristor

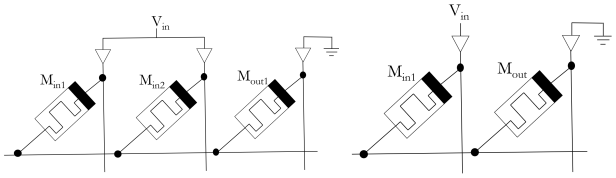


Fig. 1: MAGIC design style implementation (a) NOR gate (b) NOT gate

(M_{in1}) as shown in Fig. 1. If the input memristor has logic 0, the output memristor remains in logic 1 state. Whereas if the input memristor has a logic 1 state the output memristor M_{out} state is changed to logic 0.

Prior works on memristor based adders have mostly limited themselves to ripple carry adders [7], [9]. In this work for the first time, we compare and analyze various adder architectures. We generated seven different adder architectures namely Ripple Carry (*RC*), Carry-Lookahead (*CL*), Ladner-Fischer (*LF*), Kogge-Stone (*KS*), Brent-Kung (*BK*), Carry Skip (*CK*), and Serial Prefix (*SE*) [11], [12]. The general structure of the parallel prefix adder is shown in Fig. 3 [11]. It has three major parts. The first part is used to obtain the generate and propagate signals from the primary inputs using (1). The second stage uses these signals to generate the carry bits. There are multiple ways in which the carry generation logic can be implemented leading to different adder designs using (2,3). Lastly, the carry and the propagate bits are used to generate the final sum of the addition using (4).

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

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

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

$$s_i = p_i \oplus g_{i-1:0} \tag{4}$$

While these adder architectures have been extensively studied and tailored depending upon the constraints of the digital IC design, the same is not true for memristors. This motivates the need for in-depth analysis and comparison across the adder

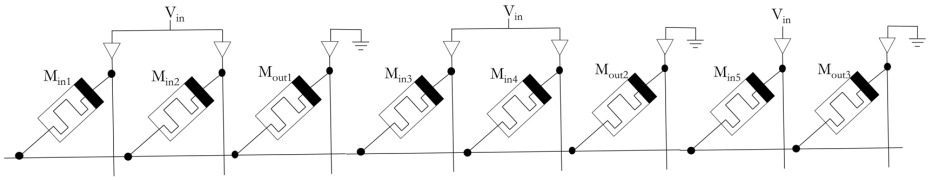


Fig. 2: Mapping 2 NOR and 1 NOT operation to a single row using MAGIC design style

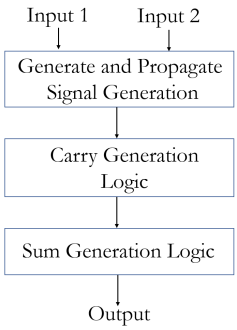


Fig. 3: General structure of a parallel prefix adder

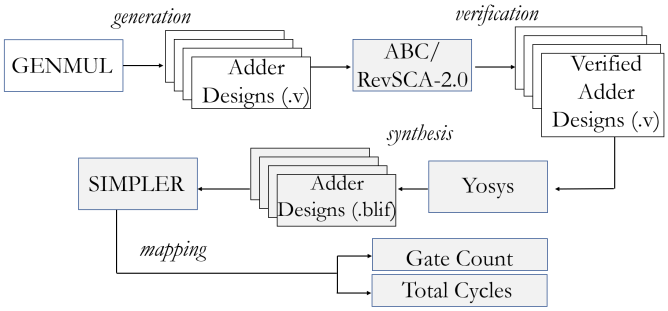


Fig. 4: Framework used for analysis of adder architectures

architectures to identify the most suited adder architecture for the MAGIC based design style. In this work, we have the following contributions:

- We present the first in-depth comparison across 7 different adder architectures for bit-widths ranging from 8-bit to 64-bit for IMC using MAGIC design style for memristor crossbars.
- Contrary to the existing works that use ripple carry adders for MAGIC-based design we show that serial prefix adders are best suitable for MAGIC-based design style. On average across all bit-widths serial prefix adder has 8.5% lesser gate count and 8.9% lesser cycles as compared to ripple carry adder.
- We will make the formally verified adder designs and the magic design style-based mapping open source as they can be used both as a benchmark and for building larger designs.

The rest of the paper is organized as follows. In Section II, we discuss the framework used for evaluation. In Section III, we discuss the results and analysis. We conclude the paper in Section IV.

II. Framework For Evaluation

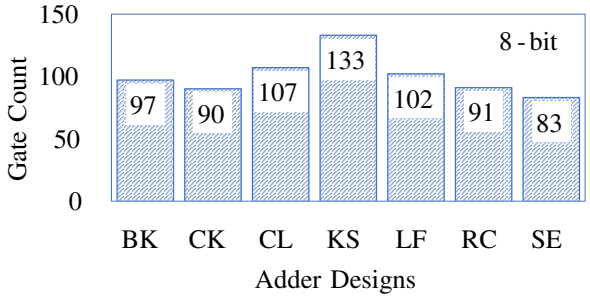
The overall framework for evaluation is shown in Fig. 4. We used the GENMUL framework to generate the design of adders [13]. We generated adders ranging from bit-width 8 to 64. For verification purposes, all the adder designs were converted to the .aig (AND-Inverter Graph) format using YOSYS synthesis tool. We formally verified the *RC* adder designs using the RevSCA-2.0 framework, which is a word-level verification method that uses symbolic computer algebra [14]. We then performed the combinational equivalence checking (CEC) using the ABC tool and the verified *RC* adders as the gold designs [15]. All the formally verified adder designs were then synthesized using the YOSYS synthesis tool. The YOSYS tool generates the .blif (Berkeley Library Exchange Format) of the adder designs. We then perform mapping of these adder designs to the memristor crossbars using the SIMPLER framework [7]. We started with 25 memristors and kept increasing the memristor count in steps of 25 until all the designs for a particular bit-width had a mapping using SIMPLER. SIMPLER provides the mapping of the adder designs to NOR and NOT operations on a memristor crossbar. We get the gate count and the number of cycles from the SIMPLER tool as the design metrics. We have performed a technology-independent mapping using the SIMPLER tool. The mapping can be used to implement the design on a crossbar and perform circuit-level analysis.

III. Results and Discussion

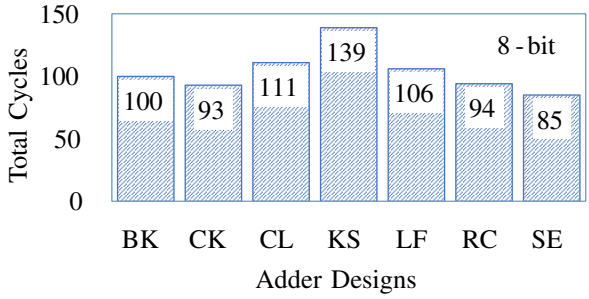
In this section, we discuss the results obtained by mapping adders to memristor crossbars. *RC* adders have been explored in earlier works related to mapping memristor to crossbars and have shown to be the most efficient [7], [9]. Hence, we have used *RC* adders mapped using the state-of-the-art SIMPLER tool as the baseline for comparison.

A. 8-bit Adders

The result for the 8-bit adder is shown in Fig. 5. We see that the gate count and total cycles vary from 83 (*SE*) to 133 (*KS*) and 85 (*SE*) to 139 (*KS*) respectively. *SE* and *CK* adders are better for MAGIC design style as compared to *RC* adders by 8.7% and 1.1% in terms of gate count and 9.6% and 1.1% in terms of total cycles respectively.

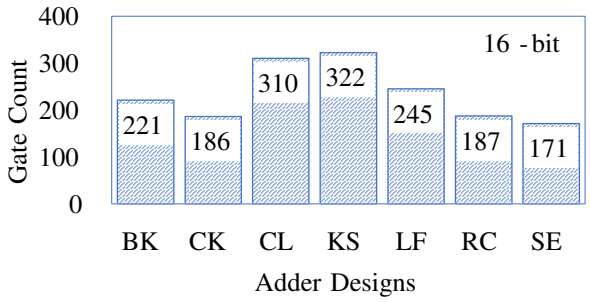


(a) Gate Count for various 8-bit adder designs

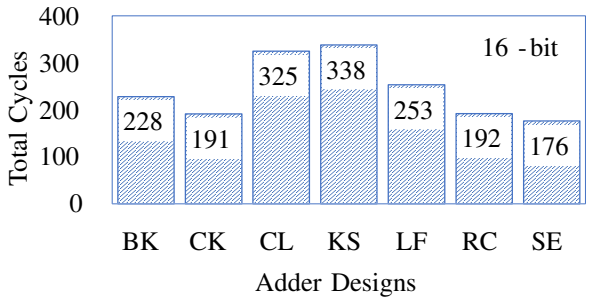


(b) Total Cycles for various 8-bit adder designs

Fig. 5: Gate Count and Total Cycles for 8-bit Adders using 50 memristors

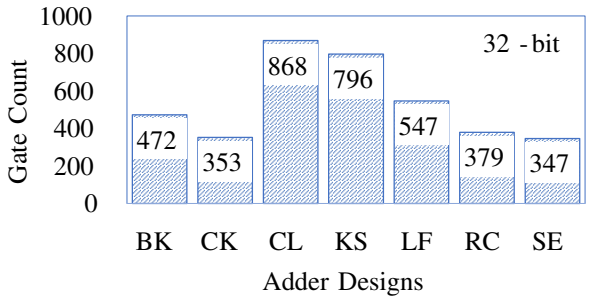


(a) Gate Count for various 16-bit adder designs

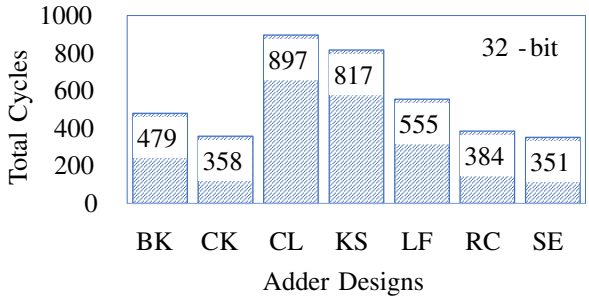


(b) Total Cycles for various 16-bit adder design

Fig. 6: Gate Count and Total Cycles for 16-bit Adders using 75 memristors



(a) Gate Count for various 32-bit adder designs



(b) Total Cycles for various 32-bit adder design

Fig. 7: Gate Count and Total Cycles for 32-bit adders using 150 memristors

B. 16-bit Adders

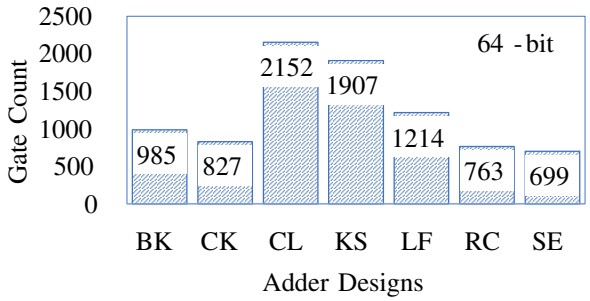
The result for the 16-bit adder is shown in Fig. 6. We see that the gate count and total cycles vary from 171 (*SE*) to 322 (*KS*) and 176 (*SE*) to 338 (*KS*) respectively. *SE* and *CK* adders are better for MAGIC design style as compared to *RC* adders by 8.6% and 0.5% in terms of gate count and 8.3% and 0.5% in terms of total cycles respectively.

C. 32-bit Adders

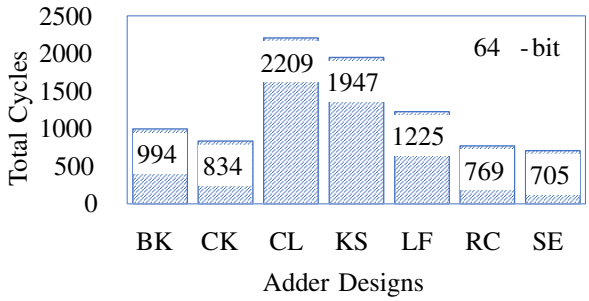
The result for the 32-bit adder is shown in Fig. 7. We see that the gate count and total cycles vary from 347 (*SE*) to 868 (*CL*) and 351 (*SE*) to 897 (*CL*) respectively. *SE* and *CK* adders are better for MAGIC design style as compared to *RC* adders by 8.4% and 6.8% in terms of gate count and 8.6% and 7.8% in terms of total cycles respectively.

D. 64-bit Adders

The result for the 64-bit adder is shown in Fig. 8. We see that the gate count and total cycles vary from 699 (*SE*) to 2152 (*CL*) and 705 (*SE*) to 2209 (*CL*) respectively. *SE* adders are better for MAGIC design style as compared to *RC* adders



(a) Gate Count for various 64-bit adder designs



(b) Total Cycles for various 64-bit adder design

Fig. 8: Gate Count and Total Cycles for 64-bit adders using 275 memristors

by 8.4% and 9.1% in terms of gate count and total cycles respectively.

E. Overall Discussion

We see that *SE* adders and *CK* adders outperform *RC* adders for bit-widths 8, 16, and 32. *SE* adder outperforms *RC* adder for bit-width 64. As we move from 8 bits to 16, 32, and 64 bits, the gate count and total cycles increase by 2.06, 4.18, and 8.42 times respectively as compared to the 8-bit *SE* adder. While existing works have used *RC* adders as the baseline for implementing adders, we showed through our analysis that it is not the most optimal for the MAGIC design style. *SE* adders are the most suitable for IMC based on MAGIC design style in memristor crossbars. Since memristors-based IMC maps the designs to limited functions which are then implemented using memristor crossbars we believe this sort of analysis is very crucial to identifying the best adder architecture suitable for a given design style. Hence, we will also make the design and the mapping open source as this work can act as a baseline for further research in this direction.

IV. Conclusion

In this work, we performed a detailed comparison across different adder architectures for performing IMC using memristors. We observed that across bit-widths, the serial prefix adder uses the least number of gates and requires the least number of cycles. On average, across all bit-widths, serial prefix adder has 8.5% lesser gate count and 8.9% lesser cycles as compared to ripple carry adder. All the adder designs and the mapping are made open source at https://github.com/agra-uni-bremen/icee2022-magic-adder-lib, so that they can be used as benchmarks and promote further research in this direction. In the future, we would like to extend the analysis to other memristor-based design styles suitable for IMC.

Acknowledgement

This work was supported by the German Research Foundation (DFG) within the project PLiM (DR 287/35-1).

References

[1] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory devices and applications for in-memory computing," *Nature nanotechnology*, vol. 15, no. 7, pp. 529–544, 2020.

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

[3] 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.

[4] Y. Li, Z. Wang, R. Midya, Q. Xia, and J. J. Yang, "Review of memristor devices in neuromorphic computing: materials sciences and device challenges," *Journal of Physics D: Applied Physics*, vol. 51, no. 50, p. 503002, 2018.

[5] 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.

[6] P. Mannocci, G. Pedretti, E. Giannone, E. Melacarne, Z. Sun, and D. Ielmini, "A universal, analog, in-memory computing primitive for linear algebra using memristors," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 12, pp. 4889–4899, 2021.

[7] 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.

[8] 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.

[9] P. Thangkhiew, R. Gharpinde, D. N. Yadav, K. Datta, and I. Sengupta, "Efficient implementation of adder circuits in memristive crossbar array," in *TENCON 2017-2017 IEEE Region 10 Conference*. IEEE, 2017, pp. 207–212.

[10] 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.

[11] N. H. Weste and D. Harris, *CMOS VLSI design: a circuits and systems perspective*. Pearson Education India, 2015.

[12] A. Mahzoon and R. Drechsler, "Polynomial formal verification of prefix adders," in *2021 IEEE 30th Asian Test Symposium (ATS)*. IEEE, 2021, pp. 85–90.

[13] R. Drechsler and D. Große, *Recent Findings in Boolean Techniques*. Springer, 2021.

[14] A. Mahzoon, D. Große, and R. Drechsler, "RevSCA-2.0: SCA-based formal verification of nontrivial multipliers using reverse engineering and local vanishing removal," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 5, pp. 1573–1586, 2021.

[15] A. Mishchenko, S. Chatterjee, R. Brayton, and N. Een, "Improvements to combinational equivalence checking," in *2006 IEEE/ACM International Conference on Computer Aided Design*. IEEE, 2006, pp. 836–843.