Abstract—A wide variety of multiplier architectures optimized for area, delay, and energy have been proposed in the literature. These multipliers have been extensively studied for CMOS technology. While in-memory computing (IMC) using memristors has garnered significant interest in recent years, multiplier designs have received far less attention. In this work, we aim to bridge this gap and for the first time analyze and compare diverse multiplier architectures for IMC based on memristors. We analyze 275 different signed and unsigned multiplier architectures for the 4-bit, 8-bit, 16-bit, 32-bit, and 64-bit multipliers. We have used the state-of-the-art mapping tool called SIMPLER to perform the mapping of these multiplier designs to the memristor crossbars. We have used the memristor count and the number of cycles as design metrics to identify the most suitable architectures for IMC using the MAGIC design style. We show that there is a difference of $1.2\times$ in the design metrics between the best and worst multiplier architectures across all bit widths. We also show that several Array and Dadda Tree based multipliers are best suited for 4x4 multipliers. Multipliers having Dadda Tree based partial product accumulator and Serial Prefix final stage adder are best suited for 8-bit or higher bit width multipliers. We will make all the multiplier designs and memristor crossbar mapping files generated from SIMPLER open-source at https://github.com/agra-uni-bremen/newcas2023-magic-multiplier-lib. We believe that our work will act as a benchmark for future works in this direction and the designers can use them to perform further optimizations, synthesis for other design styles, verification, etc.

I. INTRODUCTION

In-memory computing (IMC) has attracted the interest of both industry and academia in recent years. IMC aims to reduce the issue of memory bottleneck in the conventional von Neumann architectures [1]. It also improves the energy efficiency as it reduces the to and fro data movement between memory and processing/compute unit [2]. IMC using memristors has been extensively explored as memristors can act both as storage and compute units [3]. A memristor is a two-terminal device that can change its resistance depending upon the magnitude and direction of the applied voltage across its terminals [4]. It can be used to perform both analog and digital computations [5], [6]. In this work, we focus on using memristors to perform digital computations [7]. Memristors can be configured to be in a high resistance state (HRS), i.e., logic 0 or a low resistance state (LRS), i.e., logic 1. These states can then be used to perform digital computations. There are several design styles to perform logic computation using memristors [8]–[14]. In this work, we have used one of the most popular design styles called the MAGIC design style [13]. The NOR and NOT operations performed using this design style can be mapped to memristor crossbars and hence are suitable for IMC [13].

Multiplication is one of the most common operations that need to be performed in applications [15]. Handcrafted in-memory multipliers designs based on memristors have received interest in recent years and remain an active area of research [16]–[18]. However, the analysis is limited to a single architecture or for a subset of bit widths. Hence, there is a need for overall analysis and comparison between different architectures and bit widths. Multiplication can be performed using a wide variety of architectures [19]–[21]. Each of these architectures has been studied in detail for digital IC designs [15], however, the same is not true for multipliers based on memristors for IMC. In this work, for the first time, we investigate these architectures for IMC using memristors. Since these architectures are optimized for digital IC design, the same designs when mapped to memristors can have different design properties. This makes it necessary to perform analysis to identify the best multiplier architectures when they are mapped to memristor crossbars using a design style. Our work has the following contributions:

- In this work, we present the first in-depth analysis and comparison of various multipliers implemented using the MAGIC design style on memristor crossbars.
- We analyzed 55 different signed and unsigned multipliers designs for each of the 4-bit, 8-bit, 16-bit, 32-bit, and 64-bit multipliers, i.e., a total of 275 multiplier designs.
- We used the number of gates and the total cycles as metrics for the evaluation of the multiplier designs mapped to memristor crossbars.
- We show the difference between the best and worst case multiplier designs is $1.2\times$ across all bit widths.
- We also show that Dadda Tree based partial product accumulator and Serial Prefix adder are best suited for
The rest of the paper is organized as follows. In Section II, we discuss the necessary background. In Section III, we discuss the framework used for performing the analysis. In Section IV, we discuss the comparison results of various multiplier designs and in Section V, we conclude the paper.

II. BACKGROUND

In this section, we discuss the necessary background of the MAGIC gates [13], the mapping tool SIMPLER [22], and multiplier architectures.

A. MAGIC Gates

In this work, we have used the MAGIC design style to implement the logic function using memristors [13]. The NOR and the NOT function can be mapped to the crossbar array as shown in Fig. 1. The output memristor \((M_{out})\), is set to 1 before the evaluation of any operation. Since it is a two-input NOR gate the total number of input combinations is 4. For all combinations except 00, there exists a path between the \(V_{in}\) and ground. Hence the current flows through the output memristor, increasing its resistance. This changes the logic state of the output memrisor from 1 to 0. When the input is 00, the output memristor remains in the low resistance state maintaining its logic state 1. For the NOT operation the current flows when the input memristor is in a low resistance state, i.e., logic 1, changing the logic state of the output memristor to 0.

For the case when the input memristor is in a high resistance state the output memristor maintains logic 1.

B. Multiplier Design

Several methods have been developed over the years to design multipliers targeting area, power, and delay but the overall architecture remains the same [23]. The first stage consists of the partial product generator. This generates the partial products depending upon the inputs to the multiplier. The next stage is the partial product accumulator which adds all the partial products to generate an intermediate output. The intermediate output is fed to the final stage adder to finally obtain the multiplication output. There are several ways to implement the partial product accumulator and the final stage adder as shown in Table I. In this work, we have used the GENMUL tool to generate the multiplier designs [23].

III. EVALUATION FRAMEWORK

The overall evaluation framework is shown in Fig. 2. We will now discuss the stages of the framework in detail.

A. Multiplier Design Generation (GENMUL)

In the first stage, we used GENMUL to generate the multiplier designs as shown in Fig. 2 1. GENMUL is an open source multiplier generator that takes the following parameters as input: a) bit width, b) partial product generator type, c) partial product accumulator type, and d) final stage adder type [23]. The values of these parameters are shown in Table I. We have used GENMUL to generate multipliers ranging from bit width of 4 to 64 for both signed and unsigned multipliers as shown in Fig. 2 2. We also generated all possible combinations of partial product generators and partial product accumulators1. Thus we have 28 and 27 unsigned and signed multiplier designs respectively for each bit width. Hence, overall we analyze 275 different multiplier designs.

B. Intermediate Synthesis (Yosys)

We have synthesized the Verilog design generated from GENMUL using Yosys to generate the Berkeley Library Exchange Format (.blif) as shown in Fig. 2 3. We used the `synth` and `flatten` commands of Yosys to generate the .blif files of the multiplier designs as shown in Fig. 2 4 [24]. This was done to make the designs compatible with the next stage of mapping them to memristor crossbars.

C. Mapping to Crossbar (SIMPLER)

SIMPLER is the state-of-the-art tool used to map the design to a memristor crossbar as shown in Fig. 2 5. In the final stage, we passed the .blif multiplier designs to the SIMPLER tool. SIMPLER performs the mapping of these designs to a memristor crossbar array using MAGIC NOR and NOT gates. SIMPLER maps these designs to a single row and reuses the memristors when necessary. There are various knobs for optimization in SIMPLER. We found the minimum number of memristors in a row that can be used to implement multiplier designs. We started with 25 memristors and kept increasing in steps of 25 until all the designs for a particular bit width had a mapping using SIMPLER. The number of gates and the total cycles were then obtained as metrics for evaluating the design as shown in Fig. 2 6.

IV. RESULTS AND DISCUSSION

In this section, we discuss the results obtained using our framework. Table II to Table VI, shows the result from 4-bit to 64-bit multipliers. The design names are abbreviated in the tables. For example, Dadda Tree partial product accumulator with Serial Prefix Adder as DT_SE as also shown in Table I.

1GENMUL throws an exception for a signed array multiplier with carry skip adder. Hence we have one less signed multiplier.
The minimum number of memristors required for mapping 4x4 multipliers with 50 memristors is 75. For unsigned multipliers, the number of gates and the cycle count range from 648-780 and 671-815 respectively. For signed multipliers, the number of gates and the cycle count range from 651-820 and 673-857 respectively. The best designs in terms of cycles are the ones based on Dadda Tree based partial product accumulation and Serial Prefix adders. The best unsigned multiplier has 648 gates and the number of cycles required for multiplication is 671 as highlighted in Table III. The best signed multiplier has 651 gates and the number of cycles required for multiplication is 673 as highlighted in Table III.

### C. 16x16 Multipliers

The result for the 16x16 multipliers is shown in Table IV. The minimum number of memristors required for mapping is 175. For unsigned multipliers, the number of gates and the cycle count range from 2822-3489 and 2856-3541 respectively. For signed multipliers, the number of gates and the cycle count range from 2825-3506 and 2859-3556 respectively. The best designs in terms of cycles are the ones based on Dadda Tree based partial product accumulation and Serial Prefix adders. The best unsigned multiplier has 2822 gates and the number of cycles required for multiplication is 2859 as highlighted in Table IV. The best signed multiplier has 2825 gates and the number of cycles required for multiplication is 2859 as highlighted in Table IV.

### D. 32x32 Multipliers

The result for the 32x32 multipliers is shown in Table V. The minimum number of memristors required for mapping is 350. For unsigned multipliers, the number of gates and the cycle count range from 11716-14452 and 11792-14575. For signed multipliers, the number of gates and the cycle count range from 11729-14486 and 11804-14602 respectively. The best designs in terms of cycles are the ones based on Dadda Tree based partial product accumulation and Serial Prefix adders.
Dadda Tree based partial product accumulation and Serial Prefix adders. The best unsigned multiplier has 11716 gates and the number of cycles required for multiplication is 11792 as highlighted in Table V. The best signed multiplier has 11729 gates and the number of cycles required for multiplication is 11804 as highlighted in Table V.

E. 6x64 Multipliers

The result for the 64x64 multipliers is shown in Table VI. The minimum number of memristors required for mapping is 700. For unsigned multipliers, the minimum number of gates and the cycle count range from 47892-58223 and 48047-58490 respectively. For signed multipliers, the number of gates and the cycle count range from 47898-57975 and 48052-58232 respectively. The best designs in terms of cycles are the ones based on Dadda Tree based partial product accumulation and Serial Prefix adders. The best unsigned multiplier has 47892 gates and the number of cycles required for multiplication is 48047 as highlighted in Table VI. The best signed multiplier has 47898 gates and the number of cycles required for multiplication is 48052 as highlighted in Table VI.

F. Overall Analysis (Inter Bit Width)

Overall we see that Dadda Tree with Serial Prefix adder requires the least number of gates and cycles for all bit widths. For 4x4 bit width, both Array multiplier and Dadda Tree multipliers give the best mapping. We see that as we move from bit width 4 to 8, 16, 32, and 64 the minimum number of memristors required for mapping increases by 1.5×, 3.5×, 7×, and 14× as compared to 4-bit respectively. For the best case gate count and the number of cycles as we move from bit width 4 to 8, 16, 32, and 64 increase by around 5×, 22×, 91×, and 374× respectively as compared to the 4-bit multiplier.

G. Overall Analysis (Intra Bit Width)

We now look at the overall variation in the metrics for a given bit width. We see that the difference between the worst case and the best case multiplier designs is around 1.2× for both gate count and the number of cycles for all the bit widths. Hence, by performing the design space exploration and comparisons across different multiplier architectures we can gain benefits of 1.2×, as compared to arbitrarily choosing any multiplier architecture. The best multiplier architecture can then be further optimized to gain more benefits. Since there are several design styles using memristors, we believe that this sort of analysis is necessary and useful as it helps us to identify the best multiplier architecture for a given memristor based design style. Since the designs and the mapping output of the SIMPLER tool will be open-source, these designs can not only be used as benchmarks but can also be further optimized, mapped to other design styles, and used as inputs for verification methodologies tailored for memristors.

V. CONCLUSION

In this work, we present an in-depth analysis of 275 different signed and unsigned multiplier architectures for bit width ranging from 4 to 64 for IMC. We used the state-of-the-art mapping tool and performed a thorough comparison to obtain the number of gates and the total cycles across different architectures. We observed that across different multiplier architectures, there is a 1.2× difference in these design metrics making this sort of study very important. We also identified that the design that uses Dadda Tree based partial product accumulator and Serial Prefix adder is the best suited for MAGIC-based IMC. Array based partial product accumulation also gives the best designs for a bit width of 4. We will make the multiplier designs and mapping obtained using SIMPLER open-source to facilitate further research in this direction.

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


