Mixed-Signal Memristor-based Iterative Montgomery Modular Multiplication

Mehdi Kamal and Massoud Pedram
Department of Electrical and Computer Engineering, University of Southern California
{mehdi.kamal, pedram}@usc.edu

***Abstract*—In this paper, we present a mixed-signal implementation of iterative Montgomery multiplication algorithm (called X-IMM) for using in large arithmetic word size (LAWS) computations. LAWS is mainly utilized in security applications such as lattice-based cryptography where the width of the input operands may be equal to or larger than 1,024 bits. The proposed architecture is based on the iterative implementation of Montgomery multiplication (MM) algorithm where some critical parts of the multiplication are computed in the analog domain by mapping them on the memristor crossbar. Using a memristor crossbar reduces the area usage and latency of the modular multiplication unit compared to its fully digital implementation. The devised mixed-signal MM implementation is scalable by cascading the smaller X-IMMs to support dynamically adjustable larger operand sizes at runtime. The effectiveness of the proposed MM structure is assessed in the 45nm technology and the comparative studies show that the proposed 1,024-bit Radix-4 (Radix-16) Montgomery multiplication architecture provides about 13% (22%) higher $GOPS/mm^2$ compared to the state-of-the-art digital implementations of the iterative MM.**

***Index Terms*—Mixed-Signal Computing, Montgomery Multiplier, Memristor, Security, Latency**

I. Introduction

We are living in a data explosion era where there are billions of the devices that are connected to the internet. These devices generate huge amount of data every second that should be analyzed and interpreted by the computing systems. Thus, different data-driven computing technologies have been extended to reach higher capability for a variety of applications. However, along with high computational capability, data privacy is the another key consideration that has been raised in many application scenarios [1].

To preserve the users' data and models, several cryptosystems have been introduced such as elliptic-curve cryptography (ECC) [2] and fully homomorphic encryption (FHE) [1]. FHE schemes are specially attractive because they enable computations to be performed on ciphertexts directly, thereby, keeping the user data in encrypted form even when it is undergoing an operation by an untrusted third-party. High time complexity is a common serious issue in these systems and thus custom integrated circuit implementation is a promising solution for improving the throughput of these systems. In most of these systems, the cryptograhic algorithms are based on the computations on very long integer numbers, and thus, long word modular multiplication is the most important basic arithmetic operation for these systems [3], [4]. Hence, the speed and area complexity of the hardware structure used for modular multiplication is critical to efficient realization of custom hardware targeting cryprosystems. Montgomery modular multiplication is a well-known algorithm for doing modular multiplication [5]. Since cryprographic algorithms deal with long word computations, iterative Montgomery multiplication provides a balance between the latency and area usage. In this case, the area usage and the required cycle count for completing the modular multiplication operation depends on the chosen radix for the computation [6].

Generally, optimizations at different design abstraction levels (from devices to algorithms) are needed to improve the efficiency of the hardware designs. Moreover, computing paradigm shift to in-memory computing is the other promising solution [7]. Recently, using emerging non-volatile memories (NVMs) such as PCM, ReRAM, MRAM to compose analog computational units has received a lot of attention by the academia and industry. One of these emerging devices is the memristor device [8]. Multiplication is an operation that memristors can perform very efficiently, and due to this fact alone, memristors (in the form of memristor crossbar structures) are widely utilized in machine learning applications as an analog matrix-vector multiplication (MVM) engine. Using memristors leads to lower energy consumption and area usage as well as higher computational speed [9]. The memristor crossbar is fabricated as part of the back end of line (BEOL) processing, and thus, they are typically placed on the top of the CMOS devices, and therefore, their additional area footprint can be quite small. Note however that although a memristor does not occupy any space among the transistors, its access transistor (*e.g.,* a MOSFET, in 1T1R scheme) requires a small area footprint. The memristor device may also be used as multi-level cell and in this case more than one bit of data could be stored in it resulting in higher density in mixed-signal designs. However, the memristor device suffers from the non-ideality of its parameters that can limit its ability to provide higher speed and energy efficiency [10].

In this paper, we present a memristor-based iterative Montgomery multiplication architecture to be used in modern (post-quantum) security cryptosystems such as FHE. This architecture relies on a mixed-signal approach to compute the intermediate and final results of the modular multiplication. The memristor crossbar is utilized to perform the dot-product operation in analog domain and generates the intermediate results in redundant representation. A column circuitry is used to convert the analog intermediate results to digital values so that the remaining steps of the Montgomery multiplication

978-3-9819263-7-8/DATE23/© 2023 EDAA

algorithm can be done in the digital domain. The X-IMMs could be chained to support wider operands in the runtime. The scalability feature gives flexibility to support different security levels by the fabricated chip. As the knowledge of the authors goes, this work is the first attempt for developing a memristor-based mixed-signal structure for *accelerating the computations* in the cryptosystems.

The remainder of the paper is organized as follows. In Section II, a brief background of the Montgomery multiplication will be provided. Also, relevant prior work is reviewed. The details of the X-IMM structure is explained in Section III. In Section IV, the effectiveness of the proposed MM structure is assessed. Finally, the paper is concluded in Section V.

II. Background

A. Montgomery Modular Multiplication

The Montgomery multiplication efficiently obtains the result (*i.e.,* $\hat{Z}$ in the range of $[0, M)$) of the modular multiplication (with modulus M) of two long operands (*i.e.,* $\hat{Z} = \hat{X}\hat{Y}$ mod M, where $\hat{X}$ and $\hat{Y}$ are in the range of $[0, M)$). In this algorithm, first the input operands should be transformed to the Montgomery domain (*e.g.,* $X = \hat{X}R$ mod M, where R is a power of two number and $R < M$) and then the multiplication operation is performed and the result (Z) in the Montgomery domain is calculated by applying the Montgomery reduction. The result in the Montgomery domain is converted to the ordinary integer domain by applying $\hat{Z} = ZR^{-1}$ mod M, where R^{-1} is the modular inverse of the R i.e., $|RR^{-1}|_M = 1$.

Note that in typical applications (e.g., when doing modular exponentiation required in many cryptosystems), the digital data is transferred to the Montgomery domain only one time and many modular operations are done on this data before final output result is transformed back to the ordinary integer domain. Thus, in the Montgomery multiplication algorithm, the cost of data conversion between domains is not high. Thus, the steps for computing output Z are as follows: 1) $W = X \times Y$; 2) $q = |W \times M^{-1}|_R$; 3) $Z = (W + qM)/R$; and 4) $Z = (Z > M)\ ?\ Z - M : Z$.

B. Prior Work

There are several prior work references in the area of designing Montgomery modular multipliers. In this subsection, the latest ones are reviewed briefly. Note that all of these works use digital circuits and have been implemented on the FPGA or ASIC by using CMOS devices.

In [5], a fast Montgomery multiplier based on the redundant number representation (RBR) has been proposed. In this structure, small adders have been used instead of the multiplier. Also, by using RBR, the add operations was performed carry-free which led to high parallel computation. Using RBR also has been used in [6] for parallel computation. Moreover, a modified booth coding technique has been introduced that by combining with the RBR more speed has been gained. A Montgomery multiplier based on the Residue Number System (RNS) which could support up 8K-bit input operands has been suggested in [3]. A low memory usage scalable Montgomery multiplier structure has been described in [11]. To reach lower memory consumption, the intermediate results representation was converted from redundant number to the binary one and FIFOs have been used to manage the data access flow. A high-radix scalable Montgomery multiplier structure has been introduced in [4]. In this structure by using carry save compressors and decomposing the multiplicand, the delay of the modular multiplication has been reduced. In [12] a ReRAM-based multiplier unit has been suggested for fully homomorphic computation. Nevertheless, the suggested structure computed the multiplication in digital domain with add-and-shift technique (requires $O(N^2)$ cycles). Also, the modulo operation was performed separately after the multiplication (Barrett and Montgomery reductions) while the implemented modulo operation only supported a set of three small moduli.

III. Proposed Mixed-Signal Montgomery Multiplication Structure

The pseudocode of iterative Montgomery multiplication is provided in Algorithm 1 where its details could be found in [4]. The term r determines the radix of the computation, and Y_i is the i^{th} part of the input operand Y (*i.e.,* $Y_i = Y[(i+1)m - 1 : im]$). Notice that since r is in the form of 2^m, multiplication and division by it require a simple shift operation that can be implemented by hard wiring. Computing $q^{(i)}$ is not complex. $|Z^{(i-1)}|_r$ is the m lower bits of $Z^{(i-1)}$ and $|Z^{(i-1)} \times M'|_r$ is implemented by a $m \times m$ multiplier where only its m lower bits are required. Thus, the hardware complexity of obtaining $q^{(i-1)}$ is low. On the other hand, computing $Z^{(i-1)}$ requires long-width multiplication and addition, and it is the main part of the IMM hardware implementation. This structure performs modular multiplication in d cycles. Note that as shown in [4], the last stage for checking the Z value (step 4 for computing output Z as described in subsection II-A) is not required.

ALGORITHM 1: Pseudo-code of the iterative MM

input : $X, Y \in [0, 2M)$, odd $M < 2^N$, $r = 2^m$, $d = \lceil \frac{N+m+2}{m} \rceil$, $R = r^{d-1}$, $rr^{-1} - MM' = 1$
output: $Z \in [0, 2M)$

1 $Z^{(-1)} \leftarrow 0$
2 **for** $i \leftarrow 0$ **to** $d - 1$ **do**
3 $\quad q^{(i)} = (Z^{(i-1)} \bmod r)M' \bmod r$
4 $\quad Z^{(i)} = (Z^{(i-1)} + q^{(i)}M + XY_i r)/r$
5 **end**
6 $Z \leftarrow Z^{(d-1)}$
7 **return** (Z)

A. Analog Domain

A column of a memristor crossbar can perform the dot-product operation where each memristor models the multiplication operation. Figure 1 shows the general structure of a 3×3 memristor crossbar where the input voltage of each row is indicated by V_i and the conductance of the memristors is denoted by G. At the end of each column, a Trans-Impedance Amplifier (TIA) is placed to convert the column current to the voltage. Thus, input operands of the dot-product operation are voltages of the rows (*i.e.,* V_i) and the conductance of the

memristors (*i.e.,* $G_{i,j}$). We calculate $Z^{(i-1)}$ by employing the memristor crossbar. The general structure of the proposed X-IMM is illustrated in Figure 2.

Each memristor crossbar column performs a dot product operation, and in theory a column of the memristor crossbar is enough for computing $Z^{(i)}$. However, since in current technologies, the data width which can be stored on each memristor cell is limited to a few bits (e.g., < 8 bits) and the required voltage resolution is very low (even lower than the noise level), we partition the computation of $Z^{(i)}$ on a set of memristor crossbar columns (*i.e.,* bit-slicing method). In this structure, the memristors are used to store m bit data. In each column (except for the first column), 8 memristors are placed where two of these memristors are used to store the M and X input operands. However, the stored values in the two other ones is 1, and the rest of them store 0. In the j^{th} column, values of $X_{(j+1)}$ and M_j ($X_j = X[(j+1)m-1:jm]$ and $M_j = M[(j+1)m-1:jm]$) are programmed onto two memristors in a column. Compared to M, the input X is shifted to the left by m. This is because the structure should compute YXr ($r = 2^m$). Thus, value X is stored from the second column, while its corresponding memristor in the first column is dummy and stores a value 0 (as will be explained later, this memristor will be used in the case of cascading X-IMMs). Therefore, the memristor crossbar contains $\lceil log(2M)/r \rceil$ columns.

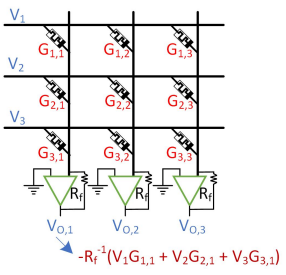

Fig. 1. General structure of a 3×3 memristor crossbar.

To store a value on the memristor, first its equivalent conductance value should be determined. In this work, to reduce the impact of the telegraph noise, which is the most dominant noise in the high resistance state (HRS) [13], we propose the following mapping function to obtain the corresponding conductance (*i.e.,* G) of an input value (*i.e.,* K):

$$\begin{aligned} G &= G_{max} - K \times (G_{max} - G_{min})/(2^m - 1) \\ &= G_{max} - K \times \Delta_G. \end{aligned} \quad (1)$$

where G_{min} and G_{max} denote the minimum and maximum conductance of the employed memristors. Input voltages of the first and second columns are the equivalent voltages of Y_i and $q^{(i)}$, respectively. Similar to (1), the equivalent voltage of a value K is

$$V = K \times (V_{DD} - V_{SS})/(2^m - 1) = K \times \Delta_V. \quad (2)$$

In this structure, to reduce the memristance drift, we set V_{SS} to $-V_{DD}$ (instead of 0) leading to lower read voltages in order to have a large margin between the read voltage and the threshold voltage of the employed memristor.

As will be explained later, in the proposed structure, $Z^{(i)}$ is represented in the redundant number system ($Z1^{(i)}$ and $Z2^{(i)}$). Thus, in each column, we consider two memristors (with conductance of G_{max} - Δ_G which is denoted by G^1) for $Z^{(i-1)}$. The input voltages of these memristors are the corresponding voltages of the $Z1^{(i)}$ and $Z2^{(i)}$. If each column only consists of these four memristors, the output voltage of the j^{th} column TIA (*i.e.,* $V_{out,j}$) is

$$-\frac{V_{out,j}}{R_f} = V_{Y_i} G_{X_{j-1}} + V_{q^{(i)}} G_{M_j} + V_{Z1_{j+1}^{(i-1)}} G^1 + V_{Z2_{j+1}^{(i-1)}} G^1. \quad (3)$$

In computing $Z^{(i)}$, in the last stage, the result should be divided by r (*i.e.,* shifting to right by m bits). Thus, in (3), for computing $Z_j^{(i)}$ in j^{th} column, instead of $Z1_j^{(i-1)}$ and $Z2_j^{(i-1)}$, $Z1_{j+1}^{(i-1)}$ and $Z2_{j+1}^{(i-1)}$ are used. Now, by substituting the voltages and conductances in (3) by their corresponding values (based on (1) and (2)), the corresponding TIA output value of the j^{th} column (*i.e.,* $O_{out,j}$) is

$$\begin{aligned} &-\frac{O_{out,j}\Delta_V}{R_f} = \\ &Y_i\Delta_V(G_{max} - X_{j-1}\Delta_G) + q^{(i)}\Delta_V(G_{max} - M_j\Delta_G) \\ &+ Z1_{j+1}^{(i-1)}\Delta_V(G_{max} - \Delta_G) + Z2_{j+1}^{(i-1)}\Delta_V(G_{max} - \Delta_G) \\ &= \Delta_V G_{max}(Y_i + q^{(i)} + Z1_{j+1}^{(i-1)} + Z2_{j+1}^{(i-1)}) - \\ &\Delta_V\Delta_G(Y_iX_{j-1} + q^{(i)}M_j + Z1_{j+1}^{(i-1)} + Z2_{j+1}^{(i-1)}). \end{aligned} \quad (4)$$

In (4), there is an extra term of $\Delta_V G_{max}(.)$ that should be removed. For removing this, we consider four additional memristors with the conductance of G_{max} (equal to storing value 0), with the negative corresponding input voltages of the Y_i, $q^{(i)}$, $Z1^{(i-1)}$ and $Z2^{(i-1)}$. In this case, the output value of the j^{th} column TIA is

$$O_{out,j} = \Delta_G R_f(Y_iX_j + q^{(i)}M_j + Z1_{j+1}^{(i-1)} + Z2_{j+1}^{(i-1)}). \quad (5)$$

These four memristors (with corresponding values of 0) as well as the two ones used for $Z1^{(i-1)}$ and $Z2^{(i-1)}$ (with corresponding values of 1) are programmed at one time and since their resistances are low (i.e., they are close to the LRS value), impact of the telegraph noise on currents passing through them is negligible. Based on (5), the resistance of R_f will be equal to 1 / Δ_G.

As mentioned above, the bit width of the computation in each column is $2m + 2$. This is due to the fact that the bit widths of Y_iX_{j-1}, $q^{(i)}M_j$, $Z1_{j+1}^{(i-1)}$ and $Z2_{j+1}^{(i-1)}$ are $2m$, $2m$, m and m bits, respectively. Hence, we use a $(2m + 2)$-bit analog-to-digital converter (ADC) for converting the analog output of the memristor crossbar to the digital value. For this converter, we use a low triangle neural network (LTNN) ADC, which was presented in [9]. The structure of this ADC is based on the Hopfield network, where its weights are implemented using memristors. Details of designing this type of ADC are provided in [9], [14], [15]. Note that in this work, similar to [9] we use an inverter as the network neuron leading to considerably lower delay and power consumption.

B. Digital Domain

The output of each memristor column ADC is $(2m + 2)$ bits which compose the $Z^{(i-1)}$. However, the outputs of each three adjacent columns have overlap with each other. In this

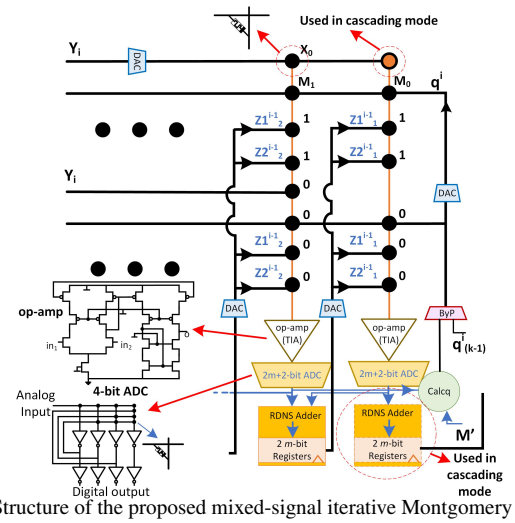

Fig. 2. Structure of the proposed mixed-signal iterative Montgomery multiplier.

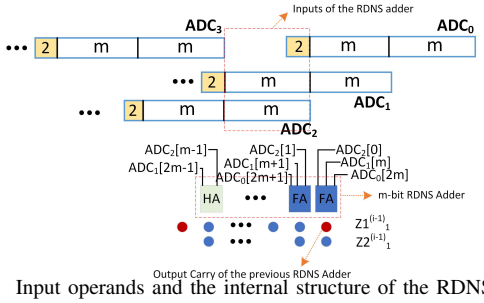

Fig. 3. Input operands and the internal structure of the RDNS adder.

situation, the output $Z^{(i-1)}$ may be considered as a redundant number such that in each bit position it has three bits. Since only in some positions, there are three bits, we use an m-bit redundant number system (RDNS) adder for each column to reduce the number of bits to two in each bit position. This adder is comprised of two Full adders (in the LSB) and $m-2$ Half adders (in the MSB). The output of the adder (denoted as $Z1$ and $Z2$) is stored in two m-bit registers. The overlap between the outputs of the ADCs and the structure of the RDNS adder are illustrated in Figure 3.

In the digital domain, $q^{(i-1)}$ (line 3 of Algorithm 1) is computed by *Calcq* unit. This unit is an add-and-multiply unit, which first adds two m-bit numbers and then its summation result is multiplied by the M'. The input operands of its adder are the outputs of the ADCs of the the first and second columns. Thus, the adder of this unit acts as a RDNS-to-binary converter. Next, the adder output is multiplied by M'. Note that only the m lower bits of the multiplier output are considered as $q^{(i-1)}$.

After d cycles (line 2 of Algorithm 1), the final result is stored in the registers of the RDNS adder in the format of RDNS. Thus, at the final step, an N-bit binary adder (named *Conv Adder*) is employed (although it is not shown in Figure 2) to convert the result in the RDNS format to the binary number system (BNS). The adder structure (Kogge-Stone adder, Carry Save Adder, etc.) should be explored and an structure which does not change the latency of the X-IMM (but also has a low area usage) should be chosen. The said binary adder may convert the number system in a single cycle or multiple cycles. By enlarging the bit width of the input operands or decreasing the radix of the computation, the probability of using multi-cycle adder is increased.

In the X-IMM architecture, the m lower output bits of the first ADC are not used. However, the proposed structure is scalable and by cascading C identical X-IMM units, one can compute larger input operands. In this case, $q^{(i)}$ is computed by the first X-IMM unit, while the other X-IMM modules bypass their computed q and use the result of the first X-IMM unit. The bypassing is performed by the ByP multiplexer as shown in Figure 2. Moreover, in case of cascading X-IMM units, the m lower output bits of the first ADC of the c^{th} X-IMM unit is passed to the $(c-1)^{st}$ X-IMM unit for computing the $Z1^{(i-1)}$ and $Z2^{(i-1)}$ in the last memristor column of the $(c-1)^{st}$ X-IMM module. Moreover, the dummy memristor of the first column of the X-IMM units (except the first X-IMM module) is used for storing input operand X. Note that in this case, the number of iterations for computing the final result (*i.e.,* d) will be $\lceil \frac{CN+m+2}{m} \rceil$ (where C denotes the number of the cascaded X-IMMs). Finally, to convert the final results from RDNS to BNS format, the *Conv Adders* are chained and a block-based adder is composed.

C. Reliability

While using the memristors can lead to higher speed and energy efficiency, the design suffers from various reliability issues, including the telegraph noise, memristance drift, write variability, and low endurance [16]. As mentioned before, the proposed integer to conductance mapping (*i.e.,* (1)) leads to lowering of the telegraph noise on most of the memristors on each column. However, to reduce the impact of the telegraph noise on the other memristors, one may consider a higher minimum conductance value compared to the actual G_{min}. Note that this approach leads to lower R_{off}/R_{on} which generally results in to more sensitivity to the other nonideality sources of the design parameters, especially in the case of multi-level cell. Thus, this approach is effective in the case of memristors with large R_{off}/R_{on}.

The memristance drift depends on the read voltage. On the other hand, in the FHE applications, the input operands tend to have uniform distribution. Thus, on average, all memristors in a column experience almost similar current density. Nevertheless, their stored values are different. Thus, the amount of drift can be different. Therefore, as mentioned before, in the proposed architecture, we use $V_{SS} = -V_{DD}$ to reduce the absolute value of the read voltage, and reduce the resistance drift of the memristors. The other nonidealities are the write variability and asymmetric write, which may be alleviated by choosing the proper writing method. Different precise writing techniques have been proposed in the literature (*e.g.,* [17], [18]). Note that in the X-IMM architecture, only a small subset of the memristors (the ones for storing the X and M) should be programmed for each new input operand pair. In this case, considering that the accuracy of the ADC is ± 0.5 LSB and that there are no voltage variations, the maximum tolerable variation of the conductance (δG) of these memristors in j^{th} column can be formulated as:

$$\frac{1}{2}\Delta V > \frac{1}{\Delta G}(V_{Y_i}\delta G_{X_{j-1}} + V_{q^{(i)}}\delta G_{M_j}) \qquad (6)$$

By assuming the maximum input values for the Y_i and $q^{(i)}$, (6) could be written as:

$$\frac{1}{2}\Delta V > \frac{1}{\Delta G} V_{DD}(\delta G_{X_{j-1}} + \delta G_{M_j}) \Rightarrow \\ \frac{\Delta G}{2r} > (\delta G_{X_{j-1}} + \delta G_{M_j}) \quad (7)$$

Thus, for precise computing, the maximum tolerable conductance variation of these two memristors is $\Delta G/4r$. Based on this, memristors with larger R_{off}/R_{on} are preferable. Moreover, the conductance variation around HRS is higher [19]. Hence, when R_{off}/R_{on} is large, one may consider a lower maximum resistance, which is lower than the actual HRS. In addition, as a general circuit level technique, one may improve the reliability of the memristor crossbar computation by considering identical redundant memristor crossbars (which may be placed on different metal layers) (see [20]).

IV. Results and Discussion

The efficacy of the proposed Montgomery multiplier was assessed by implementing its digital parts in Verilog HDL and its analog parts in SPICE. The Synopsys design Compiler and Synopsys HSPICE were employed for extracting the design parameters of the digital and analog parts. Moreover, a high-level model of the multiplier was developed in the Python language to verify the functionality of the proposed solution. The 45nm NanGate technology [21] was used for digital part implementation, while the 45nm PDK has been used for implementing the analog parts. Moreover, the memristor model of [22] with conductance range of [0.12 $\mu\mho$, 8 $\mu\mho$] and an area of $50nm \times 50nm$ was utilized for the X-IMM implementation. The operating voltage level of the analog (digital) part was between $-0.5V$ to $+0.5V$ ($0V$ to $+1V$).

Table I reports the design parameters of the proposed 1,024- and 2,048-bit mixed-signal solutions and compares them with those of [5], [6], [11]. In case of the X-IMM, we have not considered the memristor cells in the area usage. Since, as mentioned before, they are implemented in the wiring layers. Also, we have considered four NAND2 ($2n$ INV) cells as the cell counts of the TIA (n-bit ADC) circuit. Note that in the case of the ADC structure, the INV cells with different sizes were employed. As the reported figures in the table show, the period (*i.e.,* delay) of the X-IMM is increased a factor of nearly $1.7\times$ by increasing m (*i.e.,* radix-2^m) from 2 to 4 . Nevertheless, the area usage is decreased by about 11%. This originates from the fact that in the case of $m = 4$, the memrsitor crossbar columns are two times lower than that of the X-IMM with $m = 2$ (a.k.a., the memristor crossbar computes wider intermediate results in case of $m = 4$). Thus, although the ADCs occupy more area in case of $m = 4$, the number of ADCs (as well as TIAs) is halved. Thus, the area usage was decreased by considering higher radix. Also, by increasing the radix, the latency is improved. As the results show by increasing the value of m from 2 to 4, the latency, on average, improved by about 13%.

The area and delay breakdowns of the proposed 1,024-bit X-IMM are illustrated in Figure 4. Notice that, while the area usage of memristors due to their placement, was not considered

TABLE I
Comparing the design parameters of the proposed MM with those of the some prior works.

Architecture	N	m	Time			Complexity	
			Cycle	Period (ps)	Latency (ns)	Cell Count	Area (um2)
X-IMM (45nm)	1024	2	515	610	314.15	27.1	62940
		4	259	1063	275.32	24.6	55690
	2048	2	1028	618	635.30	52.88	124852
		4	515	1070	551.05	47.7	111099
[6] (45nm)	1024	2	520	620	322.4	21.2	69252
		4	264	780	205.92	33.6	90949
	2048	2	1032	625	645	41.8	138459
		4	520	790	410.8	61.1	180505
[6] (14nm)	1024	2	520	280	145.6	35.5	13754
		4	264	348	91.87	49.2	18569
	2048	2	1032	293	302.38	68.5	27147
		4	520	357	185.64	95.4	36487
[5] (65nm)	1024	2	514	1060	545	85.1	N.A.
		4	258	1460	377	121.8	
	2048	2	1026	1070	1098	162	
		4	514	1529	786	224.8	
[11] (90nm)	1024	2	595	1100	655	97	N.A.

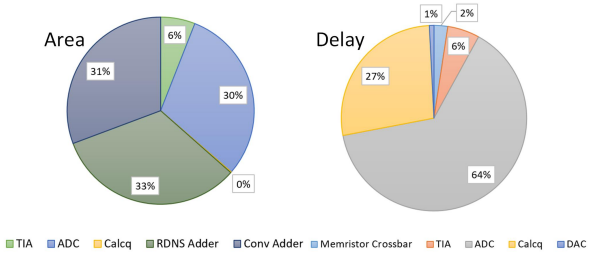

Fig. 4. Area and delay breakdowns of the proposed 1024-bit X-IMM.

in the area usage of the X-IMM, its area usage in the wiring layers was considerably lower than that of the transistors (by about $3\times$). Most of the X-IMM area is occupied by the RDNS adder units (containing a RDNS adder alongside of the registers for storing the intermediate results), ADCs and Final Adder units (totally about 90%). By increasing the width of the X-IMM (*e.g.,* 2048 bits), the area usage of the final adder module is increased more than the others. On the other hand, the delay of the X-IMM depends strongly on the delay of the ADCs. In addition, the delay of the unit for computing $q^{(i)}$ is notable, while the delay of the others is low. The delay of the *RDNS Adder* unit is lower than the *Calcq* unit, and hence, it is not in the critical path.

Owing to the different technology nodes being used, comparing the design parameters of the considered prior work with those of the X-IMM is not simple. However, as reported in [6], design parameters of [6] are considerably better than those of the two considered other works in similar FPGA platforms. Also, as shown in [6], the delay and area usages of the scalable MM proposed in [4] are higher than those of the solution proposed in [6] on an FPGA platform. On the other hand, compared to the 45nm technology implementation of [6], the latency of the proposed Radix-4 X-IMM is about 2.5% lower than that of Radix-4 [6]. However, the latency of Radix-16 [6] is about 33% lower than that of Radix-16 X-IMM. Thus, the latency of [6] is lower than that of the X-IMM, mainly due to large ADC delay of in X-IMM architecture. Nevertheless, by employing memristor cells, the area usage of the proposed X-IMM is lower than that of [6] (on average, 24%). Moreover, in X-IMM, by increasing the radix, more area saving could be gained. Figure 5 gives the throughput per area usage of the proposed X-IMM and [6] over different operand widths. As the results show, X-IMM leads to higher throughput per area

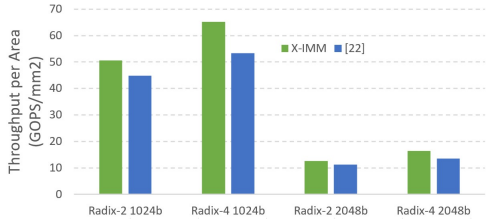

Fig. 5. Throughput per area of the proposed X-IMM and [6] over different operand widths.

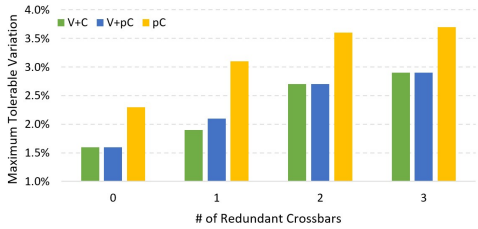

Fig. 6. Maximum amount of variation that the X-IMM could tolerate under different amount of memristor crossbar redundancy.

and by increasing the computation radix, the superiority of the X-IMM is increased.

We performed Monte Carlo simulations to determine the sensitivity of the multiplication output on the memristor crossbar variation (*i.e.,* (4)). For this study we used the developed python X-IMM model and assumed 1% variation on the input voltage of the memristor cells or their conductance. As the results show the sensitivity on the voltage and conductance variations of the memristors computing $Y_i X_{j-1}$ and $q^{(i)} M_j$ is much more than the others. While the lowest sensitivity is on the memristors which computes $\Delta_V G_{max}(Y_i + q^{(i)} + Z1_{j+1}^{(i-1)} + Z2_{j+1}^{(i-1)})$ (about $10^{-4}\times$ lower).

Finally, we evaluated the reliability of the proposed solution by considering the redundant crossbar under three scenarios by applying the variations on the voltage and conductance of the memristors. The scenarios consisted of 1) considering voltage and conductance variations on all memristors of the crossbar (denoted by V+C scenario), 2) considering voltage variation on all memristors while considering conductance variations on memristors of X and M (denoted by V+pC scenario), and 3) considering only the conductance variation on the memristors of X and M (denoted by pC scenario). Note that considering the conductance variation only on the memristors of X and M was due to the fact that the others had constant values. Also, in this study, we have applied the same amount of variations on voltage and conductance. The maximum amount of variation that the X-IMM could tolerate (*i.e.,* exact computing) under different amount of memristor crossbar redundancy is illustrated in Figure 6. As the results show, in all scenarios, by increasing the redundancy the reliability of the design is improved. By increasing the number of the memristor crossbars from 1 to 4, the maximum tolerable variation is improved about 70%. Also, as we have expected, the tolerable variations under the two scenarios of V+C and V+pC are almost similar due to the low sensitivity of the X-IMM output to the memristor crossbar variation with constant values to the voltage and conductance variations. On the other hand, by omitting the voltage variation, about $1.5\times$ more variation is tolerable by the X-IMM.

V. Conclusion

In this paper, a mixed-signal iteration Montgomery multiplier structure has been introduced. In this structure, in each iteration, the multiplication operations have been done on the memristor crossbar in analog domain. By employing column circuitry including TIA and ADC, the analog output of the memristor crossbar has been converted to the digital domain and in the form of the redundant number system. In the runtime, the proposed X-IMMs had the ability to cascade and make mixed-signal iteration Montgomery multiplier with longer input operands. The studies showed that the latency-area metric of the proposed 1024-bit Radix-4 X-IMM was about 13% more than that of the latest prior work.

References

[1] A. Acar et al., "A survey on homomorphic encryption schemes: Theory and implementation," *ACM Comput. Surv.*, vol. 51, no. 4, July 2018.

[2] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of computation*, vol. 48, no. 177, pp. 203–209, 1987.

[3] Z. Ahmadpour and G. Jaberipur, "Up to $8k$8k-bit modular montgomery multiplication in residue number systems with fast 16-bit residue channels," *IEEE Trans. on Computers*, vol. 71, no. 6, pp. 1399–1410, 2022.

[4] B. Zhang et al., "High-radix design of a scalable montgomery modular multiplier with low latency," *IEEE Trans. on Comp.*, vol. 71, no. 2, 2022.

[5] B. Li, J. Wang, G. Ding, H. Fu, B. Lei, H. Yang, J. Bi, and S. Lei, "A high-performance and low-cost montgomery modular multiplication based on redundant binary representation," *IEEE Trans. on Circuits and Systems II: Express Briefs*, vol. 68, no. 7, pp. 2660–2664, 2021.

[6] B. Zhang et al., "An iterative montgomery modular multiplication algorithm with low area-time product," *IEEE Trans. on Comp.*, 2022.

[7] A. Sebastian et al., "Memory devices and applications for in-memory computing," *Nature nanotechnology*, vol. 15, no. 7, pp. 529–544, 2020.

[8] 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.

[9] A. Fayyazi et al., "An ultra low-power memristive neuromorphic circuit for internet of things smart sensors," *IEEE IoT Jnl.*, vol. 5, no. 2, 2018.

[10] A. BanaGozar et al., "Robust neuromorphic computing in the presence of process variation," in *DATE Conf., 2017*, 2017, pp. 440–445.

[11] T. Wu, "Reducing memory requirements in csa-based scalable montgomery modular multipliers," in *ICSICT)*, 2014, pp. 1–3.

[12] H. Nejatollahi et al., "Cryptopim: In-memory acceleration for lattice-based cryptographic hardware," in *2020 DAC*, 2020, pp. 1–6.

[13] S. Choi et al., "Random telegraph noise and resistance switching analysis of oxide based resistive memory," *Nanoscale*, vol. 6, pp. 400–404, 2014.

[14] X. Guo et al., "Modeling and experimental demonstration of a hopfield network analog-to-digital converter with hybrid cmos/memristor circuits," *Frontiers in Neuroscience*, vol. 9, 2015.

[15] L. Gao et al., "Digital-to-analog and analog-to-digital conversion with metal oxide memristors for ultra-low power computing," in *Int'l Symposium on Nanoscale Architectures*, 2013, pp. 19–22.

[16] V. Yon et al., "Exploiting non-idealities of resistive switching memories for efficient machine learning," *Frontiers in Electronics*, vol. 3, 2022.

[17] W.-Q. Pan et al., "Strategies to improve the accuracy of memristor-based convolutional neural networks," *IEEE Trans. on Electron Devices*, vol. 67, no. 3, pp. 895–901, 2020.

[18] I.-T. W. et al., "3d ta/TaOisubx/sub/i/TiOsub2/sub/ti synaptic array and linearity tuning of weight update for hardware neural network applications," *Nanotechnology*, vol. 27, no. 36, 2016.

[19] X. Guan, S. Yu, and H.-S. P. Wong, "A spice compact model of metal oxide resistive switching memory with variations," *IEEE Electron Device Letters*, vol. 33, no. 10, pp. 1405–1407, 2012.

[20] D. Joksas et al., "Committee machines—a universal method to deal with non-idealities in memristor-based neural networks," *Nature communications*, vol. 11, no. 1, pp. 1–10, 2020.

[21] NanGate, "Nangate freepdk45 open cell library," 2016.

[22] W. Lu et al., "Two-terminal resistive switches (memristors) for memory and logic applications," in *ASP-DAC*, 2011, pp. 217–223.