ModSRAM: Algorithm-Hardware Co-Design for Large Number Modular Multiplication in SRAM

Jonathan Ku[1], Junyao Zhang[1], Haoxuan Shan[1], Saichand Samudrala[2], Jiawen Wu[2], Qilin Zheng[1], Ziru Li[1], JV Rajendran[2], Yiran Chen[1]
[1]Duke University, [2]Texas A&M University
jonathan.ku@duke.edu

ABSTRACT

Elliptic curve cryptography (ECC) is widely used in security applications such as public key cryptography (PKC) and zero-knowledge proofs (ZKP). ECC is composed of modular arithmetic, where modular multiplication takes most of the processing time. Computational complexity and memory constraints of ECC limit the performance. Therefore, hardware acceleration on ECC is an active field of research. Processing-in-memory (PIM) is a promising approach to tackle this problem. In this work, we design ModSRAM, the first 8T SRAM PIM architecture to compute large-number modular multiplication efficiently. In addition, we propose R4CSA-LUT, a new algorithm that reduces the cycles for an interleaved algorithm and eliminates carry propagation for addition based on look-up tables (LUT). ModSRAM is co-designed with R4CSA-LUT to support modular multiplication and data reuse in memory with 52% cycle reduction compared to prior works with only 32% area overhead.

ACM Reference Format:
Jonathan Ku, Junyao Zhang, Haoxuan Shan, Saichand Samudrala, Jiawen Wu, Qilin Zheng, Ziru Li, JV Rajendran, Yiran Chen. 2024. ModSRAM: Algorithm-Hardware Co-Design for Large Number Modular Multiplication in SRAM. In *61st ACM/IEEE Design Automation Conference (DAC '24), June 23–27, 2024, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3649329.3656496

1 INTRODUCTION

Security has become increasingly important in recent years as people care more about privacy and the protection of personal data on the Internet. Public key cryptography (PKC) is commonly used for various applications, such as digital signature and encryption, to name a few. Elliptic curve cryptography (ECC) [10] is one of the popular algorithms. It has the benefit of fewer bitwidth for private keys compared to RSA [22] with the same security level. Another application that is based on ECC is zero-knowledge proof (ZKP) [7], which is an emerging cryptographic protocol that can prove to the verifier that one statement is true without sharing any secret information other than the statement itself.

However, ECC needs to perform modular multiplications for operands with a large bitwidth (at least 224 bits [6]), and a large number of intermediate results will be generated during the computation process. Thus, deploying the ECC algorithm on the hardware efficiently is a challenging issue due to its high memory bandwidth requirement and high computational complexity. For example, [28] mentioned it requires 2.98 TB/s bandwidth in 100 MHz to compute a ZKP scheme, which is impractical for the current systems. To mitigate this problem, processing-in-memory (PIM) is an emerging field of research that aims to minimize the gap between computing and memory units. Previous works [5, 9, 14, 25, 29] have shown promising results in ML PIM and the works [12, 13, 19, 20, 23, 26, 27] have demonstrated possible solutions in cryptographic applications ranging from advanced encryption standard (AES), homomorphic encryption (HE) to post-quantum cryptography (PQC). However, none of them target ECC as the computation requires large bitwidth. As shown in Figure 1, the computation cycles and bitwidth of ECC are higher than PQC. The interface circuit sizes or processing cycles scale up exponentially for large-number modular multiplication. As a result, the existing design methodology for PIM is unsuitable for performing an efficient computation on ECC.

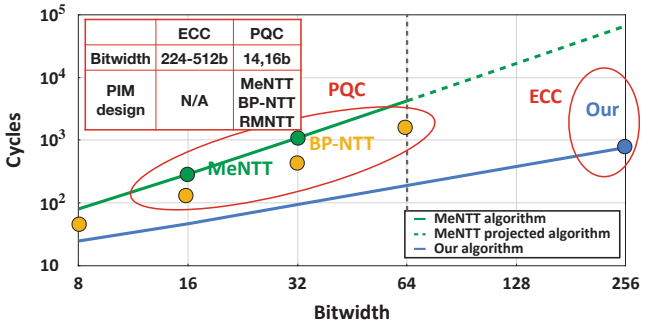


Figure 1: Algorithm complexity and performance comparison with previous work.

To alleviate the computational complexity problem, in this work, we propose an algorithm-hardware co-design methodology customized for PIM-based architecture. Inspired by previous works [8, 15], our proposed algorithm uses a radix-4 encoder and carry save addition features to reduce the computational complexity of the large modular multiplication. In addition, we further customized an SRAM-based PIM architecture to efficiently support the algorithm. Bitwise logic in-memory circuit and simple near-memory circuit features in our proposed SRAM-based PIM architecture provide a significant hardware efficiency improvement due to greater throughput and short critical path.

Overall, our work has the following contributions:

- We propose R4CSA-LUT, a novel algorithm based on interleaved modular multiplication co-designed with ModSRAM. The latency is greatly improved by using a radix-4 encoder to reduce iterative cycles and employing carry-save addition to eliminate intermediate carry propagation.

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06.
https://doi.org/10.1145/3649329.3656496

Algorithm 1 Interleaved Modular Multiplication

Require: n-bit $A = (a_{n-1}, ..., a_0), B, p; 0 \leq A, B \leq p$
Ensure: $C = A \times B \mod p$
1: $C \leftarrow 0$
2: **for** a_i from a_{n-1} to a_0 **do**
3: $C \leftarrow 2 \times C$
4: **if** $C > p$ **then**
5: $C \leftarrow C - p$
6: **end if**
7: $C \leftarrow C + a_i \times B$
8: **if** $C > p$ **then**
9: $C \leftarrow C - p$
10: **end if**
11: **end for**

Algorithm 2 Radix-4 Modular Multiplication

Require: n-bit $A = (a_{n-1}, ..., a_0), B, p; 0 \leq A, B \leq p$
precomputed radix-4 encoding & overflow LUT
Ensure: $C = A \times B \mod p$
1: $C \leftarrow 0$
2: **for** i from $\lceil \frac{n}{2} - 1 \rceil$ to 0 **do**
3: $C \leftarrow 4 \times C$
4: **if** $C > p$ **then**
5: $C \leftarrow LUT(C)$
6: **end if**
7: $E = ENC(a_{2i+1}, a_{2i}, a_{2i-1})$
8: $C \leftarrow C + E \times p$
9: **if** $C > p$ **then**
10: $C \leftarrow C - p$
11: **end if**
12: **end for**

- We design ModSRAM, our cryptographic accelerator. It is an 8T SRAM PIM architecture that is co-designed with R4CSA-LUT. ModSRAM utilizes bitwise logic operations to efficiently compute carry save addition in SRAM with simple in/near-memory circuits. Our accelerator is the first to realize large-number modular multiplication in SRAM.
- ModSRAM is implemented and verified through simulation in TSMC 65nm PDK. We have our result through circuit-level simulation and layout, which achieves 52% fewer cycles with only 32% area overhead under large bitwidth compared to prior works.

2 BACKGROUND AND RELATED WORK

This section provides the necessary background useful in understanding R4CSA-LUT and previous works on logic PIM [1]. One of the applications for logic PIM is cryptography. Even though the target applications from previous works are different than ours, we provide them for completeness.

2.1 Modular Multiplication Algorithms

In modular reduction while doing multiplication, interleaved algorithm [3] shown in Algorithm 1 is the fundamental algorithm. It is based on the traditional shift-and-add fashion to do multiplication with a reduction step in every iteration. The total iterations scale with bitwidth, which can be a serious issue in large numbers. Booth-encoded multipliers [4] are used in modern computers to accelerate multiplication. Instead of iterating through each bit, booth-encoded radix-4 multipliers process three bits at a time with one bit overlapping, which is equivalent to processing two bits in every iteration. Thus, the total iterations are cut in half with the use of an extra encoder. The encoder follows the logic from Table 1a. Radix-8 multipliers are very similar. Four bits are processed with one bit overlapping. As a result, the total iterations are cut down by one-third. The idea of a booth-encoded multiplier can be integrated with the interleaved algorithm as shown in Algorithm 2. Hardware implementation results are shown in [8] with significant reduction.

[1]Logic PIM here is categorized for PIM computing bitwise logic operations, which is in opposition to ML PIM.

Table 1: Radix-4 Computation Tables

(a) Radix-4 Booth Encoder

a_{i+1}	a_i	a_{i-1}	ENC
0	0	0	0
0	0	1	+1
0	1	0	+1
0	1	1	+2
1	0	0	-2
1	0	1	-1
1	1	0	-1
1	1	1	0

(b) Radix-4 Precomputation LUT

ENC	LUT-radix4
0	0
+1	B
+2	$2 \times B \mod p$
-2	$-2 \times B \mod p$
-1	$-1 \times B \mod p$

The work [15] proposed a carry-save addition-based interleaved algorithm to improve performance. For every loop in Algorithm 1, there is a shift, two comparisons followed by subtractions, and a full addition. Shift induces an extra reduction step (comparison then subtraction) since the result is doubled and full addition induces carry propagation, thus increasing hardware resources and latency.

To mitigate this issue, the shift can be considered as adding a new value that is the original value after reduction. The new value can be determined by an extra bit induced by the shift, which we call carry overflow. Since the intermediate results are not our concern, we can adopt carry-save addition to replace full addition. This makes the operation much easier to implement in hardware.

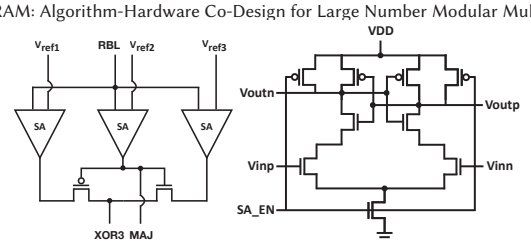
2.2 Cryptographic PIM

Recently, there have been many cryptographic PIM accelerators in SRAM [12, 23, 26, 27] and ReRAM [13, 19, 20] that tried to compute cryptographic schemes in/near-memory. Among all these works, AES and PQC are the most popular. The basic operations in AES are bitwise logic and shift, which are proposed in many logic PIMs [1, 11, 23] already. HE and PQC, on the other hand, is a rising field. HE is an encryption scheme that allows computation directly on ciphertext where plaintext after deciphered, is computed as well. PQC is the field for encryption algorithms that are safe from quantum attacks. No matter the target application, they are all based on polynomial computation, which is usually computed via number theoretical transform (NTT) [21]. It is a generalization for discrete Fourier transform (DFT) over a finite field. The basic operation to do so is modular arithmetic. For these applications, the accelerators are designed for small bitwidth, commonly in 14/16-bit. These designs don't scale with bitwidth in applications such as ECC, where at least 224 bits [6] is recommended to date.

Since the operation in cryptographic PIM can further decompose into bitwise logic and simple logic near-memory, architectures from logic PIM provides the basic design. 2-input logic operations in SRAM are supported in previous works [1, 11] and 3-input logic operations in SRAM are first implemented in [23]. It is the first to realize XOR3 and MAJ (majority) logic functions, which are the sum and carry for addition. The logic-SA module they proposed is illustrated in Figure 2.

3 R4CSA-LUT ALGORITHM

Modular multiplication algorithms can be generalized into two groups as mentioned in Section 1. Montgomery reduction [18] and Barrett reduction [2] are the two most popular methods in reduction after multiplication. Montgomery reduction avoids carry



(a) Logic-SA module (b) SA

Figure 2: Logic-SA module for addition proposed in [23] and latch-type SA structure [24].

Algorithm 3 Proposed Modular Multiplication in-SRAM

Require: n-bit $A = (a_{n-1}, ..., a_0), B, p; 0 \le A, B \le p$
LUT-radix4 & LUT-overflow
Ensure: $C = A \times B \mod p$
1: sum ← 0 ▷ remain n+1 bits
2: carry ← 0 ▷ remain n+1 bits
3: **for** i from $\lceil \frac{n}{2} - 1 \rceil$ to 0 **do**
4: $[\text{overflow}_{\text{sum}}, \text{sum}] \leftarrow \text{sum} << 2$
5: $[\text{overflow}_{\text{carry}}, \text{carry}] \leftarrow \text{carry} << 2$
6: overflow ← $\text{overflow}_{\text{sum}}$ + $\text{overflow}_{\text{carry}}$ + MSB(LUT-radix4)
7: sum ← XOR3(LUT-radix4(a_{i+1}, a_i, a_{i-1}), sum, carry)
8: carry ← MAJ(LUT-radix4(a_{i+1}, a_i, a_{i-1}), sum, carry)
9: carry ← carry << 1
10: sum ← XOR3(LUT-overflow(overflow), sum, carry)
11: carry ← MAJ(LUT-overflow(overflow), sum, carry)
12: carry ← carry << 1
13: **end for**
14: C ← sum + carry

propagation and prevents expensive modular operation by first transforming the operands into Montgomery form. The computations in the Montgomery form are much easier than in its direct form. As a result, the speedup in Montgomery reduction is obvious. Barrett reduction uses another multiplication in place of division for modular reduction. Unfortunately, both of them involve n-bit multiplication, resulting in 2n-bit intermediate results that require more hardware resources to store and compute. In addition, Montgomery reduction requires extra transformation into and out of Montgomery form, which is an unavoidable real modular operation. Barrett reduction induces a 3n-bit intermediate result after the regular multiplication for modular reduction, which takes up even more hardware resources. Both of them reduce the computational latency at the cost of a very complex circuit and memory design in tradeoff. Interleaved modular multiplication [3], on the other hand, is a potential hardware-friendly solution for reduction while doing multiplication. Numerous algorithms have been proposed based on interleaved algorithm as in Section 2.1. The proposed algorithm overview and the mapping to our hardware will be discussed.

3.1 Algorithm Overview

In view of the strengths and weaknesses of previous works, we proposed a new algorithm combining the merits of each algorithm called radix-4 carry save addition, a look-up table based interleaved algorithm (R4CSA-LUT). Since the classical interleaved algorithm has long latency due to a large number of iterations, radix-4 modular multiplication in Algorithm 2 is adopted in R4CSA-LUT to cut iterations in half with only an extra booth encoder as in Table 1a. The value added every iteration can be precomputed as in Table 1b since there are only five possible values and only three of them need computation. These results can be reused as long as the multiplicand remains the same. However, Algorithm 2 still suffers from carry

Table 2: Carry Overflow Precomputation LUT

a_{n+3}	a_{n+2}	a_{n+1}	LUT-overflow
0	0	0	0 (n+1 bits)
0	0	1	001 (0...0) mod p
0	1	0	010 (0...0) mod p
0	1	1	011 (0...0) mod p
1	0	0	100 (0...0) mod p
1	0	1	101 (0...0) mod p
1	1	0	110 (0...0) mod p
1	1	1	111 (0...0) mod p

propagation. This issue seriously affects performance when the numbers to be multiplied become larger. Carry save addition can be adopted into the original radix-4 modular multiplier to eliminate long carry propagation latency as previously mentioned. The values for carry overflow can also be precomputed for eight possible cases. They can be reused as long as the modulo number remains the same. R4CSA-LUT is shown in Algorithm 3. It achieves half iterations compared to an interleaved algorithm without carry propagation via carry-save addition. It is co-designed with our architecture so that the operations are hardware-friendly and data can be reused through LUT, which will be introduced in Section 4.

3.2 Mapping to Hardware

The algorithm can be separated into three parts: precomputation, main iteration and computation for the final result. Precomputation can be stored for later use during the main iteration. The LUTs required to store precomputation results are represented in Tables 1b and 2, which are stored in each wordline (WL) in SRAM. The sum and carry overflow can be used to determine the value added for the next cycle. It depends on the most significant four bits of sum, carry, and the most significant bit (MSB) of radix-4 LUT. They can be computed with a rather low cost compared to the whole modular multiplication because their bitwidths are at most n+3 bits. These results can be reused over multiple iterations and multiple calculations, thus reducing memory movement and maximizing data reuse. For the main iteration, carry save addition is the essential operation and bitwise XOR3 and MAJ logic functions represent the sum and carry, respectively. The left shift by two is due to processing two bits in radix-4 modular multiplication. 3-input logic functions are made possible to compute in-memory by logic-SA module [23] in Figure 2. This provides the fundamental building block to realize R4CSA-LUT in SRAM. The final step is a full addition of the sum and carry in n+1 bits with a reduction step to get the final value. This is inevitable and is best to be computed near-memory. Combining all previous parts, we get our proposed algorithm that can run efficiently on our designed hardware.

4 MODSRAM ARCHITECTURE

4.1 Architecture Overview

Figure 4 illustrates the overall architecture of ModSRAM. It is an SRAM PIM design with custom in/near memory computing circuits to execute the R4CSA-LUT algorithm, which aims to compute modular multiplication in 256 bits efficiently. ModSRAM consists of a 64x256 8T SRAM array with a read port and a write port. The in-memory computing (IMC) circuit is the logic-SA module used to implement XOR3 and MAJ bitwise logic function for carry save addition discussed in detail in Section 4.2. The rest of the peripheral

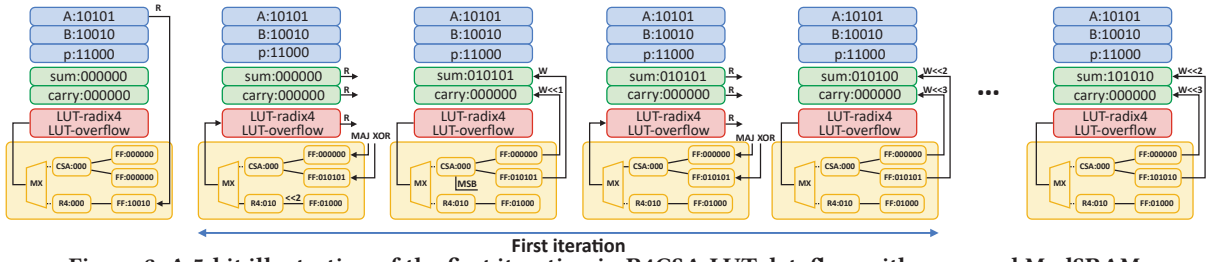


Figure 3: A 5-bit illustration of the first iteration in R4CSA-LUT dataflow with proposed ModSRAM.

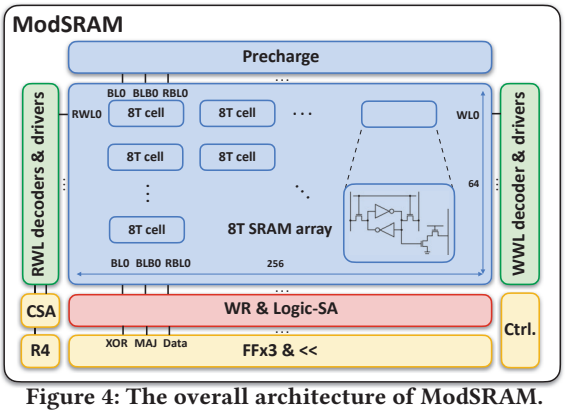


Figure 4: The overall architecture of ModSRAM.

circuits include read wordline (RWL) and write wordline (WWL) decoders as well as near-memory computing (NMC) circuits. They are a radix-4 encoder, combinational logic for overflow, three D flip-flops (DFF) for sum, carry, multiplicand and a controller (Ctrl.), which will be discussed in Section 4.3.

4.2 In-Memory Computing

The IMC part includes precharges, SRAM array and a modified sense amplifier (SA) block to enable logic operation. The SRAM cell is standard 8T that supports one read port and one write port. We adopt this design because our algorithm is based on XOR3 and MAJ logic operations, which are three-input logic operations that open three WLs simultaneously. Traditional 6T SRAM suffers from read disturb since read and write share one single port. This issue is even worse when activating two WLs to enable IMC. Since three WLs will be activated simultaneously in our design, read disturbance is no longer negligible. Therefore, a separate read port is necessary to prevent read disturbance while improving read latency. We adopt the logic-SA module shown in Figure 2a from [23]. Three SAs are used for each read bitline (RBL) to differentiate RBL voltage levels for all the 3-input logic functions in this module, with a total of 256 RBLs. SAs used in ModSRAM are conventional voltage-based latch-type sense amplifiers.

4.3 Near-Memory Computing

Outputs from the IMC circuit are sent to the NMC circuit. They are first stored in FFs, shifted and written back to SRAM for the next iteration. Part of the bits are used to do computation and pass through a MUX to select LUT. To start the iteration, the multiplier is read from SRAM to the near-memory FF. To get the radix-4 encoded computation results in LUT, we take the most significant three bits of the multiplier fetched and encode the following Table 1. For every iteration, the multiplier is shifted to the left by two to get the next value for encoding.

The whole iteration can be partitioned into two sections, which include the first operation for radix-4 LUT and the second operation for overflow LUT. They basically follow the same dataflow, except the data retrieved are in different LUTs, which are different WLs in SRAM. The dataflow for near-memory components is as follows. First, the sum and carry from the previous iteration are shifted to the left by two bits, namely multiplying by four. The overflow bits are stored in a temporary FF for computation in the second section. Next, the encoded result mentioned previously is used to activate WL in radix-4 LUT along with sum and carry. The result from IMC is written back to SRAM with sum first and carry second because during the writeback of sum, carry will be shifted to the left by one bit due to the nature of carry. The overflow bits calculated at the beginning are used to activate WL in overflow LUT along with sum and carry. The result again follows the same datapath.

After the last iteration, we will get n+1 bits of sum and carry, which requires a full addition and reduction to get our final value. However, since the bitwidth is reduced, this step is rather cheap compared to 2n bits without reduction while doing multiplication. The whole NMC circuit is compact as there are only shifters, three full-bitwidth FFs for the multiplier, sum, carry, and some negligible FFs for overflow. Controller for all SRAM operations such as precharge, activating WLs, enabling SA and FSM for near-memory are all realized via Verilog.

4.4 Algorithm Illustration

A simplified version of the 5-bit R4CSA-LUT demonstration on ModSRAM is illustrated in Figure 3. For 5-bit modular multiplication, there are three iterations. In Figure 3, only the first iteration is shown. The first step is to read multiplier A into near-memory FF. Then it will be left shifted by two to select the WL in radix-4 LUT. Three WLs are activated at the same time for IMC. The results of IMC are XOR and MAJ, which will be stored in FFs. They are then left-shifted and written back to SRAM. The next step follows the same, except this time overflow LUT is used for IMC. The final results are shown in the end.

5 EVALUATION AND DISCUSSION

5.1 Evaluation Methodology

We evaluate ModSRAM using TSMC 65nm technology PDK. Full-custom circuits including SRAM array and IMC modules are designed in Cadence Virtuoso. Digital circuits including WL decoders, NMC modules, and a controller are designed in Verilog, and synthesized in Synopsys Design Compiler. Simulations are done in both HSPICE as well as Verilog testbench to get the experimental results.

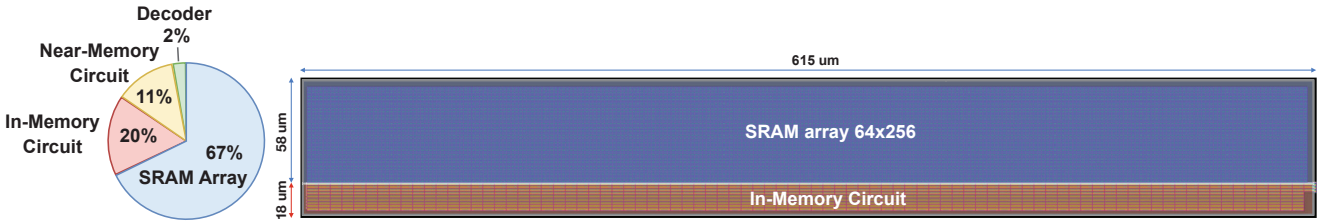


Figure 5: Area breakdown on ModSRAM and full custom layout for SRAM array and in-memory circuit.

A full-custom layout and synthesis result are included in the analysis to get the design area. The area breakdown and full-custom layout are shown in Figure 5.

5.2 Memory Utilization

Since we aim for ECC applications, the security level recommended by NIST is at least 224 bits [6]. Among all the popular elliptic curves (EC), Secp256k1 and BN254 are used for Bitcoin and Zcash, respectively. As a result, we chose 256-bits to be our target. Each WL stores an operand that can be either multiplicand, multiplier, or modulo. Our design is accommodated to fit operands of a point addition operation in EC which are composed of several modular multiplications. During the computation stage, only sum and carry are considered intermediate results that need to be stored in SRAM. Radix-4 and overflow LUTs require a total of 13 WLs, but they can be reused for multiple iterations and for multiple calculations, thus not considered intermediate results. Figure 6 shows the memory utilization comparison for operand storage and intermediate of our work along with existing SRAM PIMs [12, 26]. LUTs are introduced in our work as shown.

5.3 Experimental Results

The number of clock cycles for doing one modular multiplication is recorded in Table 3. For 256-bit, it can be done in 767 cycles with the clock frequency given as 420 MHz. R4CSA-LUT algorithm has a complexity of O(n), which scales linearly to bitwidth. The computation result is in the direct form, so no extra conversion cost is needed. The area achieved is small since it only demonstrates the operation of one modular multiplication. The area breakdown in Figure 5 shows that the memory array occupies two-thirds of the whole design. SAs constitute most of the area in the in-memory circuits with the area of MUX as two transistors negligible. Since our design computes in-memory, the near-memory circuit is compact with very small WL decoders. ModSRAM induces only 32% area overhead by including near-memory circuits and two SAs since the regular SRAM design includes a WL decoder and an SA already.

5.4 Comparison with existing PIM works

Even though no PIM works currently implement large bitwidth modular multiplication for ECC applications, some works demonstrated the possibility of PQC NTT [12, 13, 19, 20, 26]. The problems in previous works motivated our work. Their number of cycles of a single modular multiplication are scaled to meet our bitwidth and compared. The rest of the experimental data are extracted directly from the works, which are shown in Table 3.

Regarding SRAM PIM works, [12] is one of the first SRAM PIMs in PQC NTT. Their access pattern is bit serial as shown in Figure 6,

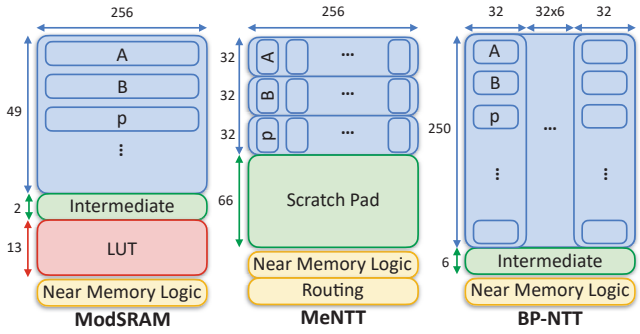


Figure 6: Comparison of data organization for different SRAM PIM designs for modular multiplication.

meaning that the data is stored across the same BL instead of across WL in order to match with their algorithm. This design faces difficulties when scaling the bitwidth because all the operands are stored in the same BL. Doing the computation in 256 bits requires a total of 1282 rows, which is impractical for an SRAM bank. The corresponding algorithm needs $(n+1)^2$ cycles shown in Figure 1 compared to $3n-1$ cycles in our work. Another work [26] improved the performance by adopting a bit-parallel algorithm. It applies the Montgomery transform to avoid carry propagation in their NTT computation. However, the major issue in this design is the transformation cost. They assumed the precomputation of the Montgomery transform for the operands was readily available before they used the inputs in their PIM. However, when the bitwidth increases, the transformation cost is no longer negligible.

As for ReRAM PIM works, [19] introduced PQC NTT with three possible values to choose for modulo. This simplifies the computation yet limits the generality utilized on other applications. [13, 20] on the other hand, solved this issue by providing the modulo as an input. They achieved low latency for NTT at the cost of a large design done only in a simulator instead of in circuit-level simulation. The computations are done with modular reduction after multiplication, therefore no cycle results are presented. To accommodate the need for lossless IMC, both designs required a huge area for analog-to-digital converters (ADC) that occupied more than 70% of the total architecture.

6 FUTURE WORK

This work focuses on the design of a modular multiplier. The goal is to reduce latency and area used in large-scale cryptographic applications by utilizing memory and computing components in-memory. The increases in reusability and compactness make it a desired prototype for further research. This paper serves as a

Table 3: Comparison on modular multiplication in PIM designs.

Reference	This work	MeNTT [12]	BP-NTT [26]	RM-NTT [20]	CryptoPIM [19]	X-Poly [13]
application type	ECC	PQC NTT	PQC NTT	HE NTT	PQC NTT	PQC NTT
Computation method	direct	direct	Montgomery	Montgomery	Montgomery/Barrett	Barrett
technology	65 nm	65 nm	45nm	28nm	45 nm	45nm
Cell type	8T SRAM	6T SRAM	6T SRAM	ReRAM	ReRAM	ReRAM
Array size	64x256	4x162x256	4x256x256	64x4x128x128	512x512	16x128x128
Frequency(MHz)	420	151	3.8k	400	909	400
Bitwidth	256	14/16/32	2/4/8/16/32/64	14/16	16/32	16
Cycles *	767	66049	1465	-	-	-
Area (mm^2)	0.053	0.36	0.063	-	0.152	0.27

* Cycles for other works are generated from frequency, latency and number of modular multiplication in NTT scaled to a fair comparison in the same bitwidth (256b).

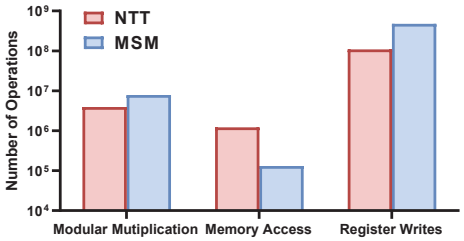


Figure 7: Illustration of the number of operations in ZKP components: NTT [17], and MSM [28], when the input vector is of size 2^{15} and each input bitwidth is 256 bits.

pioneer work on realizing large bitwidth modular operation in-memory that was not possible previously.

With the design of this work as the basis, we plan to integrate the module into a system-level application. In the future, we aim to improve elliptic curve computations, both number theoretical transform (NTT) and multi-scalar multiplication (MSM) algorithms, which are essential in the scheme of ZKP. Figure 7 illustrates the scale of memory accesses, modular multiplications, and their intermediate register writes in ZKP components. The values for NTT in Figure 7 are based on simulations of [16]. The values for MSM are calculated using the architecture in [28]. Our work computes large bitwidth modular multiplications efficiently in-SRAM and avoids intermediate register writes and memory accesses, which can significantly improve the performance of ZKP.

7 CONCLUSION

In this paper, we propose R4CSA-LUT, a new algorithm based on LUTs that combines the merits of both radix-4 modular multiplication and carry save addition in the interleaved algorithm. We also design ModSRAM, an SRAM PIM architecture that aims to compute modular multiplication for ECC based on our co-designed algorithm. The operations in R4CSA-LUT are hardware-friendly and they use LUTs to maximize data reusability. ModSRAM is implemented in state-of-the-art technology and design flow. To the best of our knowledge, we are the first to implement 256-bit modular multiplication in SRAM. We demonstrate a possible solution for combining large-number modular multiplication in SRAM.

ACKNOWLEDGMENTS

This work is supported in part by National Science Foundation (NSF) CCF-2328805 and NSF CNS-2112562.

REFERENCES

[1] Amogh Agrawal et al. 2018. X-SRAM: Enabling In-Memory Boolean Computations in CMOS Static Random Access Memories. *IEEE Transactions on Circuits and Systems I: Regular Papers* 65, 12 (2018), 4219–4232.
[2] Paul Barrett. 1987. Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor. In *Advances in Cryptology — CRYPTO' 86*, Andrew M. Odlyzko (Ed.).
[3] G.R. Blakely. 1983. A Computer Algorithm for Calculating the Product AB Modulo M. *IEEE Trans. Comput.* C-32, 5 (1983), 497–500.
[4] ANDREW D. BOOTH. 1951. A SIGNED BINARY MULTIPLICATION TECHNIQUE. *The Quarterly Journal of Mechanics and Applied Mathematics* (1951).
[5] Fan Chen et al. 2018. ReGAN: A pipelined ReRAM-based accelerator for generative adversarial networks. In *2018 23rd ASP-DAC*.
[6] Lily Chen et al. 2023. Digital Signature Standard (DSS). https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=935202
[7] Shafi Goldwasser et al. 1989. The Knowledge Complexity of Interactive Proof Systems. *SIAM J. Comput.* 18, 1 (1989), 186–208. https://doi.org/10.1137/0218012
[8] Khalid Javeed and Xiaojun Wang. 2014. Radix-4 and radix-8 booth encoded interleaved modular multipliers over general Fp. In *2014 24th FPL*.
[9] Houxiang Ji et al. 2018. ReCom: An efficient resistive accelerator for compressed deep neural networks. In *DATE*.
[10] Neal Koblitz. 1987. Elliptic curve cryptosystems. *Mathematics of computation* 48, 177 (1987), 203–209.
[11] Kyeongho Lee et al. 2020. Bit Parallel 6T SRAM In-memory Computing with Reconfigurable Bit-Precision. In *2020 57th ACM/IEEE DAC*.
[12] Dai Li et al. 2022. MeNTT: A Compact and Efficient Processing-in-Memory Number Theoretic Transform (NTT) Accelerator. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 30, 5 (2022), 579–588.
[13] Mengyuan Li et al. 2023. Accelerating Polynomial Modular Multiplication with Crossbar-Based Compute-in-Memory. *arXiv preprint arXiv:2307.14557* (2023).
[14] Ziru Li et al. 2022. ASTERS: adaptable threshold spike-timing neuromorphic design with twin-column ReRAM synapses. In *Proceedings of the 59th ACM/IEEE Design Automation Conference* (San Francisco, California) *(DAC '22)*. 1099–1104.
[15] Oleg Mazonka et al. 2022. Fast and Compact Interleaved Modular Multiplication Based on Carry Save Addition. In *Proceedings of the 41st IEEE/ACM ICCAD*.
[16] A. C. Mert et al. 2020. Parametric-ntt. https://github.com/acmert/parametric-ntt.
[17] Ahmet Can Mert et al. 2022. An Extensive Study of Flexible Design Methods for the Number Theoretic Transform. *IEEE Trans. Comput.* 71, 11 (2022), 2829–2843.
[18] Peter L Montgomery. 1985. Modular multiplication without trial division. *Mathematics of computation* 44, 170 (1985), 519–521.
[19] Hamid Nejatollahi et al. 2020. CryptoPIM: In-memory Acceleration for Lattice-based Cryptographic Hardware. In *2020 57th ACM/IEEE DAC*.
[20] Yongmo Park et al. 2022. RM-NTT: An RRAM-Based Compute-in-Memory Number Theoretic Transform Accelerator. *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits* 8, 2 (2022), 93–101.
[21] J. M. Pollard. 1971. The fast Fourier transform in a finite field. *Math. Comp.* 25 (1971), 365–374. https://api.semanticscholar.org/CorpusID:123174851
[22] R. L. Rivest, A. Shamir, and L. Adleman. 1978. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM* (feb 1978).
[23] Amitesh Sridharan et al. 2022. A 1.23-GHz 16-kb Programmable and Generic Processing-in-SRAM Accelerator in 65nm. In *IEEE 48th ESSCIRC*.
[24] B. Wicht et al. 2004. Yield and speed optimization of a latch-type voltage sense amplifier. *IEEE Journal of Solid-State Circuits* 39, 7 (2004), 1148–1158.
[25] Bonan Yan et al. 2019. RRAM-based Spiking Nonvolatile Computing-In-Memory Processing Engine with Precision-Configurable In Situ Nonlinear Activation. In *2019 Symposium on VLSI Technology*.
[26] Jingyao Zhang et al. 2023. BP-NTT: Fast and Compact in-SRAM Number Theoretic Transform with Bit-Parallel Modular Multiplication. arXiv:2303.00173
[27] Yiqun Zhang et al. 2018. Recryptor: A Reconfigurable Cryptographic Cortex-M0 Processor With In-Memory and Near-Memory Computing for IoT Security. *IEEE Journal of Solid-State Circuits* 53, 4 (2018), 995–1005.
[28] Ye Zhang et al. 2021. PipeZK: Accelerating Zero-Knowledge Proof with a Pipelined Architecture. In *2021 ACM/IEEE 48th ISCA*. 416–428.
[29] Qilin Zheng et al. 2020. Lattice: An ADC/DAC-less ReRAM-based Processing-In-Memory Architecture for Accelerating Deep Convolution Neural Networks. In *2020 57th ACM/IEEE DAC*.