COPMA: Compact and Optimized Polynomial Multiplier Accelerator for High-Performance Implementation of LWR-Based PQC

Pengzhou He, Yazheng Tu, Tianyou Bao, Leonel Sousa, and Jiafeng Xie

***Abstract*— The rapid progress in quantum computing has initiated a new round of cryptographic innovation, that is, developing postquantum cryptography (PQC) to resist attacks from well-established quantum computers. In this brief, we propose a novel compact and optimized polynomial multiplier accelerator (COPMA) for high-performance implementation of learning-with-rounding (LWR)-based PQC. As not many LWR-based PQC schemes are available in the literature, we have just used Saber, the National Institute of Standards and Technology (NIST) third-round PQC standardization finalist, as a typical case study example. First of all, we have formulated the polynomial multiplication, the major component of Saber, into a novel "subpolynomial"-based processing format for compact computation (yet has the potential for fast operation). Then, we have designed the proposed algorithm into an area-efficient polynomial multiplication hardware accelerator with high-frequency operational capability. Finally, we have verified the efficiency of the developed COPMA and have deployed it to build a cryptoprocessor. The implementation and analysis demonstrate the superior performance of the proposed COPMA. The proposed strategy is highly efficient and can be extended to build other PQC hardware accelerators.**

***Index Terms*— Compact and optimized polynomial multiplier accelerator (COPMA), high-performance, learning-with-rounding (LWR), postquantum cryptography (PQC).**

I. Introduction

There is a pressing need for acceptable alternatives for traditional and extensively used cryptosystems such as Rivest Shamir Adleman and Elliptic Curve cryptography, as these cryptosystems have been proved to be vulnerable against quantum attacks [1], [2]. Therefore, research related to postquantum cryptography (PQC) has drawn increasing attention recently [3], [4]. As a result, various categories of encryption schemes and algorithms have been developed and optimized.

Among all the proposed schemes, the learning-with-rounding (LWR) problem has gained much attention from the research community due to its high quantum attack resistance and relatively simple implementation complexity [4], [5]. LWR is a variant of the learning with errors (LWE) problem, where the errors are produced by a rounding operation [6]. Recently, research has also been carried out on efficient hardware implementations for LWR-based PQC, especially on the field-programmable gate array (FPGA) platform [2], [7].

Manuscript received 30 August 2022; revised 25 December 2022; accepted 14 January 2023. Date of publication 20 February 2023; date of current version 22 March 2023. The work of Jiafeng Xie was supported by the NIST under Grant 60NANB20D20 and by the NSF under Grant SaTC-2020625. *(Corresponding author: Pengzhou He.)*

Pengzhou He, Yazheng Tu, Tianyou Bao, and Jiafeng Xie are with the Department of Electrical and Computer Engineering, Villanova University, Villanova, PA 19085 USA (e-mail: phe@villanova.edu; ytu1@villanova.edu; tbao@villanova.edu; jiafeng.xie@villanova.edu).

Leonel Sousa is with the Electrical and Computer Engineering Department (DEEC), Instituto Superior Técnico (IST), Universidade de Lisboa, 1000-029 Lisbon, Portugal (e-mail: las@inesc-id.pt).

Digital Object Identifier 10.1109/TVLSI.2023.3242640

Polynomial multiplication over the ring is, typically, the key arithmetic operation of LWR-based PQC, which is very obvious in the representative LWR-based scheme (Saber) [6]. As the National Institute of Standards and Technology (NIST) third-round PQC standardization finalist, herein Saber is used as a study example (not many mature LWR-based schemes are available in the literature). However, there still exist challenges for the efficient hardware implementation of this polynomial multiplication: 1) most of the existing designs require relatively large area usage to obtain high-speed operations [8], [9]; and 2) almost all the existing lightweight polynomial multipliers involve long computation time [10], [11], [12]. Besides that, though there exist other hardware implementation strategies such as high-level synthesis or hardware/software co-design, we here focus on full-hardware design as it can lead to the most efficiency [8]. Based on these considerations, we propose a novel "subpolynomial"-based processing strategy to design a polynomial multiplier with the improved area and modest calculation cycles to obtain a novel compact and optimized polynomial multiplier accelerator (COPMA). COMPA was also deployed to build an efficient LWR-based PQC coprocessor. Overall, the **key contributions** of this brief are as follows.

1) Formulating the polynomial multiplier of LWR-based PQC into a novel "subpolynomial"-based processing format for compact computation and potential fast operation.
2) Mapping the proposed algorithm into an efficient polynomial multiplier accelerator COPMA with the optimized area and high-frequency operational capability.
3) Deploying the proposed COPMA (efficiency verified) to build an LWR-based PQC coprocessor with comparison to showcase the efficiency of the proposed design strategy.

The rest of the brief is arranged as follows. Section II presents the proposed algorithm. The accelerator is presented in Section III. The complexity and implementation of COPMA are shown in Section IV. Further cryptoprocessor building is given in Section V. Conclusion are drawn in Section VI.

II. COPMA: Mathematical Derivation

Brief overview: We have used the key encapsulation mechanism (KEM) Saber, a module-LWR-based PQC [6], as a study case applying the related notations and parameters throughout the brief. For details of Saber, one can refer to [6].

Contribution-I: Overall, we have proposed a novel "subpolynomial"-based polynomial multiplication algorithm for Saber for compact computation and potential fast operation.

Definition: Let us define the polynomial multiplication as $W = \sum_{i=0}^{N-1} w_i x^i = DB \bmod f(x)$, where $D = \sum_{i=0}^{N-1} d_i x^i$ (d_i is 13-bit), $B = \sum_{i=0}^{N-1} b_i x^i$ (b_i is 4-bit), and w_i is 13-bit [6]. The size of b_i and d_i are specified by Saber (also $f(x)$), but the proposed

algorithm is also applicable to the polynomial multiplication of arbitrary unequal-sized polynomials such as [13] (which can be one of our future works). We further have

$$W = d_0\left(b_0 + \cdots + b_{n-1}x^{N-1}\right) \bmod f(x) + \cdots + d_{N-1}x^{N-1}\left(b_0 + \cdots + b_{N-1}x^{N-1}\right) \bmod f(x). \quad (1)$$

Then, since for $f(x) = x^N + 1$, $x^N \equiv -1$, which can be substituted into (1) to execute the modulo operation as

$$W = d_0\left(b_0 + b_1x + \cdots + b_{N-1}x^{N-1}\right) + \cdots + d_{N-1}\left(-b_1 - b_2x - \cdots - b_{N-1}x^{N-2} + b_0x^{N-1}\right). \quad (2)$$

Without loss of generality, we can define $B^{(0)} = b_0 + \cdots + b_{N-1}x^{N-1} = B, \ldots, B^{(N-1)} = -b_1 - b_2x - \cdots - b_{N-1}x^{N-2} + b_0x^{N-1}$. Thus, we can rewrite (1) as

$$W = \sum_{i=0}^{N-1} B^{(i)}d_i. \quad (3)$$

Existing strategy: The existing high-speed designs are either based on polynomial-wise-based schoolbook algorithms [8], [11] (similar to (3), relatively large area) or Karatsuba algorithm (incurs very large area). While considering the existing compact designs ([10], [11], [12]), not much algorithmic derivation has been made: [10] is based on the Toom-Cook method, [11] is a lightweight version of the high-speed architecture, [12] is based on the technique of sharing of two point-wise multipliers within one DSP, and [14] is a hardware/software co-design and no specific algorithm has been presented.

Proposed derivation strategy: Based on the above discussion, we propose here a novel "subpolynomial"-based polynomial multiplication algorithm, that is, only one part of the polynomial is being accumulated to obtain compact implementation and possibly fast operation. We thus define

$$B^{(i)} = \sum_{j=0}^{u-1} B_j^{(i)} \quad (4)$$

where $B_j^{(i)}$ is a subpolynomial with v coefficients, for $N = uv$ (u and v are integers). For instance, we have

$$\begin{aligned} B_0^{(0)} &= b_0 + \cdots + b_{v-1}x^{v-1}, \ldots \ \ldots \ \ldots \\ B_{u-1}^{(0)} &= b_{N-v}x^{N-v} + \cdots + b_{N-1}x^{N-1} \end{aligned} \quad (5)$$

which similarly applies to other $B_j^{(i)}$ $(1 \le i \le N-1)$.

Therefore, (3) can again be

$$W = \sum_{i=0}^{N-1} B^{(i)}d_i = \sum_{i=0}^{N-1}\sum_{j=0}^{u-1} B_j^{(i)}d_i \quad (6)$$

where the polynomial multiplication is computed by: 1) when $i = 0$, execute $\sum_{j=0}^{u-1} B_j^{(0)}d_0$; 2) then switches to $i = 1$ and do the same computation process, where the results are, respectively, accumulated with the previous round of computation; and 3) repeat the computation process until $i = N - 1$.

We can finally have [from (6)]

$$W = \sum_{j=0}^{u-1} W_j \quad (7)$$

where $W_j = \sum_{i=0}^{N-1} B_j^{(i)}d_i$. We can have the algorithm as shown in Algorithm 1 [following (4)–(7)]. This whole computation process (Algorithm 1), undoubtedly, fulfills the proposed derivation strategy that the final result is delivered through "subpolynomial"-based accumulation and related operations.

Algorithm 1 Proposed Polynomial Multiplication Algorithm Applied to the LWR-Based PQC (Saber)

Input : D and B polynomials (D is a polynomial with 13-bit integer coefficients and B is a polynomial with 4-bit integer coefficients).
Output: $W = DB \bmod f(x)$ $(f(x) = x^N + 1)$. W is a polynomial with 13-bit integer coefficients
Initialization step
1 $\overline{W} = 0$;
Main step
2 **for** $i = 0$ *to* $N - 1$ **do**
3 // sequential, by clock
4 **for** $j = 0$ *to* $u - 1$ **do**
5 // concurrent, by multiplication unit
 $\overline{W} = \overline{W} + B_j^{(i)}d_i$. // following (6)
6 **end**
7 $W_j = \overline{W}$;
8 **end**
Final step
9 Obtain the final output W from serially delivered W_j;

III. COPMA: Hardware Structure

Brief overview: In this section, the proposed polynomial multiplication hardware accelerator is presented by applying Algorithm 1, which consists of four main components: 1) the input processing component; 2) the multiplication component; 3) the accumulation and output component; and 4) the control unit.

Contribution-II: We have proposed efficient algorithm-to-architecture mapping techniques to design this accelerator. Specifically, we have proposed a new accumulation format to realize the proposed "subpolynomial"-based operation with compact resource usage and high-frequency capability.

Input processing component: This component functions to make the input polynomials ready for the computation of Algorithm 1. As shown in Fig. 1, this component contains two circular shift registers (CSRs) to load in d_i and b_i, namely CSR-I and CSR-II, respectively. CSR-I is a generic serial-in–serial-out shift register, which takes one coefficient (13 bit) per cycle (during loading) and sends out one coefficient every u cycles during the multiplication process. Unlike CSR-I and CSR-II is a serial-in–parallel-out circular shift register, as shown in Fig. 2, which takes one coefficient per cycle in the loading mode and delivers v coefficients out in parallel during the multiplication process. CSR-II also shifts coefficients by $(v - 1)$ positions to obtain all $B_j^{(i)}$, and the switching of the modes for both CSR-I and CSR-II are orchestrated by the control unit.

Multiplication component: The multiplication component executes the N point-wise multiplications of step 5 of Algorithm 1. As shown in Fig. 3, this component consists of two parts, that is, the precalculator and the selector. The precalculator reads in one coefficient d_i and then calculates 2–4 multiples of d_i. The generated multiples are then fed to the selector part. Then, v selectors choose the corresponding multiple of d_i by reading b_i. One MUX then selects the positive/negative value of the desired result. In this way, the fan outs of the MUXes are significantly reduced because we only need

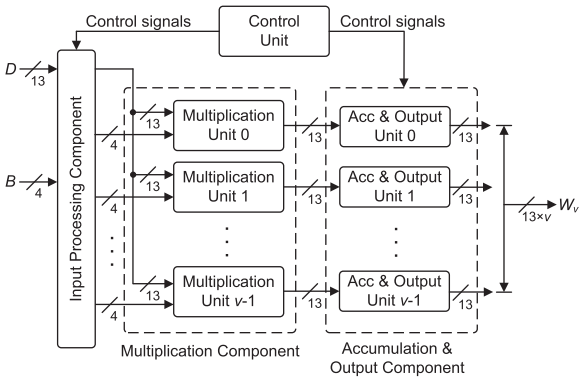

Fig. 1. COPMA: the proposed hardware accelerator.

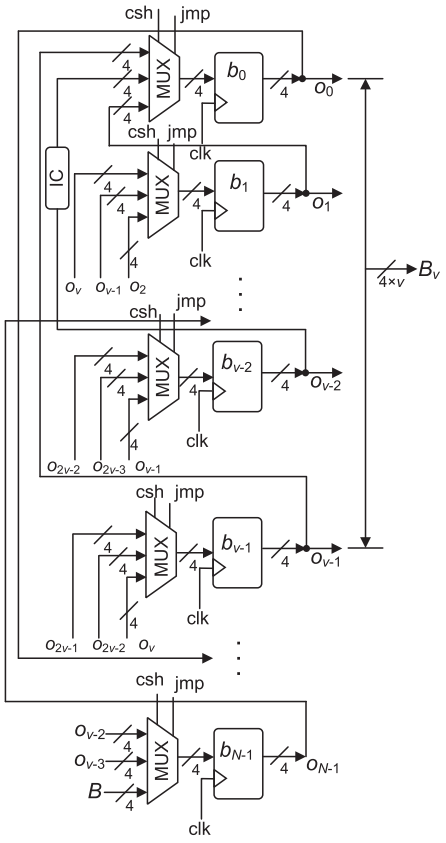

Fig. 2. Internal structure of CSR-II for B (IC: inverter cell).

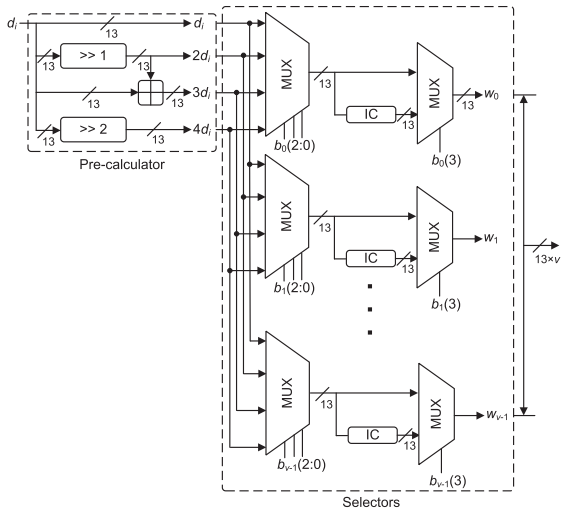

Fig. 3. Multiplication component (IC).

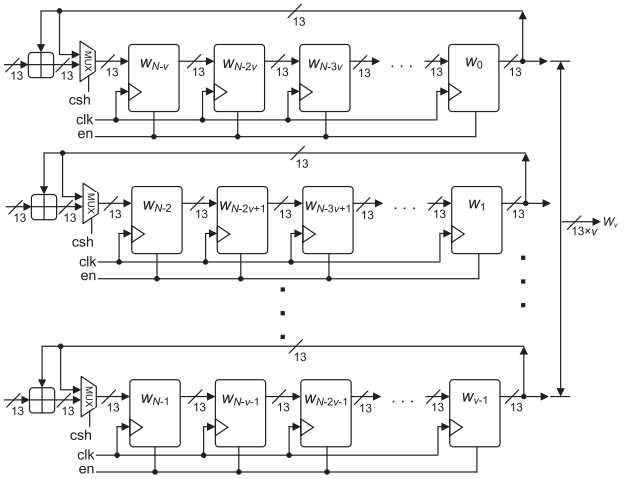

Fig. 4. Details of the accumulation and output component.

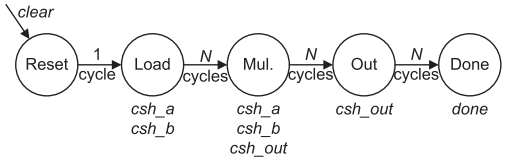

Fig. 5. FSM with major control signals generated in each state.

to select from 5 multiples (0–4) rather than 9 (−4–4), as the b_is are denoted in the sign-magnitude format.

Accumulation and output component: We have designed a novel accumulation component to realize the "subpolynomial"-based operation with the compact area and high-frequency capability. As seen from Fig. 4, this component consists of v shift registers and v adders. Each shift register is composed of $13 \times N/v$ 1-bit registers. Each adder takes in a 13-bit product sent out from the multiplication core along with the output from the tail shift registers as inputs, then adds the two input together, and loads it to the head of the shift register which will shift a 13-bit chunk per cycle. The products are accumulated and stored in the accumulation component and finally shifted out in a correct order, which innovatively realizes the proposed "subpolynomial"-based accumulation. As the accumulation and output delivery are contained in the same component, resource usage is minimized. Meanwhile, as a large number of registers are involved, the designed component enables the accelerator to operate at a high-frequency mode.

Control unit: The control unit produces signals such as "load," "reset," "shift," and "jump" to coordinate overall operation of COPMA. We used a finite-state machine (FSM) with five consecutive states, namely "reset," "load," "calculate," "output," and "done," to produce related control signals (Fig. 5). Here, *csh_b*, *csh_d*, and *csh_out* are the signals determining whether the shift register for B, D, and products of coefficients are taking in or shifting out the data.

IV. Implementation and Comparison

Contribution-III.a: In this section, we provide the FPGA-based implementation (and comparison) to confirm the efficiency of the proposed COPMA.

FPGA-based implementation: The proposed design was coded in VHDL and tested by Modelsim and then was implemented by

AMD-Xilinx Vivado 2020.2 for $N = 256$ and b_i as $[-4, 4]$ [6]. To evaluate the compactness of the proposed design, we have implemented the design with different v. Due to the available hardware resources on the FPGA devices, we implemented the proposed accelerator for $v \leq 64$ (v, a power of 2) on the Artix-7 XC7A12TLCSG325, while this limit grows to $v = 128$ for the UltraScale+ XCZU9EG-2FFVB1156. The results are listed in Table I, including LUT, register (FF), slices, and maximum frequency (Fmax), latency, and the area–delay product (ADP). Note that the existing designs did not report the slice usage and we just used the number of LUTs to calculate ADP. Nevertheless, as the existing designs such as [10], [12] also used other resources (DSPs and BRAMs), their actual ADPs are higher than those listed.

Performance discussion: As shown in Table I, the area of the proposed design increases as v grows. Also, as more components are getting involved as v increases (causing a larger delay of signal propagation), the maximum frequency gradually decreases. However, we notice that the latency drops significantly as v doubles, which provides a better computation time. Thus, we can see that ADP constantly decreases as v increases: a larger v would lead to better overall efficiency.

Comparison: As shown in Table II, our proposed design provides much better area–time complexities than the state-of-the-art. For $v = 64$, the ADP reduction is at least 67.50%, 41.05%, and 12.2% compared to [10], [11], [12], respectively. Note the design of [12] needs extra two BRAMs and two DSPs and its actual ADP is higher than the listed value in Table II (the same to [10]). It is noted that for a fair comparison, we only compare the proposed one with the existing compact designs.

TABLE I
IMPLEMENTATION RESULTS FOR THE PROPOSED COPMA

design	LUT	FF	Slice	Fmax[1]	latency[2]	ADP[3]
UltraScale+ XCZU9EG-2FFVB1156 FPGA device						
$v = 4$	4,684	4,781	810	474	16,384	161,904
$v = 8$	4,882	4,784	837	465	8,192	86,007
$v = 16$	5,277	4,784	854	431	4,096	50,150
$v = 32$	6,135	4,786	995	429	2,048	29,288
$v = 64$	7,767	4,792	1,379	400	1,024	19,883
$v = 128$	11,028	4,816	1,680	387	512	14,590
Artix-7 XC7A12TLCSG325-2L FPGA device						
$v = 4$	4,667	4,781	1,420	143	16,384	534,714
$v = 8$	4,836	4,784	1,448	135	8,192	293,456
$v = 16$	5,192	4,784	1,561	129	4,096	164,856
$v = 32$	5,926	4,785	1,737	129	2,048	94,080
$v = 64$	7,641	4,801	2,165	126	1,024	62,098

[1]Unit for Fmax: MHz. [2]: Number of cycles needed for computation.
[3]: ADP=#LUT×(latency/Fmax). Unit: LUT/MHz.

TABLE II
COMPARISON WITH EXISTING DESIGNS

design	LUT	FF	Slice	Fmax[1]	latency[2]	ADP*
Artix-7 XC7A12TLCSG325-2L FPGA device						
[10][3]	2,927	1,279	-	125	8,176	$> 191,449$
[11][4]	541	301	-	100	19,471	$> 105,338$
[12][5]	561	302	-	130	16,384	$> 70,703$
$v = 32$	5,926	4,785	1,737	129	2,048	94,080
$v = 64$	7,641	4,801	2,165	126	1,024	62,098

[1]Unit: MHz. [2]: Computation cycles. [3]: [10] also needs 38 DSPs.
[4]: also needs 1 BRAM. [5]: [12] also needs 2 BRAMs and 2 DSPs.
*: Though ADP is calculated as ADP=#LUT×(latency/Fmax). the existing designs' actual ADPs are higher than listed due to the extra resource usage.

V. EXTENSION TO THE LWR-BASED PQC COPROCESSOR

Contribution-III.b: We further deployed COPMA to construct an LWR-based PQC coprocessor that operates key generation, encapsulation, and decapsulation of Saber, following the existing design style in [8]. The implementation and comparison confirm the efficiency of the proposed design.

This coprocessor contains the processor interface and control unit, data RAM, program RAM, data bus and manager, and individual arithmetic building blocks (i.e., binomial sampler, Keccak core, polynomial multiplier (COPMA), etc.), as shown in Fig. 6. We have made several adjustments to the polynomial multiplier while following the same construction for the other components. We adjusted the input size of the shift registers to 64-bit, instead of 4 and 13 bits, to interface correctly with the RAM. We added a counter for the "load" state in the control unit to enable/disable the two shift registers at the proper time to load B and D, consecutively. An additional signal *read* is introduced when all the modular polynomial multiplications are done. The control unit will not go to the "output" state until it receives the *read* signal, so that it can keep accumulating the multiplication result and output afterward. We also changed the output size from $13 \times v$-bit to 64-bit so that the multiplier can output 1 word per cycle at the "output" state.

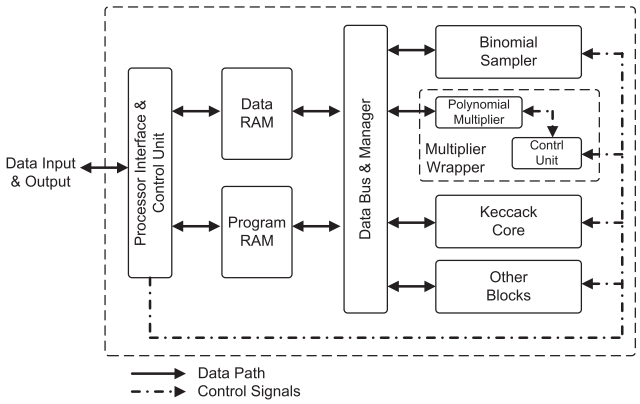

Fig. 6. High-level description of the proposed instruction-set compact LWR-based PQC coprocessor.

Implementation and comparison: We have implemented the coprocessor, for $v = 64$ and $v = 128$, respectively, on the UltraScale+ XCZU9EG-2FFVB FPGA using Vivado 2020.2.

As shown in Table III, as different designs used different resources, it is hard to use a normalized metric such as ADP for comparison. Nevertheless, it is seen that our coprocessor possesses the highest frequency, which is more than 1.5× higher than the existing high-performance Saber implementation [8], for both $v = 64$ and $v = 128$. Although the number of cycles for KenGen./Enca./Deca. is slightly higher than the existing high-performance implementation [8], our coprocessor still has a good delay time due to the high operating frequency. Also, when comparing with the existing work in [8] (best balanced one in the literature), for $v = 64$, although the delay of the proposed coprocessor is higher, the circuit area is only 85.74%; for $v = 128$, the novel design beats the existing work with a 9.56% less area [the number of configurable logic blocks (CLBs)] used, while the delay time is similar; the constructed coprocessor consumes 26.55% and 15.89% less than [8], for $v = 64$ and $v = 128$, respectively. Note that the design of [8] has already shown its efficiency over [10], [14]. When comparing with [15], the proposed design ($v = 128$) has roughly doubled the delay, but with almost half the number of LUTs and FFs. While the design of [15] has used 64 additional DSPs, the existing one of [15] has more extra resource usage than the proposed one, that is, the overall area–time

TABLE III
COMPARISON OF THE AREA–TIME COMPLEXITIES FOR THE PROPOSED AND EXISTING HARDWARE IMPLEMENTATIONS ON FPGA

Design	Device	Freq. (MHz)	Time[1] (KenGen./Enca./Deca.) (μs)	Area				
				LUT	FF	CLB	DSP	BRAM
Comparing with Existing High-Performance LWR-based PQC Implementation (High-Performance)								
$v = 64$	UltraScale+	390	33.07/42.42/52.40	16.6k	12.5k	2,833	0	2
$v = 128$	UltraScale+	390	21.32/26.75/32.83	20k	12.5k	3,244	0	2
[8]	UltraScale+	250	21.8/26.5/32.1	23.6k	9.8k	3,857	0	2
[10]	Artix-7	125	3.2k/4.1k/3.8k	7.4k	7.3k	-	28	2
[14][1]	UltraScale+	322	-/60/65	$\simeq$12.5k	11.6k	-	256	4
[15][3]	UltraScale+	250	10.2/12.6/15.6	41.5k	22.3k	-	64	2
[16][2,3]	UltraScale+	200	54.9/69.7/94.9	18.5k	9.3k	-	4	24
With Existing High-Performance LWR-based PQC (Public-Key Encryption Scheme Only) Implementation								
[9][4]	UltraScale+	160	6.7/7.2/2.6	28.1k	9.5k	-	85	6

[1]: Keccak is executed by the software (no reported related resources).
[2]: The result is from a unified cryptoprocessor for lattice-based signature and key-exchange schemes. [3]: These designs are pre-printed results.
[4]: The FPGA version in [9] only supports the public-key encryption scheme, but not the key encapsulation mechanism (KEM) scheme.

complexities of the proposed design is better than [15]. Besides that, it is clear that the design of [16] exhibits significantly less performance at the approximate same cost than the proposed one. Finally, when comparing [9], the proposed design has efficiency in much smaller resource usage, while the one of [9] covers only a public-key encryption scheme.

Moreover, as shown in Table III, the proposed design used relatively a larger number of FFs (yet still with small CLBs) to achieve high-frequency operation, which leads to the overall area–time efficiency. Meanwhile, after carefully measuring and comparing the performance for different choices of v, we conclude that the coprocessor reaches its best operating efficiency at $v = 128$. With this setup, the coprocessor reaches a high operating speed with a relatively low area usage, which is suitable for high-performance applications. On the other hand, the proposed design of $v = 64$ uses considerably less resource usage with a relatively slow speed and hence is preferred for compact applications. Practically, users can always select the ideal v to obtain the optimized design for specific applications.

Discussion and related works: While this work aims to develop an efficient COPMA, future work may focus on its extension and side-channel attacks (though the proposed design has constant time and is resistant to timing attacks).

Other works also include the regular ring-LWE-based PQC designs [17], [18], a polynomial multiplier accelerator [19], and high-level synthesis of PQC schemes [20]. As these designs used different study cases, we do not directly compare them though they are also important works in the PQC field.

VI. CONCLUSION

This brief proposes COPMA, a novel hardware polynomial multiplier accelerator for LWR-based PQC. We first derived the proposed algorithm by using Saber as a study case. A novel accelerator is then proposed, along with the implementation result on FPGA devices (with comparison). A PQC coprocessor was built deploying the proposed polynomial multiplier. A comparative evaluation with the state-of-the-art shows the superior efficiency of the proposed COPMA.

REFERENCES

[1] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proc. 35th Annu. Symp. Found. Comput. Sci.*, 1994, pp. 124–134.

[2] J. Xie, K. Basu, K. Gaj, and U. Guin, "Special session: The recent advance in hardware implementation of post-quantum cryptography," in *Proc. IEEE 38th VLSI Test Symp. (VTS)*, Apr. 2020, pp. 1–10.

[3] D. Micciancio and O. Regev, "Lattice-based cryptography," in *Post-Quantum Cryptography*. Cham, Switzerland: Springer, 2009.

[4] *PQC Round 3 Submissions*. [Online]. Available: https://csrc.nist.gov/projects/postquantumcryptography/round-3-submissions

[5] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *J. ACM*, vol. 56, no. 6, pp. 1–40, 2009.

[6] J.-P. D'Anvers et al., *SABER: Mod-LWR Based KEM (Round 3 Submission)*, document, 2021, pp. 1–44.

[7] G. Alagic et al., "Status report on the second round of the NIST post-quantum cryptography standardization process," U.S. Dept. Commerce, NIST, Gaithersburg, MD, USA, Tech. Rep., 2020.

[8] S. S. Roy and A. Basso, "High-speed instruction-set coprocessor for lattice-based key encapsulation mechanism: Saber in hardware," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2020, pp. 443–466, Aug. 2020.

[9] Y. Zhu et al., "LWRpro: An energy-efficient configurable crypto-processor for module-LWR," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 3, pp. 1146–1159, Mar. 2021.

[10] J. M. B. Mera, F. Turan, A. Karmakar, S. S. Roy, and I. Verbauwhede, "Compact domain-specific co-processor for accelerating module lattice-based KEM," in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2020, pp. 1–6.

[11] A. Basso and S. S. Roy, "Optimized polynomial multiplier architectures for post-quantum KEM saber," in *Proc. 58th ACM/IEEE Design Autom. Conf. (DAC)*, Dec. 2021, pp. 1285–1290.

[12] Y. Zhang, Y. Cui, Z. Ni, D.-E.-S. Kundi, D. Liu, and W. Liu, "A lightweight and efficient schoolbook polynomial multiplier for saber," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2022, pp. 2251–2255.

[13] B. J. Lucas et al., "Lightweight hardware implementation of binary ring-LWE PQC accelerator," *IEEE CAL*, vol. 21, no. 1, pp. 17–20, Jan. 2022.

[14] V. B. Dang, F. Farahmand, M. Andrzejczak, and K. Gaj, "Implementing and benchmarking three lattice-based post-quantum cryptography algorithms using software/hardware codesign," in *Proc. Int. Conf. Field-Program. Technol. (ICFPT)*, Dec. 2019, pp. 206–214.

[15] W. Tan, A. Wang, Y. Lao, X. Zhang, and K. K. Parhi, "Low-latency VLSI architectures for modular polynomial multiplication via fast filtering and applications to lattice-based cryptography," 2021, *arXiv:2110.12127*.

[16] A. C. Mert et al., "A unified cryptoprocessor for lattice-based signature and key-exchange," *IEEE Trans. Comput.*, pp. 1–13.

[17] W. Liu, S. Fan, A. Khalid, C. Rafferty, and M. O'Neill, "Optimized schoolbook polynomial multiplication for compact lattice-based cryptography on FPGA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 10, pp. 2459–2463, Oct. 2019.

[18] S. Bian, M. Hiromoto, and T. Sato, "Filianore: Better multiplier architectures for LWE-based post-quantum key exchange," in *Proc. 56th Annu. Design Autom. Conf.*, Jun. 2019, pp. 1–6.

[19] Y. A. Birgani et al., "Area-time-efficient scalable schoolbook polynomial multiplier for lattice-based cryptography," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 60, no. 12, pp. 5079–5083, Dec. 2022.

[20] K. Basu, D. Soni, M. Nabeel, and R. Karri, "NIST post-quantum cryptography—A hardware evaluation study," *Cryptol. ePrint Arch.*, pp. 1–16, 2019.