CROP: FPGA Implementation of High-Performance Polynomial Multiplication in Saber KEM based on Novel Cyclic-Row Oriented Processing Strategy

Jiafeng Xie, Pengzhou He
Villanova University, Villanova, PA, USA
e-mail: jiafeng.xie,phe@villanova.edu

Chiou-Yng Lee
Lunghwa University of Science & Technology, Taiwan
e-mail: pp010@gm.lhu.edu.tw

***Abstract*—The rapid advancement in quantum technology has initiated a new round of post-quantum cryptography (PQC) related exploration. The key encapsulation mechanism (KEM) Saber is an important module lattice-based PQC, which has been selected as one of the PQC finalists in the ongoing National Institute of Standards and Technology (NIST) standardization process. On the other hand, however, efficient hardware implementation of KEM Saber has not been well covered in the literature. In this paper, therefore, we propose a novel cyclic-row oriented processing (CROP) strategy for efficient implementation of the key arithmetic operation of KEM Saber, i.e., the polynomial multiplication. The proposed work consists of three layers of interdependent efforts: (i) first of all, we have formulated the main operation of KEM Saber into desired mathematical forms to be further developed into CROP based algorithms, i.e., the basic version and the advanced higher-speed version; (ii) then, we have followed the proposed CROP strategy to innovatively transfer the derived two algorithms into desired polynomial multiplication structures with the help of a series of algorithm-architecture co-implementation techniques; (iii) finally, detailed complexity analysis and implementation results have shown that the proposed polynomial multiplication structures have better area-time complexities than the state-of-the-art solutions. Specifically, the field-programmable gate array (FPGA) implementation results show that the proposed design, e.g., the basic version has at least less 11.2% area-delay product (ADP) than the best competing one (Cyclone V device). The proposed high-performance polynomial multipliers offer not only efficient operation for output results delivery but also possess low-complexity feature brought by CROP strategy. The outcome of this work is expected to provide useful references for further development and standardization process of KEM Saber.**

***Index Terms*—Cyclic-row oriented processing (CROP) strategy, field-programmable gate array (FPGA), high-performance, key encapsulation mechanism (KEM) Saber, polynomial multiplication, post-quantum cryptography (PQC)**

I. Introduction

Along with the rapid advancement in quantum computing, it has been proven that the well-established quantum computer employing Shor's algorithm can break the current public-key cryptosystems such as Rivest Shamir Adleman (RSA) and Elliptic Curve Cryptography (ECC) [1]–[4]. Therefore, post-quantum cryptography (PQC) and related implementations have drawn significant attention from the research community recently [3]. As indicated by the National Institute of Science and Technology (NIST) 3rd round PQC standardization process [5], there are lattice-based cryptography, code-based cryptography, isogeny-based cryptography (etc.) currently under consideration for PQC candidates. Among these candidates, lattice-based cryptography is recognized as one of the most promising schemes [5]–[7].

The lattice-based cryptography can be built on the learning-with-errors (LWE) problem and its variants [6-11]. One of the important variants of the LWE is the learning-with-rounding (LWR) problem [12]. Quite a good number of works have been released on this problem [13-23], including the key encapsulation mechanism (KEM) Saber [5], which is one of the NIST 3rd round PQC finalists.

Existing Works. KEM Saber is built on the Module-LWR (MLWR) problem, which is a module variant of LWR. Upon its original introduction in [13], [14], many works have been released on this interesting PQC scheme, ranging from security level, implementation, and attack analysis [14]–[16]. Especially for the hardware implementations, including both the system-level and component-level designs (such as polynomial multiplication), we can categorize them into two types: (i) hardware-software co-design; and (ii) full hardware design. The first type includes the recent one of [17], where the Toom-Cook method is used to implement the polynomial multiplication for KEM Saber. Another recent design of [18] also uses the Toom-Cook approach to achieve high-performance implementation. A very recent report has proposed to use the number theoretic transform (NTT) [19] for the implementation of the polynomial multiplication of KEM Saber on the RISC-V accelerator. For the second type, a new coprocessor for KEM Saber is introduced in [21], where the polynomial multiplication is based on a schoolbook based method. A Karatsuba algorithm based KEM Saber is reported in [22] for high-performance operation. Optimized polynomial multiplication structures for the polynomial multiplication in KEM Saber is recently presented in [23], which has better area-time complexities than the previous designs of [17], [20]. Overall, these two types of designs are the major hardware implementation works for KEM Saber.

The polynomial multiplication over ring $\mathbb{Z}_l/(x^N+1)$ (l is either q or p [13], [21]) is the critical arithmetic operation of KEM Saber. But the existing works have not well covered its efficient implementation: (i) the existing high-performance

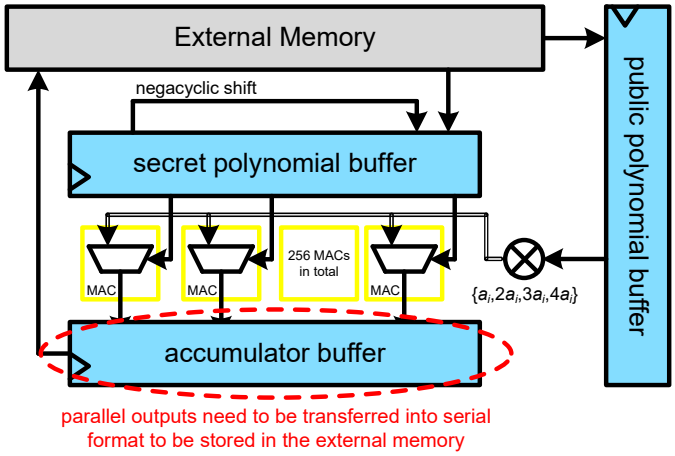

Fig. 1: The existing polynomial multiplication structure [23], where the produced outputs (in parallel) need to be transferred into serial format to be stored in the external memory.

works, such as the structure of Fig. 1 in [23] (see Fig. 1 here) produces the multiplication outputs in a parallel format, which actually requires extra resources such as multiplexers (MUXes) to transfer the parallel outputs into serial style to be stored in the external memory for further usage; (ii) not many efficient hardware structures for the polynomial multiplication have been proposed for KEM Saber. Noticing that polynomial multiplications over other fields such as binary field have been investigated widely in the literature [24]–[27], we just follow this trend to propose an efficient implementation of polynomial multiplication for KEM Saber on the field-programmable gate array (FPGA) platform for high-performance applications. Specifically, we have followed the design style presented in [28] and have proposed a novel cyclic-row oriented processing (CROP) strategy that all the outputs are circularly accumulated and can be very easily transferred to the external memory in a serial format with little extra resource usage (simple operation).

Major Contributions. In total, we have carried out three layers of innovative works for the efficient implementation of polynomial multiplication (KEM Saber), as:

- We have proposed a series of thorough mathematical derivation to lay a solid foundation for the novel CROP strategy, which is then extended further to obtain two algorithms, namely the basic version and the advanced higher-speed one (different processing throughput rates).
- We have then presented a detailed design process to innovatively transfer the CROP originated algorithms into desired multiplication architectures with the help of several algorithm-architecture co-implementation techniques.
- We have conducted a thorough complexity analysis and comparison (including both the theoretical analysis and FPGA based implementation performance) to show that the proposed polynomial multiplications have better area-time complexities than the state-of-the-art solutions.

Overall, the proposed polynomial multiplication possesses three main unique features: (i) simple and easy operation on the output result delivering; (ii) flexible offering of processing throughput; and (iii) low-complexity. Discussions about the further extension and application of the proposed polynomial multiplications have also been provided.

The rest of this paper is organized as follows. The preliminary knowledge is introduced in Section II. The formulation of the proposed CROP strategy is detailed presented in Section III along with proposed algorithms. The proposed hardware polynomial multiplication structures are provided in Section IV. Complexity analysis and comparison are presented in Section V. Finally, conclusions are given in Section VI.

II. Preliminaries

In this section, we briefly give the introduction of the KEM Saber and the involved polynomial multiplication. Interested readers can refer to the original papers of [13], [14] for details.

A. The MLWR Scheme (KEM Saber)

The LWR is a variant of LWE [12], which uses the rounding operation to replace the previous Gaussian distributed errors to obtain the hardness of the lattice problem. The LWR problem is based on the equation of $(a, b = \lfloor \frac{p}{q}\langle a, s\rangle \rceil_p) \in \mathbb{Z}_q^N \times \mathbb{Z}_p$ (where both p and q are power-of-two moduli), and the MLWR scheme is the module version of the LWR.

Saber is an MLWR scheme based PQC, which achieves both classical and quantum security [13]. Saber is first constructed as a Chosen Plaintext Attack (CPA) secure public-key encryption scheme and then developed into KEM Saber through the Fujisaki-Okamoto transformation [29].

Similar to other PQC schemes, the Saber public-key encryption scheme consists of three operational phases, i.e., the key generation, the encryption, and the decryption phases [13], [14]. In the key generation phase, the public matrix of polynomials A and a secret vector of polynomials s are used to produce the scaling and rounding output of As (also the vector b), where the public key is composed of A and b and the secret key is the vector s. In the encryption phase, the original message is encrypted through $v' = sb$ (generated new secret s') and the final produced ciphertext involves the vector b' (from rounding As'). The decryption phase uses the secret key to obtain v, which is approximately the same as v' in the encryption phase (allows the recovering of the original message from the ciphertext). KEM Saber uses the Fujisaki-Okamoto transformation [29] to further ensure its CCA-secure.

Parameter Setting [13]. The polynomial multiplication involved within KEM Saber is set as degree of $N = 256$ and the two moduli are $q = 2^{13}$ and $p = 2^{10}$, respectively. The related secrets are sampled from the binomial distribution.

B. Polynomial Multiplication for KEM Saber

The polynomial multiplication (degree of 256) is the key arithmetic operation in the above mentioned phases. One polynomial involves coefficients generated from the binomial sampler, and these coefficients lie in the value range of -4 to +4 [23], while another polynomial operand consists of coefficients of either 10-bit or 13-bit (the 13-bit based design can also be used for the 10-bit based computation). The design

of [23] has used the schoolbook algorithm to derive the desired high-performance structures. But the proposed polynomial multiplication structures have not fully optimized the output delivery, and hence further efforts are needed in this area.

III. CROP: Mathematical Formulation

Mathematical Formulation (Polynomial Multiplication) Without loss of generality, one can generalize the polynomial multiplication over ring $\mathbb{Z}_l/(x^N+1)$ (l is either $q=2^{13}$ or $p=2^{10}$ [23]) as follows.

Definition 1. Define polynomials as: $W=\sum_{i=0}^{N-1} w_i x^i$, $D=\sum_{i=0}^{N-1} d_i x^i$, and $G=\sum_{i=0}^{N-1} g_i x^i$, where g_i, d_i, and w_i are 4-bit, 13-bit, and 13-bit coefficient over ring, respectively. Define also W is the product of D and G, we can have

$$W = DG \bmod (f(x)=x^N+1). \tag{1}$$

Then, (1) can be rewritten as

$$\begin{aligned} W =& d_0(Gx^i \bmod f(x)) + \cdots \\ &+ d_{N-1}(Gx^i \bmod f(x)), \end{aligned} \tag{2}$$

where $x^N \equiv -1$ can be used and then we have

$$\begin{aligned} W = \sum_{i=0}^{N-1} w_i x^i &= d_0(g_0 + g_1 x + \cdots + g_{N-1}x^{N-1}), \\ &+ d_1(-g_{N-1} + g_0 x + \cdots + g_{N-2}x^{N-1}) + \cdots, \\ &+ d_{N-1}(-g_1 - g_2 x - \cdots + g_0 x^{N-1}). \end{aligned} \tag{3}$$

Example. To illustrate in detail the proposed design strategy, we have also given an example of $N=4$ (from (3))

$$\begin{aligned} w_0 &= g_0d_0 + (-g_3d_1) + (-g_2d_2) + (-g_1d_3), \\ w_1 &= g_1d_0 + g_0d_1 + (-g_3d_2) + (-g_2d_3), \\ w_2 &= g_2d_0 + g_1d_1 + g_0d_2 + (-g_3d_3), \\ w_3 &= g_3d_0 + g_2d_1 + g_1d_2 + g_0d_3, \end{aligned} \tag{4}$$

which can be extended to obtain all w_i ($0 \le i \le N-1$) through a circularly accumulated format, following the strategy in [28]. We have used a signal flow graph (SFG) to realize the proposed strategy for (3) in Fig. 2 (not including the signs, which can be easily handled in hardware design in Section IV).

In Fig. 2, we have put all N number of coefficients as one input to the multiplication, respectively, while another input of the multiplication is fed with the coefficients of the polynomial D in a serial format as d_0,d_1,$\cdots$,d_{N-1}. The multiplication results are then accumulated in a cyclic format to produce the N outputs w_i, i.e., the multiplication results produced from these n multipliers within related n cycles are circularly accumulated to produce the desired output. For example, in the first cycle, the multiplication (the far right one) produces g_0d_0 and then stored in the following accumulator (far left); then, in the second cycle, g_0d_0 will add the multiplication result $g_{N-1}d_1$ produced from the first multiplier (far left); and then adds with $g_{N-2}d_2$ produced from the second multiplier (from left) in the third cycle; and keeps going on until it adds the multiplication result ($d_{N-1}g_1$) of the second far right one after N cycles to produce w_0 (accumulated in the far right one),

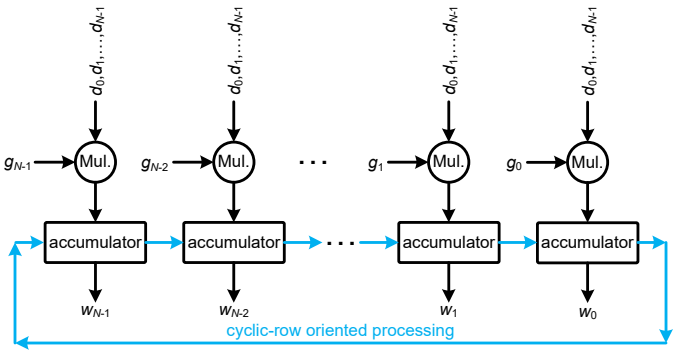

Fig. 2: The proposed SFG to realize (3) (signs are not included), where Mul. is the multiplication and the related multiplication results are accumulated in a cyclic format to produce the N outputs. The multiplication result is added with the value stored in each accumulator and then yielded out to the nearby accumulator to be stored during each cycle.

TABLE I: Example for the Proposed CROP Strategy ($N=4$)

C*	ACC-0	ACC-1	ACC-2	ACC-3
accumulation process (cyclic format, see Fig. 2)				
1	g_1d_0	g_2d_0	g_3d_0	g_0d_0
2	$g_2d_0+g_1d_1$	$g_3d_0+g_2d_1$	$g_0d_0-g_3d_1$	$g_1d_0+g_0d_1$
3	$g_3d_0+g_2d_1$ g_1d_2	$g_0d_0-g_3d_1$ $-g_2d_2$	$g_1d_0+g_0d_1$ $-g_3d_2$	$g_2d_0+g_1d_1$ g_0d_2
4	$g_0d_0-g_3d_1$ $-g_2d_2-g_1d_3$	$g_1d_0+g_0d_1$ $-g_3d_2-g_2d_3$	$g_2d_0+g_1d_1$ $g_0d_2-g_3d_3$	$g_3d_0+g_2d_1$ $g_1d_2+g_0d_3$
out	w_0	w_1	w_2	w_3

C*: cycle number. ACC: accumulator.
Each multiplication value is added with the value stored in the accumulator first and then transferred to the following (neighboring) accumulator to be stored during each processing cycle.

which matches the example procedure shown in (3). The same process applies to all the other output coefficients. Since this type of accumulation is carried out in a cyclic-row format and we just defined this kind of strategy as **C**yclic-**R**ow **O**riented **P**rocessing (CROP). Table I shows the involved procedure following the same operation of Fig. 2, based on (3).

Thus, following the CROP strategy and (4), we can have

$$w_0 = \sum_{j=0}^{N-1} G_j^{(0)} d_j, \quad \cdots, \quad w_{N-1} = \sum_{j=0}^{N-1} G_j^{(N-1)} d_j, \tag{5}$$

where $G^{(i)}$ represents the corresponding operand for w_i and $G_j^{(i)}$ denotes the related $(j-1)$th coefficient, e.g., corresponds with (4), $G^{(0)} = g_0 - g_3x - g_2x^2 - g_1x^3$ and $G_1^{(0)} = -g_3$.

We can thus obtain the proposed CROP based algorithm as

Algorithm 1 Proposed CROP based polynomial multiplication algorithm for KEM Saber (basic version)

Inputs: polynomials G and D (G and D are polynomials with 4-bit and 13-bit integer coefficients over ring, respectively).
Outputs: $W = GD \bmod f(x)$ ($f(x)=x^N+1$).

1. Initialization step

1.1. $\overline{W}=0$.

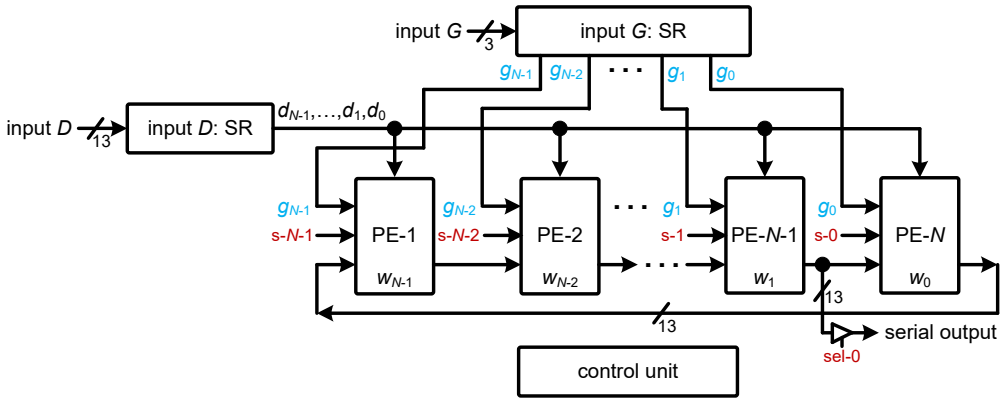

Fig. 3: The proposed polynomial multiplication structure based on Algorithm 1, where SR and PE refer to the shift-register and processing element, respectively. Note that the highlighted blue coefficients of G (initial positions) do not include the signs.

1.2. the coefficients of G and D are in the proper position.
2. Main step
2.1. from $i = 0$ to $N - 1$.
  2.2. for $j = 0$ to $N - 1$.
  2.3. $\overline{W} = \overline{W} + G_j^{(i)} d_j$. // following CROP strategy
  2.4. $w_i = \overline{W}$. // parallel obtain w_i
  2.5. end for.
2.6. end for.
3. Final step
3.1. serially deliver the output coefficients of W.

Note that these N number of w_i are processed in a parallel manner, while Step 2.3 is accumulated through N cycles' operations following the CROP strategy.

Advanced Higher-Speed Version. For a higher-speed operation, we can have

$$w_i = \sum_{k=0}^{t-1} \sum_{h=0}^{r-1} G_{kr+h}^{(i)} d_{kr+h}, \tag{6}$$

where we have defined $N = rt$ (both r and t are integers) such that the original N cycles of accumulation can be decomposed into t number of parallel sub-accumulations (where each sub-accumulation needs r cycles' operation).

We can then have the proposed higher-speed algorithm as

Algorithm 2 Proposed CROP based polynomial multiplication algorithm for KEM Saber (higher-speed version)

Inputs: polynomials G and D (G and D are polynomials with 4-bit and 13-bit integer coefficients over ring, respectively).
Outputs: $W = GD \bmod f(x)$ ($f(x) = x^N + 1$).
1. Initialization step
1.1. $\overline{W} = 0$.
1.2. the coefficients of G and D are in the proper position.
2. Main step
2.1. from $i = 0$ to $N - 1$.
  2.2. for $k = 0$ to $t - 1$.
    2.3. for $h = 0$ to $r - 1$.
    2.4. $\overline{W} = \overline{W} + G_{kr+h}^{(i)} d_{kr+h}$.//following CROP strategy
    2.5. end for.
  2.6. end for.
2.7. $w_i = \overline{W}$. // parallel obtain w_i
2.8. end for.
3. Final step
3.1. serially deliver the output coefficients of W.

---

where Step 1.2 involves the preparation of decomposing the coefficients of polynomial D into t groups and each group has r coefficients (similar to all the $G_j^{[i]}$).

Remark. The main benefits of the proposed CROP strategy include: (i) the output results are obtained through cyclic accumulation format, which facilitates the final outputs' storing into external memory (little resource is required on this operation); (ii) the overall signal processing is rather simple, i.e., one polynomial's coefficients are serially fed in while another polynomial's coefficients are attached to the corresponding multiplier, respectively. The following structural design will reveal more details on these two advantages.

IV. Proposed CROP based Hardware Structures

This section presents the proposed CROP strategy originated hardware structures. Note that the coefficients of G are represented in the sign magnitude format, following [21], [23].

Basic Version. The proposed polynomial multiplication structure-I (basic version) based on Algorithm 1 is shown in Fig. 3. The proposed structure mainly consists of three components, namely the input component, the processing element (PE) array/component, and the control unit.

The Input Component. As shown in Fig. 3, there are two shift-registers (SRs) involved, for polynomials D and G, respectively. As the coefficients of G are represented in the sign magnitude format, we just use 3-bit registers in the SR (the most significant bit (MSB) is the corresponding sign bit, which is generated by another individual circular shift-register (CSR) in the control unit). After N cycles, all the coefficients are loaded in the corresponding registers and can be delivered out to the following component. The SR for input G (serial-in parallel-out style) directly delivers all the N coefficients in parallel to the corresponding PEs, respectively. While the SR

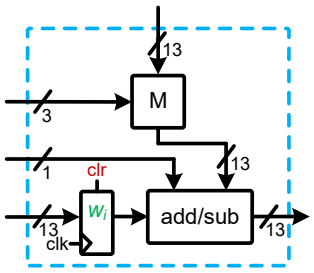

Fig. 4: The internal structure of the PE, where the highlighted green w_i $(0 \leq i \leq N-1)$ is the final value accumulated.

for input D (serial-in serial-out) delivers the coefficients of D in a serial format such that all the PEs receive one output per cycle, i.e., starting from d_0 and finally to d_{N-1}.

The PE Array/Component. The PE array/component consists of N PEs, where all the PEs have the same internal structure as that shown in Fig. 4. The PE contains one M cell, one register, and one add/sub cell. The details of the major cells in the PE are shown in Fig. 5. The M cell executes the multiplication operation for one coefficient of G with the corresponding coefficient from D. As the absolute magnitude of the coefficients of G lie in range of 0 to 4, we can just follow the design style of [23] to design a MUX-based multiplication cell, where all the possible multiplication results (0, d_i, $2d_i$, $3d_i$, and $4d_i$) are pre-obtained to be attached to the MUX (the value of $3d_i$ comes from the addition of d_i and $2d_i$, while the $2d_i$ and $4d_i$ are obtained through bit-shifting). The output of the MUX is then fed to the add/sub cell to be added/subtracted with the input from the left. Note that the sign bit (s-i, generated from the control unit, matching the corresponding sign for each g_i according to Algorithm 1) determines the operation within the add/sub cell either in the addition or subtraction status. After N cycles of operations, the desired output w_i will be available in the related registers. Note that there is a tri-state gate attached to the input of PE-N (w_0) such that all the N output coefficients are serially delivered out in the sequence of $w_0, w_1, \cdots, w_{N-1}$ (of course, all the other inputs to the PEs are set as '0' in this process). In this way, all the actual results are delivered out after $(N-1)$ cycles, which is one less than the existing one of [23].

The Control Unit. The control unit is responsible for generating all the control signals necessary for the operation of the structure, mainly in three stages. The operation in the first stage mainly refers to the loading of coefficients in the two SRs. While the major operation in the second stage is generating of sign bits for the corresponding g_i according to Algorithm 1: (i) the first cycle, all the sign bits are"$00\cdots00$" (N number of '0'); (ii) the sign bits turns into "$10\cdots00$" (the sign '1' is for g_{N-1} according to (3)); (iii) each cycle there will be one more '1'appearing in the sign bits (the third cycle the sign bits are "$11\cdots00$", only two '1') until the final cycle (i.e., "$11\cdots10$", only one '0'). The last stage's operation mainly delivers the output coefficient w_i in a serial format (the other inputs to the PE are set as '0'). We have thus used a CSR to realize this sign generation for all the corresponding coefficients, as shown in Fig. 6. Within

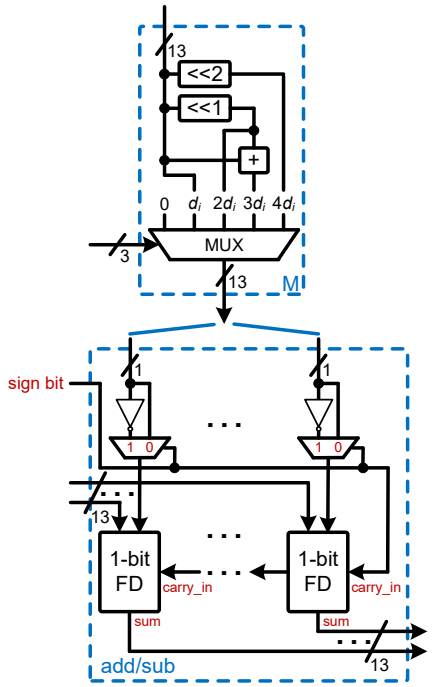

Fig. 5: The the major cells (FD: full adder) in each PE.

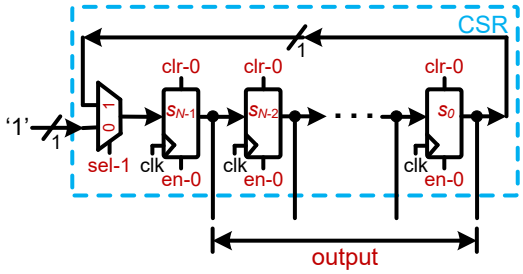

Fig. 6: The CSR for sign bits generation.

this CSR, the registers are firstly cleared up as '0', and then the MUX and enable signals for all the registers coordinate together to load the value of '1' into these registers during the actual computation process (the input for the CSR is set as '1'), thus producing the desired sign bits for all the corresponding coefficients of G during the processing stage.

Besides that, the control unit is also responsible for delivering out of desired results to the external resources (such as memory). As shown in Fig. 3, after all the desired output coefficients are accumulated in the registers of all the PEs, we can directly set the control signal to the tri-state buffer as (sel-0='0'), thus enabling all the output coefficients to be serially delivered out in the next N cycles (meanwhile, all the related input signals to the PEs are set as zero). Therefore, the whole computation process is smooth and efficient, and no specific extra resources are required for output delivery.

Advanced Higher-Speed Version. For a higher-speed operation, we can have the proposed structure-II of Fig. 7 (where we have used the example of $t = 2$).

The structure of Fig. 7 involves almost the same details as those in Fig. 3 except that each PE has doubled inputs from inputs D and G, respectively. Correspondingly, the SR (input D) now delivers two outputs at the same time, i.e., the

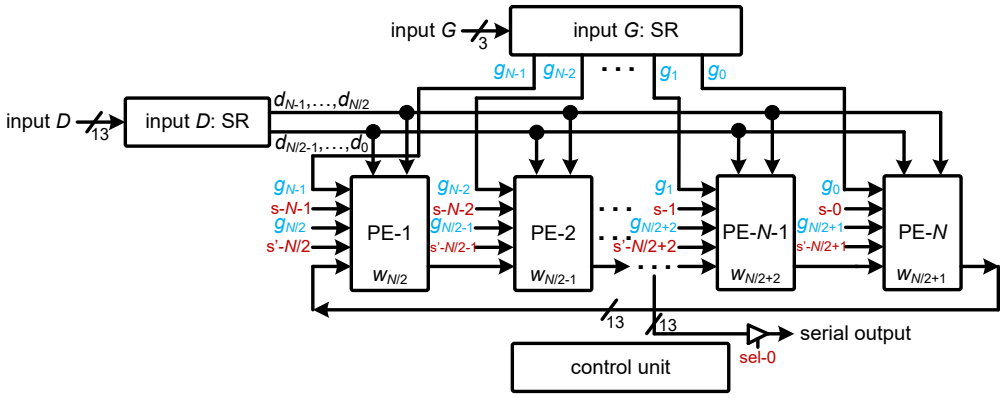

Fig. 7: The proposed polynomial multiplication structure-II based on Algorithm 2, where we have used $t = 2$.

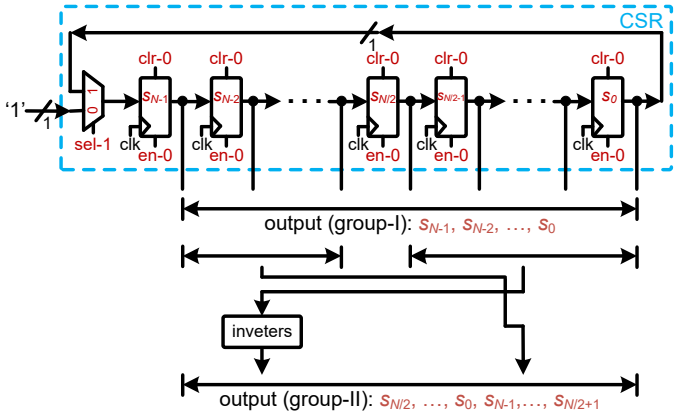

Fig. 8: The details of the internal structure for the producing of two groups of signs to be fed to the PEs.

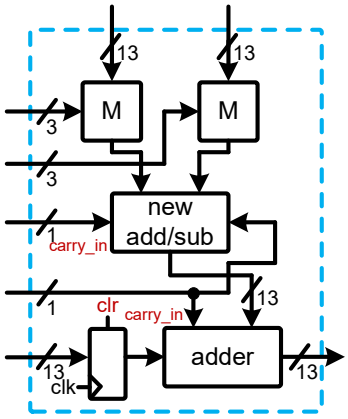

Fig. 9: The details of the internal structure of the PE.

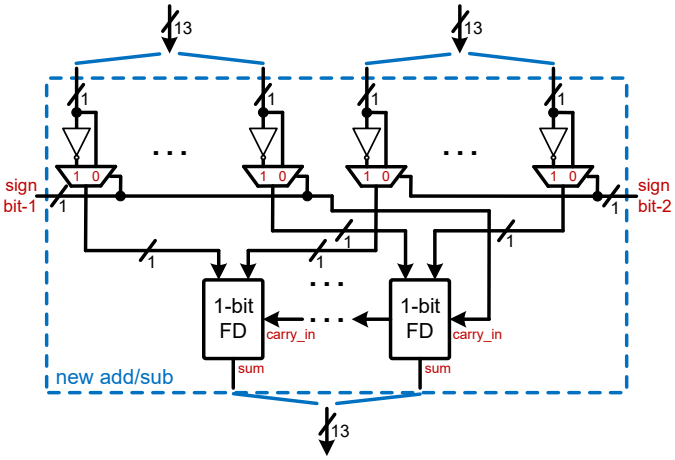

Fig. 10: The details of the new add/sub cell within the PE.

whole N coefficients of D has been divided into two groups as shown in Fig. 7 (nothing is needed on the changing of structural side except that the output of the $N/2$th register is also delivered to the PEs). Meanwhile, another group of $g_{N/2}, g_{N/2-1}, \cdots, g_0, g_{N-1}, \cdots, g_{N/2+1}$, formed from the output of the SR (input G), is also fed to the corresponding PEs according to (6) and Algorithm 2 to match the second group of outputs from the SR of input D. Meanwhile, there are also two groups of sign bits (corresponding to the two groups of coefficients of G) attached to all the N PEs, respectively. Fig. 8 shows the internal structure of the CSR producing the necessary two groups of signs, where $N/2$ numbers of inverters are used to generate the second group of $\{s_{N/2}, \cdots, s_0, s_{N-1}, \cdots, s_{N/2+1}\}$ (the rest part of the operations are the same as the original one in Fig. 6).

The internal structure of the PE is shown in Fig. 9, where there are two M cells, one new add/sub cell, one register, and one adder (regular full adder). The internal structures of the two M cells are the same as those in Fig. 5, which mainly execute two multiplications for the paired inputs from D and G, respectively. But as the coefficients from G are represented in the sign magnitude format, we need to use the new add/sub cell for the following addition operation, as shown in Fig. 9, with the coordination of the related two sign bits. The internal structure of the new add/sub cell is then shown in Fig. 10, where the two sign bits from the two coefficients of G are determining the actual operation of the adders in the status of addition or subtraction. Meanwhile, one of the sign bit is attached to the carry-in of the new add/sub cell to meet the requirement of transferring into two's complement number system (while another sign bit is also attached as the carry-in to the adder in the PE, as shown in Fig. 9). The results for output coefficients are accumulated in the registers in the PEs. But as the structure of Fig. 7 now executes two multiplications at one time, the positions of the final output coefficients are also changed, as shown in Fig. 7, which only requires $N/2$ cycles' circular accumulation and processing (the final output

TABLE II: General Complexities for Various Polynomial Multiplication Structures for Saber

design	#Mul.	#Add.	#register	latency	output style
Basic High-Performance Version					
Fig. 2 [23]	N	N	N	$2N$	parallel*
Fig. 3	N	N	N	$2N-1$	serial
Higher-Speed Version					
Fig. 2[1] [23]	N	N	N	$3N/2$	parallel*
Fig. 7	$2N$	$2N$	N	$3N/2-1$	serial

Mul. and Add. represent the multiplication and addition, respectively.
The number of additions include the add/sub cells.
Latency refers to the main computational and output delivering cycles.
*: Extra resources (like MUXes) are needed to transfer the parallel output into serial format to be stored in the external memory for further usage.
[1]: The extended higher-speed version based on the structure of Fig. 2 in [23] to achieve the computation latency of $N/2$ cycles.

is attached to the input of the register storing w_0, and the whole output delivering time is again $(N-1)$ cycles).

The internal structure of the control unit remains the same as that in Fig. 3 except that the number of cycles for accumulation now is reduced to $N/2$. Meanwhile, the position of the final output attached to the circular loop in the structure also changes, as shown in Fig. 7.

The Extension to Other Values of t. The structure presented in 7 can also be extended to other values of t for higher-speed operation, only following the principle that the number of grouped sign bits are paired with the inputs to each PE (similar to the changing of the number of new add/sub cells in each PE as well as the M cells), while the majority parts of the structure remain the same as that in Fig. 7. Besides that, there is a need for a slight update on the control unit as the main accumulation time now takes N/t cycles (the position to deliver out the final output is also correspondingly switched following the accumulation sequence of Algorithm 2).

V. Complexity Analysis and Comparison

Complexity Analysis. For simplicity of discussion, we just depict the area-time complexities of the proposed structures in Figs. 3 and 7. The proposed polynomial multiplication structure of Fig. 3 (basic version) involves two SRs, where each SR contains N number of registers with specified bit-width of 3 and 13, respectively. Meanwhile, there are in total N PEs involved within the structure, and each PE has one M cell, one register, and one add/sub cell. A control unit is needed to generate all the necessary signals as well as an extra tri-state gate to serial deliver out the final N output coefficients. The overall latency of the computation is N cycles (output delivering is $(N-1)$ cycles, and the input loading for the two SRs is also N cycles). Similar complexity estimation strategy applies to the structure of Fig. 7.

Theoretical Analysis & Comparison. When comparing with the recently released ones in [23] (since in the most recent design of [23], the authors have shown their designs outperform the other available structures such as [21] in the literature, we here just compare mainly with the ones of [23]), as shown in Table II. The proposed ones have one unique advantage, i.e., the delivery of output coefficients is very simple and almost involves no extra resource usage since all the output values are computed and accumulated in a circular format while the existing designs of [23] require extra resources (such as MUXes) to transfer the parallel outputs into a serial format to be stored into the related memory section for further processing. Besides that, this unique feature may also bring a little bit of savings in overall area-time complexities since the proposed structures also have one less output delivering cycle than the existing designs of [23].

FPGA based Implementation & Comparison. To have a further detailed fair comparison, we have coded the proposed designs (Figs. 3 and 7) along with the newly released ones in [23] and have also obtained their corresponding performance after implementation. Again, we want to mention that the authors of [23] have shown their designs outperform the other structures such as [21] available in the literature, and we hence just put the high-performance ones (the similar latency cycles) of [23] as our competing designs.

Experimental Setup. The overall experimental settings are:

(i) We have used VHDL to code the proposed polynomial multiplication structure of Fig. 3 and Fig. 7. The FPGA based implementation results are obtained through the Intel Quartus Prime 17.0 on the Stratix V 5SGXMABN1F45C2 and Cyclone V 5CSXFC6D6F31I7ES devices, respectively.

(ii) We have followed the same parameter setting in [21]–[23], i.e., one polynomial has 13-bit coefficients while another has 3-bit coefficients (not including the sign bit) ($N = 256$). We have also used the same type of adder for both the proposed and existing designs.

(iii) We have followed the structural introduction in [23] (Fig. 2 of [23]) to complete the coding (basic version, with function verified through ModelSim) and have also coded the extended higher-speed version (latency is $N/2$ =128). A corresponding control unit is also coded to coordinate the input loading, main computation, and the final output delivery. Note that similar loading shift-registers are also used for [23].

(iv) The area-time complexities/performance of the proposed and the competing designs, in terms of the number of adaptive logic module (ALM), maximum frequency (Fmax, with MHz as the unit), latency cycles (main computation and output delivering cycles), delay ((1/Fmax)×(latency cycles)), and area-delay product (ADP$\times 10^3$), are listed in Table III.

As shown in Table III, the proposed designs overall outperform the existing designs of [23] (newly available in the literature), e.g., the proposed design (basic version) has at least 11.2% less ADP than the competing one on the Cyclone V device while the proposed higher-speed structure of Fig. 7 (case of $t = 2$) involves at least 10.4% less ADP than the state-of-the-art one of [23] on the same Cyclone V device (of course, the proposed designs also have better performance than the competing design on the Stratix V device). This should be due to the fact that the proposed designs have a simple setup on the output delivering (benefited from the proposed CROP strategy) while the existing ones need extra resources for transferring the output signals into a serial format for further processing, which (i) improves the maximum frequency (also with one less

TABLE III: Comparison of the Area-Time Complexities for the Proposed and the Competing designs on the FPGA

design	ALMs	Fmax	latency	delay	ADP*	ER?
Basic Version (Stratix V device)						
Fig. 2 [23]	6,889	257.33	512	1,990	13,709	Y
Fig. 3	6,921	267.38	511	1,911	13,226	N
Basic Version (Cyclone V device)						
Fig. 2 [23]	7,084	124.52	512	4,112	29,129	Y
Fig. 3	6,915	136.63	511	3,740	25,862	N
Higher-Speed Version (Stratix V device)						
Fig. 2^1 [23]	14,689	196.43	384	1,955	28,717	Y
Fig. 7	14,341	204.54	383	1,872	26,846	N
Higher-Speed Version (Cyclone V device)						
Fig. 2^1 [23]	15,426	98.48	384	3,899	60,146	Y
Fig. 7	14,579	103.62	383	3,696	53,884	N

The design of [23] has shown its efficiency over the other ones such as [21], we hence just use the high-performance ones in [23] for comparison.
ER?: extra resources (for output delivering)? Y: Yes; N: No;
Unit for Fmax: MHz. Unit for delay: ns. *: ADP=#ALM$\times$delay ($\times 10^3$).
Latency refer to the main computation and output delivering cycles.
1: The extended higher-performance version based on the structure of Fig. 2 in [23] to achieve the computation latency of 128 cycles (total 384 cycles).

latency cycle); (ii) potentially enhances the actual mapping efficiency, of the proposed designs, on the FPGA devices.

Discussion. It is noted that we have not employed any manual area optimization techniques to obtain optimal performance on the FPGA platform for the sake of fair comparison.

Besides that, we want to mention that though we have followed the design style of [28] to implement the polynomial multiplication for KEM Saber, the actually implemented structures involve many different aspects from the former one, including the targeted PQC scheme, parameter setting, input/output setup, structural design, and even the control signal setup. This is similar to the very recent reports of [21], [23], where the schoolbook polynomial multiplication is employed to obtain the final hardware structure (the design style follows the conventional algorithm, but the implemented structures are novel to KEM Saber). Overall, as the proposed designs offer convenient output delivery, flexible speed rate, and low-complexity, they are more suitable for high-performance KEM Saber applications. Future research can focus more on the employing of the polynomial multiplication in the actual Saber implementation with respect to different security levels.

VI. Conclusion

This paper proposes a novel CROP strategy for efficient implementation of the polynomial multiplication of KEM Saber. Firstly, the main operation (polynomial multiplication) of KEM Saber is transferred into a form suitable for the derivation of the proposed CROP based algorithms. Then, the proposed algorithms are innovatively mapped into the related hardware structures. Finally, complexity and comparison have confirmed the efficiency of the proposed designs over the competing structures. The outcome of this work is expected to be useful references for NIST PQC standardization process.

Acknowledgement

Jiafeng Xie's work is supported by NSF SaTC-2020625 and NIST 60NANB20D203.

References

[1] Post-Quantum Cryptography. https://en.wikipedia.org/wiki/Post-quantum_cryptography.
[2] W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. *Symp. Founda. of Computer Science*, pp. 124-134, 1994.
[3] J. Xie et al., "Special Session: The recent advance of hardware implementation of post-quantum cryptography," *IEEE VTS*, pp. 1-10, 2020.
[4] D. Micciancio. Lattice-based cryptography. *Encyclopedia of Cryptography & Security*, 2011.
[5] Post-quantum cryptography round 3 submissions. https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions
[6] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *Journal of the ACM*, vol. 56, no. 6, 34, 2009.
[7] V. Lyubashevsky et al., "On ideal lattices and learning with errors over rings," *Int. Conf. Theory & Appl. of Crypto. Tech.*, pp. 1-23, 2010.
[8] T. Pöppelmann et al., "Area optimization of lightweight lattice-based encryption on reconfigurable hardware," *ISCAS*, 2014, pp. 2796-2799.
[9] S.S. Roy et al., "Compact Ring-LWE cryptoprocessor," *CHES*, pp. 371-391, 2014.
[10] W. Liu et al., "Optimized schoolbook polynomial multiplication for compact lattice-based cryptography on FPGA," *IEEE TVLSI Syst.*, vol. 27, no. 10, pp. 2459-2463, 2019.
[11] A. Aysu et al., "Binary Ring-LWE hardware with power side-channel countermeasures," *DATE*, pp. 1253-1258, 2018.
[12] A. Banerjee, C. Peikert, and A. Rosen, "Pseudorandom Functions and Lattices," *In EUROCRYPT 2012*, pp. 719-737, 2012.
[13] J.-P. D'Anvers et al., "SABER. Proposal to NIST PQC Standardization,"Round2, 2019. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/ round-2-submissions.
[14] J.-P. D'Anvers, A. Karmakar, S. S. Roy, and F. Vercauteren, "Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM," vol. 10831, pp. 282-305, 2018.
[15] A. Karmakar et al., "Saber on ARM CCA-secure module lattice-based key encapsulation on ARM," *IACR Cryptology ePrint*, 2018:682, 2018.
[16] J. Mera et al., "Time-memory trade-off in Toom-Cook multiplication: an Application to Module-lattice based cryptography," *IACR TCHES*, vol. 2020, no. 2, pp. 222-244, 2020.
[17] J. Mera et al., "Compact domain-specific co-processor for accelerating module lattice-based KEM," *DAC*, pp. 1-6, 2020.
[18] V. Dang et al., "Implementing and benchmarking three lattice-based post-quantum cryptography algorithms using software/hardware codesign," *FPT 2019*, pp. 206-214, 2019.
[19] J. Pollard, "The fast Fourier transform in a finite field," *Mathematics of computation*, vol. 25, no. 114, pp. 365-374, 1971.
[20] T. Fritzmann et al., "RISQ-V: Tightly coupled RISC-V accelerators for post-quantum cryptography," *Cryptology ePrint*, Report 2020/446.
[21] S. Roy and A. Basso, "High-speed instruction-set coprocessor for lattice-based key encapsulation mechanism: Saber in hardware," *IACR TCHES*, vol. 2020, no. 4, pp. 443-466, 2020.
[22] Y. Zhu et al., "A high performance hardware implementation of Saber based on Karatsuba algorithm," *Cryptology ePrint*, Report 2020/1037.
[23] A. Basso and S. Sinha Roy, "Optimized polynomial multiplier architectures for post-quantum KEM Saber," *DAC'21*, pp.1-6, 2021.
[24] P. Meher and X. Lou, "Low-latency, low-area, and scalable systolic-like modular multipliers for $GF(2^m)$ based on irreducible all-one polynomials," *IEEE TCAS-I* vol. 64, no. 2, pp. 399-408, 2017.
[25] J. L. Imaña, "LFSR-based bit-serial $GF(2^m)$ multipliers using irreducible trinomials" *IEEE Trans. Computers*, 2020 (early access).
[26] P. Meher, "Systolic and super-systolic multipliers for finite field $GF(2^m)$ based on irreducible trinomials," *IEEE TCAS-I*, vol. 55, no. 4, pp. 1031-1040, 2008.
[27] S. Namin et al., "Low-power design for a digit-serial polynomial basis finite field multiplier using factoring technique," *IEEE TVLSI*, vol. 25, no. 2, pp. 441-449, 2017.
[28] J. Xie et al., "Efficient hardware implementation of finite field arithmetic $AB+C$ over hybrid fields for post-quantum cryptography," *IEEE Trans. Emerging Topics In Computing*, pp. 1-6, 2021. (accepted)
[29] D. Hofheinz et al., "A modular analysis of the Fujisaki-Okamoto transformation," *15th International Conference Theory of Cryptography Proceedings Part I*, vol. 10677, pp. 341-371. 2017.