Compact Coprocessor for KEM Saber: Novel Scalable Matrix Originated Processing

Pengzhou He[1], Chiou-Yng Lee[2], Jiafeng Xie[1] (corresponding author)
[1]: Department of Electrical & Computer Engineering, Villanova University, Villanova, PA (jiafeng.xie@villanova.edu);
[2]: Department of Computer Information & Network Engineering, Lunghwa University of Science & Technology, (pp010@gm.lhu.edu.tw);

***Abstract*—Recent advance in the quantum computing field has initiated a new round of cryptosystem innovation as the existing public-key cryptosystems are proven to be vulnerable against the attacks launched from a mature quantum computer. Along with this innovation, several types of cryptographic algorithms have been proposed for possible post-quantum cryptography (PQC) candidates, where the lattice-based key encapsulation mechanism (KEM) Saber is one of the promising cryptosystems. Noticing that the recent trend in the field has switched more on the efficient implementation of PQC algorithms, in this paper, we propose to present a novel compact coprocessor for KEM Saber on the field-programmable gate array (FPGA) platform. Particularly, the proposed strategy aims to obtain a generic method suits for different security levels of Saber with flexible processing styles yet with low-complexity.**

In total, we have carried out four layers of major innovations to finalize the proposed work: (i) we have formulated and derived a Scalable Matrix Originated Processing (SMOP) strategy for the main computation-intensive operation, i.e., polynomial multiplication, of the mentioned KEM Saber in a general format; (ii) we have then presented the details of the SMOP strategy based polynomial multiplication algorithm including the algorithmic operation and structural/implementational innovation with respect to the Saber PQC scheme; (iii) we have also followed the existing coprocessor design process to build a new compact coprocessor for KEM Saber deploying the proposed polynomial multiplication architecture based on novel algorithm-architecture co-implementation techniques, along with other important building blocks; (iv) we have also finally given thorough complexity analysis and comparison to demonstrate the superior efficiency of the proposed compact coprocessor (including the polynomial multiplication architecture) over the state-of-the-art solutions. The proposed design strategy and the coprocessor are highly efficient: (a) ultra low-complexity; (b) flexible processing rate; (c) structural generic; and (d) suitable for practical lightweight application environments. To the authors' best knowledge, this is the FIRST paper on compact based implementation for the KEM Saber in a generic format. The outcome of this work is expected to be useful reference for further development and related standardization for KEM Saber.

***Index Terms*—Field-programmable gate array (FPGA), flexible processing, hardware implementation, instruction-set coprocessor, lattice-based cryptography, low-complexity, module-learning-with-rounding (MLWR) scheme, polynomial multiplication, post-quantum cryptography (PQC), scalable matrix originated processing**

I. Introduction

Along with the rapid progressing in quantum computing technology, more and more attentions have switched on developing next-generation cryptosystem as the current public-key cryptosystems are proved to be vulnerable toward the attacks launched from powerful quantum computers executing Shor's algorithm [1], [2] (which is expected to be happened in the next 15 to 20 years). Post-quantum cryptography (PQC) refers to the cryptographic algorithm that is resistant against quantum attacks, and several PQC algorithms have been proposed in the past years. Among these possible PQC candidates, the lattice-based cryptography is regarded as one of the most promising class due to its high security, efficiency, and simplicity for implementation [3]. Moreover, the importance of the lattice-based PQC is also confirmed by the very recently released National Institute of Standards and Technology (NIST) third round of PQC standardization process, where three out of four encryption scheme finalists are lattice-based schemes [4], [5].

The lattice-based PQC can be divided into two categories, one category is built on the N-th degree truncated polynomial ring (NTRU)-based scheme while the other is based on the learning-with-errors (LWE)-based scheme [2], [3], [6], [7]. As the latter category, many research works have developed a number of variants for possible complexity reduction and practical application, including the learning-with-rounding (LWR)-based scheme [8], [9] (as well as its variant Module-LWR (MLWR)-based scheme). The LWR is a variant of the LWE problem, where the error term is introduced by a rounding operation rather than obtaining it from a random distribution. The MLWR is based on the LWR problem with the module matrices, and quite a number of works have been done related to this important problem. Moreover, in the NIST's third round PQC standardization process, one of the lattice-based finalists, Saber [9], [10], is also based on the MLWR problem, which uses power-of-two moduli to achieve flexibility, simplicity, high security, and efficiency.

Existing Works. The authors of Saber firstly proposed to use the Toom-Cook method for the polynomial multiplication of the Saber to achieve better implementation [10]. This strategy is then optimized further in [11] and [12], respectively, for more efficient implementations. These designs are software based implementations and hence can be improved through hardware implementations to obtain better performance. Especially that the recent trend in the PQC field has gradually switched to the efficient implementation of the cryptographic algorithms on the hardware platforms [2], [5]. But overall, the reported hardware designs for the mentioned KEM Saber are still very limited. On the other side, there are mainly two types

of hardware implementation strategies, namely the hardware-software co-design and the full hardware design. The former can be seen in a recent paper at [13], where the authors use a Toom-Cook method to reduce the computational complexity of the involved polynomial multiplication for the efficient implementation of Saber on the FPGA device. Another such design is newly released in [14], where the design achieves better performance but at the cost of large resource usage. Very recent work uses the number theoretic transform (NTT) based method to implement the polynomial multiplication for Saber through field extension and the Chinese Remainder Theorem, but the main computation module is executed on a RISC-V accelerators [15]. For the latter type, i.e., the full hardware implementations, the first report is released in [16], where the main arithmetic operation is based on a school-book based method to achieve high speed and moderate area consumption. Recently, a parallel 8-level Karatsuba algorithm based polynomial multiplier is proposed to be deployed to obtain a low-power KEM Saber [17]. But due to the iterative nature, the proposed hardware structure has a large area and long critical-path. The authors of [18] very recently present area/performance trade-offs of polynomial multiplication for Saber on hardware platforms, targeting both lightweight and high-speed implementations. These mentioned works represent the main efforts in the field.

It is known that the polynomial multiplication in the ring is the key arithmetic operation for the mentioned KEM Saber, and hence the efficiency of the polynomial multiplication plays a critical role in the overall performance of the cryptosystem. For lattice-based cryptography such as Ring-LWE based scheme, the polynomial multiplication can be carried out by the fast algorithm based method (NTT) [19], based on the specific parameter setting on the modulo ring. While Saber uses power-of-two moduli and the NTT based method is thus not convenient to be used in this case. This type of parameter setup undoubtedly has made the efficient implementation of Saber an interesting and challenging research topic.

Noticing that the above mentioned KEM Saber PQC scheme is also targeted for compact and lightweight applications, its low-complexity implementations on the hardware platform, however, still needs significant improvement since (i) there is so far only one lightweight version of polynomial multiplication available in the literature for the KEM Saber; (ii) the available hardware implementations are based on existing polynomial multiplication algorithms and no further improved algorithmic derivation process is provided [16], [17]; (iii) the reported compact hardware architecture, particularly refers to the corresponding polynomial multiplication for KEM Saber (which is the key arithmetic operation of the scheme), is very limited on throughput rate (ONLY one fixed rate) and no other flexible versions proposed; (iv) there lacks a generic compact implementation of the KEM Saber, which can be fit in different application environments based on the resource availability. All these factors indicate that the efficient implementation of compact KEM Saber cryptoprocessor is highly demanded.

Major Contributions. Based on this consideration, in this paper, we aim at presenting a novel design strategy for efficient implementation of KEM Saber on the hardware platform, i.e., a novel compact coprocessor. In particular, the proposed strategy fits for various types of application environments, which still maintains efficiency in timing-complexity (also the resource usage). The proposed work is based on a novel Scalable Matrix Originated Processing (SMOP) strategy, and we have conducted several layers of coherent interdependent efforts to carry out the overall research.

- We have presented detailed mathematical formulation and derivation process to propose a novel SMOP strategy for the polynomial multiplication of the mentioned KEM Saber in a general format, which lays a solid theoretical foundation for the obtaining of the desired compact coprocessor.
- We have then presented the details of the SMOP strategy based polynomial multiplication algorithm, including the algorithmic operation and structural innovation with respect to the specific PQC encryption scheme.
- We have also followed the existing coprocessor design process to obtain the targeted compact coprocessor, deploying the proposed polynomial multiplication with the help of several algorithm-architecture co-implementation techniques.
- We have given thorough complexity analysis and comparison to confirm the superior performance of the proposed hardware coprocessor under different styles than the state-of-the-art solutions.
- We have also carried out a series of FPGA based implementation and comparison along with the competing designs (available in the literature) to demonstrate the efficiency of the proposed coprocessor over the existing ones.
- We have finally presented a thorough discussion and analysis on the unique features of the proposed strategy and structures as well as their possible extensions and applications.

To the authors' best knowledge, this is the FIRST paper on compact implementation for the KEM Saber in a generic and flexible format.

Overall, the proposed strategy and hardware coprocessor for the KEM Saber offers many unique advantages:

- Low-complexity. The proposed SMOP strategy brings efficiency on the overall computational time-complexity reduction to the coprocessor (including polynomial multiplication) theoretically and structurally and resources usage.
- Flexibility and scalability. The proposed SMOP strategy and designed coprocessor offer great flexibility and scalability on the processing size/rate (also the related area and timing performance), suitable for different application environments.
- Generic and versatility. The proposed strategy and coprocessor fit well for Saber and can be extended to the other

important lattice-based schemes.

Organization of the paper. The rest part of the paper is organized as follows. Section 1 covers the preliminary knowledge related to KEM Saber and the involved polynomial multiplication. Section 2 presents the proposed mathematical process for the general form of the polynomial multiplication and the proposed SMOP strategy. Section 3 gives a detailed coprocessor design process deploying the proposed polynomial multiplication architecture. Section 4 provides the thorough complexity analysis and comparison as well as the FPGA based implementation to show the superior performance of the proposed coprocessor over the competing ones. Finally, the conclusion of this paper is presented in Section 5.

II. Preliminaries

This section mainly gives a brief introduction of the mentioned KEM Saber and the corresponding polynomial multiplication. Note that we follow the parameter notations in the existing literature to present the preliminaries, and the detailed information can be referred to the specific papers [10].

A. Notation

We follow the existing paper [9], [10], [16] to list the following notation, for a better understanding to the potential readers. Define p and q as two powers of 2 as $p = 2^{\varepsilon_p}$ and $q = 2^{\varepsilon_q}$ and let $\mathbb{Z}_q$ be the ring of integers modulo q. We also define the ring of polynomials $\mathcal{R}_p = \mathbb{Z}_q p[x]/\langle x^N + 1\rangle$ for p and $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^N + 1\rangle$ for q, respectively. $\mathbf{a}$ is used to represent a vector and $a(x)$ is used to denote the polynomial in $\mathcal{R}$, where the coefficients can be seen as a vector and the i-th coefficient is the i-th entry of the vector. Moreover, the operator $\lfloor\cdot\rceil$ denotes the rounding operation.

Besides that, β_μ represents the binomial distribution based on parameter μ, which produces values in the range of $[-\mu/2, \mu/2]$ with probability of $\frac{\mu!}{(\mu/2+x)!(\mu/2-x)}2^{-\mu}$. $x \leftarrow \beta_\mu$ denotes that x is randomly sampled from the binomial distribution. If we replace x with X, then it means a polynomial X is sampled from the related binomial distribution. Finally, the notation of $x \leftarrow \mathcal{U}(S)$ denotes that x is uniformly selected from S.

B. The MLWR-based Encryption Scheme: KEM Saber

The LWR scheme is a variant of the LWE problem, where the error term is introduced by a rounding operation rather than obtaining it from a random distribution [10]. The samples for LWR scheme are generated by $(\mathbf{a}, b = \lfloor \frac{p}{q}\langle \mathbf{a}, s\rangle \rceil_p) \in \mathbb{Z}_q^n \times \mathbb{Z}_p$. The MLWR scheme is based on the LWR problem with the module matrices.

Saber is a IND-CCA secure KEM, which is built on the hardness of the MLWR problem to achieve both classical and quantum security [10] since it is proved to be computationally infeasible both on classical and quantum computers. Saber is firstly constructed into a Chosen Plaintext Attack (CPA) secure public-key encryption scheme based on the using power-of-two moduli (both p and q). After that, a CCA-secure Saber KEM is finalized through the Fujisaki-Okamoto transformation [20].

In brief, the Saber public-key encryption scheme consists of three phases, namely the key generation, encryption, and decryption. In the key generation phase (see Algorithm 1), a public matrix of polynomials $\mathbf{A}$ and a secret vector of polynomials $\mathbf{s}$ are generated. Meanwhile, the vector $\mathbf{b}$ is obtained through the scaling and rounding of the product $\mathbf{As}$, where the public key consists of $\mathbf{A}$ and $\mathbf{b}$ and the secret key is the vector $\mathbf{s}$.

Algorithm 1 Saber.PKE.KeyGen() [9], [10]

$seed_{\mathbf{A}} \leftarrow \mathcal{U}(0,1)^{256}$.
$\mathbf{A} = \text{gen}(seed_{\mathbf{A}}) \in \mathcal{R}_q^{l\times l}$.
$r = \mathcal{U}(\{0,1\}^{256})$.
$\mathbf{s} = \beta_\mu(\mathcal{R}_q^{l\times 1}; r)$.
$\mathbf{b} = ((\mathbf{A}^T\mathbf{s} + \mathbf{h}) \bmod q) \gg (\varepsilon_q - \varepsilon_p) \in \mathcal{R}_p^{l\times 1}$.
return $(pk := (seed_{\mathbf{A}}, \mathbf{b}),\ sk := (\mathbf{s}))$.

In the encryption phase, the message is encrypted by ${v_1}' = \mathbf{s}'\mathbf{b}^T$ ($\mathbf{s}'$ is a vector specifically generated for the encryption). The produced ciphertext involves the vector $\mathbf{b}'$ (from rounding $\mathbf{As}'$). The details can seen at Algorithm 2.

Algorithm 2 Saber.PKE.Enc($pk := (seed_{\mathbf{A}}, \mathbf{b}), m \in \mathcal{R}_2; r$) [9], [10]

$\mathbf{A} = \text{gen}(seed_{\mathbf{A}}) \in \mathcal{R}_q^{l\times l}$.
if r is not specified then
  $r = \mathcal{U}(\{0,1\}^{256})$.
$\mathbf{s}' = \beta_\mu(\mathcal{R}_q^{l\times 1}; r)$.
$\mathbf{b}' = ((\mathbf{As}' + \mathbf{h}) \bmod q) \gg (\varepsilon_q - \varepsilon_p) \in \mathcal{R}_p^{l\times 1}$.
${v_1}' = \mathbf{b}^T(\mathbf{s}' \bmod p) \in \mathcal{R}_p$.
$c_m = ({v_1}' + h_1 - 2^{\varepsilon_p - 1}m \bmod p) \gg (\varepsilon_p - \varepsilon_T) \in \mathcal{R}_T$ return $(c := (c_m, \mathbf{b}')$.

While in the decryption phase, the message is recovered through the approximation of v_1 (from $\mathbf{sb}'$), as shown in Algorithm 3.

Algorithm 3 Saber.PKE.Dec($sk = \mathbf{s}, c = (c_m, \mathbf{b}')$) [9], [10]

$v_1 = \mathbf{b}'^T(\mathbf{s} \bmod p) \in \mathcal{R}_p$.
$m' = ((v_1 - 2^{\varepsilon_p - \varepsilon_T}c_m + b_2) \bmod p) \gg (\varepsilon_p - 1) \in \mathcal{R}_2$.
return m'.

The KEM Saber is built based on the encryption scheme with further ensuring the correctness of private key reusability []. According to FIPS 202 standard [], let $\mathcal{F} : \{0,1\}^* \rightarrow \{0,1\}^n$ and $\mathcal{G} : \{0,1\}^* \rightarrow \{0,1\}^{l\times n}$ denote the hash functions SHA3-256 and SHA3-512, respectively, we have the following Algorithms 4, 5, and 6 to represent the operations of the CCA-secure KEM Saber.

Algorithm 4 Saber.KEM.PKE.KeyGen() [9], [10]

$(seed_{\mathbf{A}}, \mathbf{b}, \mathbf{s})$=Saber.PKE.KeyGen().
$pk = (seed_{\mathbf{A}}, \mathbf{b})$.
$pkh = \mathcal{F}(pk)$.
$z_1 = \mathcal{U}(\{0,1\}^{256})$.
return $(pk := (seed_{\mathbf{A}}, \mathbf{b}),\ sk := (\mathbf{s}, z_1, pkh))$.

Algorithm 5 Saber.KEM.Encaps($pk = (seed_{\mathbf{A}}, \mathbf{b})$) [9], [10]

$m = \mathcal{U}(\{0,1\}^{256})$.
$(\hat{K}, c) = \mathcal{G}(\mathcal{F}(pk), m)$.
c=Saber.PKE.Enc($pk, m; r$).
$K = \mathcal{F}(\hat{K}, c)$.
return (c, K).

Algorithm 6 Saber.KEM.Decaps($sk = (\mathbf{s}, z, pkh), pk = (seed_{\mathbf{A}}, \mathbf{b}), c$) [9], [10]

m'=Saber.PKE.Dec($\mathbf{s}, c$).
$(\hat{K}', c') = \mathcal{G}(pkh, m')$.
c'=Saber.PKE.Enc($pk, m'; r'$).
if $c = c'$ then return $K = \mathcal{H}(\hat{K}, c)$.
else return $K = \mathcal{H}(z_1, c)$.

Parameter Setting. There are three sets of parameters setting for the NIST security levels 1, 3, and 5, respectively, called as LightSaber, Saber, and FireSaber. The polynomial degree is set as $N = 256$ and moduli $q = 2^{13}$ and $p = 2^{10}$. The related secrets are sampled from the binomial distribution as [-5,5] (LightSaber), [-4,4] (Saber), to [-3,3] (FireSaber), respectively [9], [10].

C. Polynomial Multiplication for KEM Saber

The polynomial multiplication is the key operation of the Saber protocol, where one polynomial involves small-size coefficients (e.g., in the range of -4 to +4 for Saber) and the other polynomial operand has coefficients of either 10-bit or 13-bit (the design for 13-bit can be reused for the 10-bit one). The most recent report of [18] has utilized this feature to derive a lightweight structure based on the schoolbook multiplication algorithm. Though this structure has advantages such as low-complexity, it has limitations of (i) fixed throughput rate and (ii) high memory access overhead, and hence further efforts need to be made in this area.

III. SMOP: Mathematical Formulation & Derivation

Overall Description. This section will present the general mathematical derivation strategy and then through the related processes to obtain the desired SMOP strategy.

A. Mathematical Formulation & Derivation

As discussed in Section 1, the most complicated operation for KEM Saber is the polynomial multiplication over ring $\mathbb{Z}_{p/q}/(x^N + 1)$. Without loss of generality, for simplicity of discussion, we can thus have the following definitions in a general format.

Definition 1. Following the above discussion, here we just define the polynomial multiplication over ring as

$$W = GD \bmod f(x), \tag{1}$$

where $f(x) = x^N + 1$, $W = \sum_{i=0}^{N-1} w_i x^i$, $G = \sum_{i=0}^{N-1} g_i x^i$, and $D = \sum_{i=0}^{N-1} d_i x^i$. Note that the w_i (t-bit), g_i (t-bit), and d_i (h-bit) are integers over ring $\mathbb{Z}_l/(x^N + 1)$, which will be determined by the specific PQC scheme later.

Proposed Mathematical Formulation & Derivation Strategy. For compact implementation, it is ideal that the original polynomial multiplication can be transferred into a number of small-size sub-components, where these sub-components can be realized through the form of serial accumulation, i.e., desirable for low-complexity compact and lightweight implementation. The conventional methods like Karatsuba algorithm [21], when the original polynomial multiplication is decomposed into smaller-size sub-polynomial-multiplications, the involved sub-components typically involve the irregular degree of x and thus are hard to process these sub-units in an orderly format [21]. Based on this consideration, we propose to use a novel derivation strategy to achieve this specific goal.

Following the above discussions, we set our mathematical formulation & derivation strategy as: (i) deriving the polynomial multiplication into the equivalent form of the additions of small-size sub-polynomial-multiplications (where each sub-polynomial-multiplication remains a certain degree of similarity and modularity); (ii) looking for unique/common features from these sub-components that they can be easily processed/implemented by a regular format for possible compact and lightweight implementation.

Following the above strategy, let us rewrite (1) as

$$\begin{aligned} W =& (Gd_0 + Gd_1 x + \cdots + Gd_{N-1}x^{N-1}) \bmod f(x) \\ =& G^{(0)}d_0 + G^{(1)}d_1 + \cdots + G^{(N-1)}d_{N-1}, \end{aligned} \tag{2}$$

where $G \bmod f(x) = G^{(0)} = G$, $Gx \bmod f(x) = G^{(1)}, \ldots,$ $Gx^{N-1} \bmod f(x) = G^{(N-1)}$. We can then remove the modulo reduction over $f(x)$ by substituting x^N with $x^N \equiv -1$ as

$$\begin{aligned} G^{(1)} =& -g_{N-1} + g_0 x + \cdots + g_{N-2}x^{N-1}, \\ G^{(2)} =& -g_{N-2} - g_{N-1}x + \cdots + g_{N-3}x^{N-1}, \\ & \cdots \ \cdots \ \cdots \\ G^{(N-1)} =& -g_1 - g_2 x - \cdots + g_0 x^{N-1}. \end{aligned} \tag{3}$$

Let $N = u \times v$ (u, v are integers). We can then define that

$$D = D_0 + D_1 x^u + D_2 x^{2u} + \cdots + D_{v-1}x^{(v-1)u}, \tag{4}$$

where

$$\begin{aligned} D_0 =& d_0 + d_1 x + d_2 x^2 + \cdots + d_{u-1} x^{u-1}, \\ D_1 =& d_u + d_{u+1} x + d_{u+2} x^2 + \cdots + d_{2u-1} x^{u-1}, \\ & \cdots \quad \cdots \quad \cdots \\ D_{v-1} =& d_{uv-u} + d_{uv-u+1} x + \cdots + d_{uv-1} x^{u-1}. \end{aligned} \tag{5}$$

Similarly, we can have $G = G_0 + \cdots + G_{v-1} x^{(v-1)u}$, where (the similar decomposition strategy applies to other $G^{(i)}$ for $1 \leq i \leq N-1$)

$$\begin{aligned} G_0 =& g_0 + g_1 x + g_2 x^2 + \cdots + g_{u-1} x^{u-1}, \\ G_1 =& g_u + g_{u+1} x + g_{u+2} x^2 + \cdots + g_{2u-1} x^{u-1}, \\ & \cdots \quad \cdots \quad \cdots \\ G_{v-1} =& g_{uv-u} + g_{uv-u+1} x + \cdots + g_{uv-1} x^{u-1}, \end{aligned} \tag{6}$$

Then, we can rewrite (2) into

$$\begin{aligned} W =& G(D_0 + D_1 x^u + \cdots + D_{v-1} x^{(v-1)u}) \bmod f(x) \\ =& GD_0 + G^{(u)} D_1 + \cdots + G^{(uv-u)} D_{v-1}, \end{aligned} \tag{7}$$

where the original polynomial multiplication has been decomposed into the addition of several sub-polynomial-multiplications in a preliminary way. For further decomposition, we can have the following derivations.

Without loss of generality, we just cover GD_0 of (7) first

$$\begin{aligned} GD_0 =& (G_0 + G_1 x^u + G_2 x^{2u} + \cdots + G_{v-1} x^{(v-1)u}) D_0 \\ =& G_0 D_0 + G_1 x^u D_0 + \cdots + G_{v-1} x^{(v-1)u} D_0, \end{aligned} \tag{8}$$

where we can further define that

$$T_0^{(0)} = G_0 D_0, \quad \cdots, \quad T_{v-1}^{(0)} = G_{v-1} x^{(v-1)u} D_0, \tag{9}$$

which can then be substituted into (8) to have

$$GD_0 = T_0^{(0)} + T_1^{(0)} + \cdots + T_{v-1}^{(0)}, \tag{10}$$

where the sub-polynomial-multiplication is further decomposed into the addition of smaller-size components (which satisfies the first aspect of the proposed derivation strategy).

It is clear that (consider $T_0^{(0)}$ first)

$$\begin{aligned} T_0^{(0)} =& G_0 (d_0 + d_1 x + d_2 x^2 + \cdots + d_{u-1} x^{u-1}) \\ =& G_0 d_0 + G_0^{(1)} d_1 + G_0^{(2)} d_2 + \cdots + G_0^{(u-1)} d_{u-1}, \end{aligned} \tag{11}$$

which can be transferred into a matrix-vector product form of (connecting with (3), note that $G_0 = G_0^{(0)}$)

$$\begin{aligned} [T_0^{(0)}] =& \begin{bmatrix} g_0 & -g_{N-1} & \cdots & -g_{N-u+1} \\ g_1 & g_0 & \cdots & -g_{N-u+2} \\ \vdots & \vdots & \ddots & \vdots \\ g_{u-1} & g_{u-2} & \cdots & g_0 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{u-1} \end{bmatrix} \\ =& [G_0][D_0], \end{aligned} \tag{12}$$

where $[G_0]$ is an $u \times u$ matrix. Interestingly, we can have the following observations on $[G_0]$: (i) the elements in the main diagonal are identical (say g_0); (ii) the rest elements are regularly distributed in two regions (the upper-right and the lower-left ones) and meanwhile the values in the specific region are symmetrically identical along with the direction of main diagonal of the matrix; (iii) the subscripts of the values of each row/column within each region are following a pattern of decreasing sequence (e.g., from g_{u-1} to g_0 and then to $-g_{N-u+1}$); (iv) there are actually in total $(2u-1)$ values contained in the $[G_0]$ (counting the related signs), namely $g_{u-1}, \cdots, g_0, \cdots, -g_{N-u+1}$, which is the values in the far left column and the first top row. These unique features indicate that all the elements within the matrix $[G_0]$ can be obtained through the circularly shifting of the coefficients of polynomial G, which facilitates the actual implementation (see the next Section).

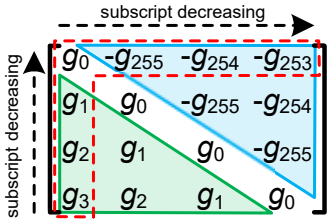

Fig. 1. Example of $N = 256$ and $u = 4$ ($[G_0]$), where the values are regularly distributed in the regions (colored areas).

For a clear demonstration and clarification, we have used a case study example of $N = 256$ and $u = 4$ and have shown $[G_0]$ in Fig. 1, where the mentioned two regions are highlighted as the blue and green areas, respectively. One can see that the actual values for this matrix, contained in the dotted red area, are in total $(2u - 1 = 7)$ numbers, where the subscripts are decreasing, following the sequence in Fig. 1. Besides that, the values in the specific region are symmetrically identical along with the line of main diagonal.

In summary, one can conclude that these unique properties are very much related to the elements in the matrix main diagonal, and the other elements are distributed following a specific order. Besides that, as the matrix size u (for $[G_0]$) is not a fixed number and it potentially involves scalability. Hence, we temporarily conclude these features as "Scalable Matrix Originated Diagonal Spreading (SMODS)".

For a more general conclusion, one can find that these observed unique features do not stick to $[G_0]$ only. In fact, these properties apply also to other sub-products of (8). For instance, we can have $T_1^{(0)}$ as

$$T_1^{(0)} = \begin{bmatrix} g_u & g_{u-1} & \cdots & g_1 \\ g_{u+1} & g_u & \cdots & g_2 \\ \vdots & \vdots & \ddots & \vdots \\ g_{2u-1} & g_{2u-2} & \cdots & g_u \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{u-1} \end{bmatrix} = [G_1][D_0], \tag{13}$$

where all the elements within the main matrix $[G_1]$ follow the same pattern of SMODS, as specified above.

Likewise, $T_2^{(0)}$, ..., $T_{v-1}^{(0)}$ of (10) can be transferred into the similar matrix-vector products and the involved matrices share the same features of SMODS, as those discussed above (connecting with (3)).

Similarly, $G^{(u)}D_1$ can be composed as

$$G^{(u)}D_1 = T_0^{(1)} + T_1^{(1)} + \cdots + T_{v-1}^{(1)}, \tag{14}$$

where $T_0^{(1)} = G_0^{(u)}D_1$, $T_1^{(1)} = G_1^{(u)}x^u D_1$, $\cdots$, $T_{v-1}^{(1)} = G_{v-1}^{(u)}x^{(v-1)u}D_1$. The same strategy can be extended to $G^{(2u)}D_2$, $\cdots$, $G^{(uv-u)}D_{v-1}$, as

$$\begin{aligned} G^{(2u)}D_2 =& T_0^{(2)} + T_1^{(2)} + \cdots + T_{v-1}^{(2)}, \\ & \cdots \ \cdots \ \cdots \\ G^{(uv-u)}D_{v-1} =& T_0^{(v-1)} + T_1^{(v-1)} + \cdots + T_{v-1}^{(v-1)}, \end{aligned} \tag{15}$$

where we can find that each sub-polynomial-multiplication of (7) has now been further decomposed into v number of sub-components. Besides that, all these sub-components can be transferred into the matrix-vector product forms, where the main $u \times u$ matrix have the features as SMODS, following the examples in (12), (13), and Fig. 1 (connecting (3)).

The above steps, mainly from (4)-(15), undoubtedly have fully satisfied the two mentioned aspects of the proposed mathematical derivation strategy. Hence, we can summarize the proposed decomposition strategy as follows:

Proposed SMOP Strategy. For a general polynomial multiplication over ring $\mathbb{Z}_l/(x^N+1)$, we can follow the above steps of (4)-(15) to decompose the original polynomial multiplication into the addition of in total v^2 number of regular sub-components, where each sub-component is equivalent to a matrix-vector product involved with the main matrix sharing the pattern of observed SMODS. As the matrices involved are following the features of SMODS, we here define this decomposition/processing strategy as **S**calable **M**atrix **O**riginated **P**rocessing (SMOP) strategy. The detailed algorithmic process for the polynomial multiplication with respect to the KEM Saber is presented below.

IV. SMOP BASED POLYNOMIAL MULTIPLICATION ALGORITHM FOR KEM SABER

Following the proposed SMOP strategy, we can further derive the desired polynomial multiplication algorithm for KEM Saber.

Let us firstly decompose W into v sub-polynomials as

$$W = W_0 + W_1 x^u + W_2 x^{2u} + \cdots + W_{v-1}x^{(v-1)u}, \tag{16}$$

where $W_0 = w_0 + w_1 x + w_2 x^2 + \cdots + w_{u-1}x^{u-1}$, $W_1 = w_u + w_{u+1}x + w_{u+2}x^2 + \cdots + w_{2u-1}x^{u-1}$, $\cdots$, $W_{v-1} = w_{uv-u-1} + w_{uv-u}x + \cdots + w_{uv-1}x^{u-1}$.

From (7), one can further have

$$\begin{aligned} W_0 =& T_0^{(0)} + T_0^{(1)} + \cdots + T_0^{(v-1)} = \sum_{j=0}^{v-1} T_0^{(j)}, \\ W_1 =& T_1^{(0)} + T_1^{(1)} + \cdots + T_1^{(v-1)} = \sum_{j=0}^{v-1} T_1^{(j)}, \\ & \cdots \ \cdots \ \cdots \\ W_{v-1} =& T_{v-1}^{(0)} + T_{v-1}^{(1)} + \cdots + T_{v-1}^{(v-1)} = \sum_{j=0}^{v-1} T_{v-1}^{(j)}. \end{aligned} \tag{17}$$

where each output sub-polynomial becomes the accumulation of v number of $T_k^{(j)}$ (for $W_k = \sum_{j=0}^{v-1} T_k^{(j)}$).

For compact implementation, we can thus have the proposed polynomial multiplication here as below

---

Algorithm 7 Proposed algorithm for the polynomial multiplication in KEM Saber

---

Inputs: G and D are polynomials (G and D are polynomials with coefficients over ring and the actual bit-width of these coefficients follows the specific KEM Saber).
Output: $W = GD \bmod f(x)$ ($f(x) = x^N + 1$).

1. Initialization (preparation) step

1.1. make ready input polynomials G and D.
1.2. $\overline{W} = 0$.

2. Main step

2.1. decompose D into $\{D_0, D_1, \cdots, D_{v-1}\}$. // see (4)
2.2. obtain $G^{(1)}$, $G^{(2)}$, $\cdots$, $G^{(N-1)}$ from G, respectively. // (3)
2.3. decompose G into $\{G_0, G_1, \cdots, G_{v-1}\}$. // see (6)
2.4. for $k = 0$ to $v-1$.
2.5. for $j = 0$ to $v-1$.
2.6. obtain all the corresponding $G_k^{(ju)}$.
2.7. $\overline{W} = \overline{W} + T_k^{(j)}$.//follow (17) & the SMOP strategy
2.8. end for.
2.9. $W_k = \overline{W}$.
2.10. end for.

3. Final step

3.1. obtain the output W from serially delivered W_k.

Details of the Algorithm. Overall, the procedures presented in Algorithm 7 are crystal clear (see also the detailed processes in Section 2) except the computation of each $T_k^{(j)}$ as well as the obtaining of related $G_k^{(ju)}$ during the actual implementation process. Here we present the details of them as below.

(a) Computation of Each $T_k^{(j)}$. The computation of each $T_k^{(j)}$ follows the regular calculation process presented in Section 2, i.e., transfer each $T_k^{(j)}$ into the equivalent matrix-vector product and then obtain the related output (u number) in parallel through point-wise multiplication-and-addition operations (Step 2.7 of Algorithm 7 is the serial accumulation of $T_k^{(j)}$).

For example, (12) can be calculated as

$$[T_0^{(0)}] = \begin{bmatrix} g_0 d_0 - g_{N-1}d_1 - \cdots - g_{N-u+1}d_{u-1} \\ g_1 d_0 + g_0 d_1 - \cdots - g_{N-u+2}d_{u-1} \\ \cdots \ \cdots \ \cdots \\ g_{u-1}d_0 + g_{u-2}d_1 + \cdots + g_0 d_{u-1} \end{bmatrix}, \tag{18}$$

which applies to other $T_k^{(j)}$ of Algorithm 7.

(b) Obtaining of Related $G_k^{(ju)}$ Sequentially. As the related T_k^j are serially accumulated, the obtaining of corresponding $G_k^{(ju)}$ also needs to be carried out in a sequential format. For

simplicity of discussion, we can have $[G_0^{(0)}]$ ($[G_0]$, see (12)) and $[G_0^{(u)}]$ (as below)

$$[T_0^{(u)}] = \begin{bmatrix} -g_{N-u} & -g_{N-u-1} & \cdots & -g_{N-2u+1} \\ -g_{N-u+1} & -g_{N-u} & \cdots & -g_{N-2u+2} \\ \vdots & \vdots & \ddots & \vdots \\ -g_{N-1} & -g_{N-2} & \cdots & -g_{N-u} \end{bmatrix}, \tag{19}$$

where there are actually $(2u-1)$ number of values involved within, i.e., $\{-g_{N-1}, \cdots, -g_{N-u}, \cdots, -g_{N-2u+1}\}$, according to the SMOP property (Section 2). Comparing with the actual $(2u-1)$ values contained in $[G_0]$, namely $\{g_{u-1}, \cdots, g_0, \cdots, -g_{N-u+1}\}$, these values (subscripts) are circularly related to one another and there also exist an overlap of $(u-1)$ values (i.e., $\{-g_{N-1}, \cdots, -g_{N-u+1}\}$). This property facilitates the using of circular shift-register (CSR) to deliver out the desired outputs per every cycle for the construction of proper $G_k^{(ju)}$ (the detailed hardware structure is presented in Section 4). Similar strategy applies to the following obtaining $G_0^{(2u)}$ from $G_0^{(u)}$, which can be extended to the obtaining of other $G_k^{(ju)}$ in a sequential order.

Another aspect of obtaining $G_k^{(ju)}$ in Algorithm 7 also involves the assigning of correct signs to the corresponding coefficient within a certain $[G_k^{(ju)}]$ since the original coefficients of the polynomial G are assumed to have positive values (no additional sign inverting). Again, here we combine the SMOP property (Section 2) with the feature of the sign distributions within two regions of the matrix $[G_k^{(ju)}]$ to obtain the accurate sign assignment, and the detailed implementation process can be seen in Section 4.

Parameters With Respect to the Specific PQC Scheme. Finally, when the proposed Algorithm 1 is applied to the actual PQC scheme, we need to take the corresponding parameter setting into consideration. Specifically, following the original setup, one polynomial for KEM Saber has coefficients of either 13-bit or 10-bit (the structure for the 13-bit can be used for the 10-bit one), while another polynomial (from the binomial sampler) involves coefficients of 4-bit [9] ([-5,5], [-4,4], [-3,3]). The polynomial-size N is fixed at 256. When Algorithm 7 is applied, we set G as the polynomial of 13-bit coefficients ($t = 13$), and D is the one with 4-bit coefficients ($h = 4$). Note that we follow the setup in [22], [23] that the coefficients of D are represented in the sign-magnitude binary numbers.

V. Proposed Compact Cryptoprocessor for KEM Saber

A. Design Strategy and Overview of the Coprocessor

While considering the existing fully hardware implementation of the cryptoprocessors, there are basically two types: the first type belongs to the traditional design strategy that the major arithmetic operations are mapped into the related hardware structures but the top-level signals such as input/output data flow, reset control, and clock signals are heavily rely on the outward resource support [24], which is "ideal"setup; while the second type emerges out recently that fully utilizes the available resource on modern hardware platform, like the System-on-a-Chip (SoC) based FPGA device, to build an instruction-set coprocessor, which not only includes the necessary arithmetic units required for the cryptosystem, but also integrates the complete instruction-set based control to coordinate the overall operation of the whole cryptosystem, ranging from the basic input/output data processing and clock signal setup to the operational phase switching and processor start & initiation, etc. Obviously, the mentioned latter type of design is more efficient than the former, especially considering the actual application of the implemented cryptoprocessor. Indeed, as the SoC FPGA has gradually become the mainstream device in the market, it is expected that the instruction-set coprocessor is highly desired.

With this point of view, in this paper, we follow the design style of the existing instruction-set coprocessor for KEM Saber [16] to obtain the proposed compact coprocessor. Despite the following of the available coprocessor design, we have also made quite a number of innovative efforts to uplift the built instruction-set coprocessor to a higher level: (i) we have innovatively transfer the proposed polynomial multiplication algorithm into desired hardware architecture in the way of generic and flexible, yet with very low resource usage and computational delay; (ii) we have also successfully deployed the proposed polynomial multiplication architecture into the coprocessor to match the other building components; (iii) we have also designed a novel input/output coordinator such that the deployed polynomial multiplication component can adequately operate with the instruction-set in a generic and flexible format; (iv) we have finally updated and improved all the necessary control signal setup, instruction-set, and data path manager for better coordination of the related building components to finalize the desired coprocessor.

Following the above-mentioned design strategy, we have thus presented the proposed cryptoprocessor for the KEM Saber, as shown in Fig. 2, where the major operations such as polynomial multiplication are based on Algorithms 1-7. The overall coprocessor consists of a number of necessary building blocks, i.e., the processor interface & control unit, memory blocks, data bus & manager, binomial sampler, polynomial multiplication block, and related input/output coordinator, Keccak core (Hash function), and other building components. The whole coprocessor is operating in the constant time according to the high-level instruction set, and all the individual building block involves constant computational cycles with respect to different operational phases of the KEM Saber, which is quite practical for further employing in emerging application environments.

The details of the involved building components of the proposed compact coprocessor are introduced below. In particular, we will give a thorough description of the proposed polynomial multiplication architecture along with necessary coordinating components for the proper operation of the proposed coprocessor.

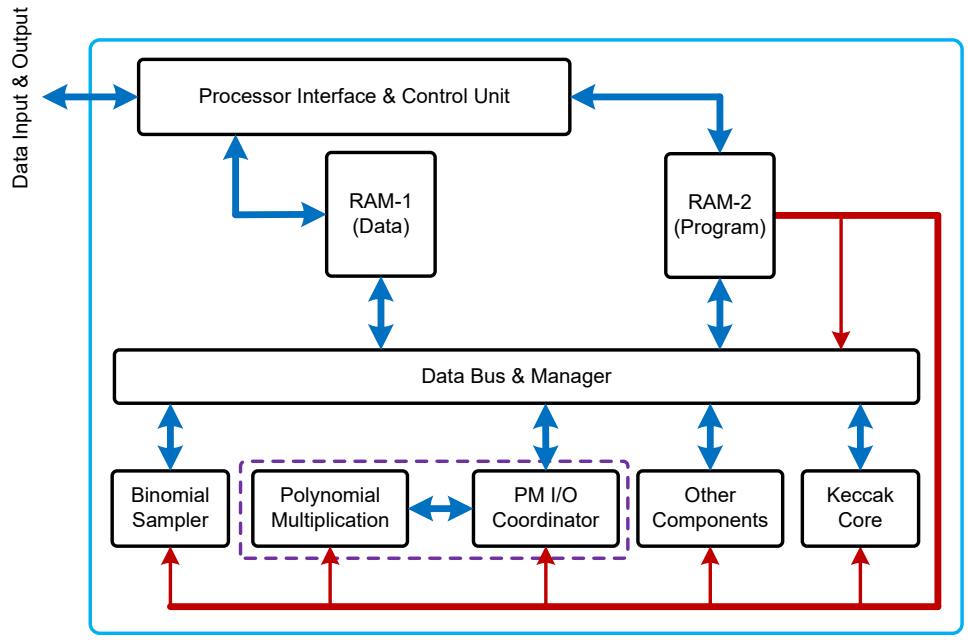

Fig. 2. High-level description of the proposed instruction-set compact coprocessor for KEM Saber. Blue line: data-path; red line: control signals. PM: polynomial multiplication.

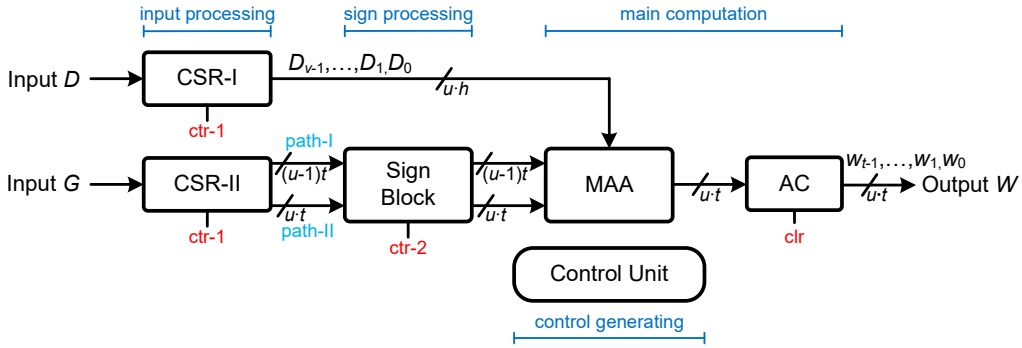

Fig. 3. The proposed polynomial multiplication structure (general format), where the number of the values processed in the signal path is marked but the actual bit-width of the signal depends on the specific condition ($t = 13, h = 4$). MAA: multiplication-and-addition. AC: accumulation. CSR: circular shift-register. PISO: parallel-in serial-out.

B. Data Memory: RAM-1

The data memory functions to store/read the data from/to the processor interface and the data bus & manager as well as those of the important components such as sampler, polynomial multiplier, Keccak core, etc. We follow the existing coprocessor design style and have chosen the block RAM of 64-bit available in the FPGA, which facilitates the fast and easy transmission of data between different building blocks within the coprocessor since one single word contains enough room for multiple values/signals. Meanwhile, this type of RAM selection also benefits the possible communication between host computers and reduces the potential communication cost on both data transfer and delay aspects.

C. Polynomial Multiplication Building Block

As introduced in Section 1. polynomial multiplication over ring can be regarded as the most important building block in the coprocessor, not only because the polynomial multiplication is the most computational intensive operation within the KEM Saber, but also that the other components such as Keccak core can be directly obtained from the open-access resources (though it has relatively large area-complexity) except the polynomial multiplier core. The details of the proposed polynomial multiplication architecture are presented below, along with several algorithm-architecture co-implementation techniques.

Overview of Polynomial Multiplication Structure (General Format). The overall polynomial multiplication structure (general format) based on Algorithm 7 is shown in Fig. 3, where it consists of five main components, namely the input processing component, the sign processing component, the main computation component, and the control generating component. In terms of the constitution of each component, there are: (i) two CSRs in the input processing component;

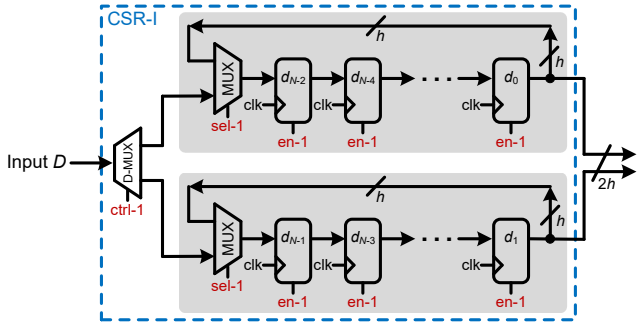

Fig. 4. The CSR-I ($u = 2$), where the registers are loaded with initial values.

(ii) a sign block in the sign processing component; (iii) one multiplication-and-addition (MAA) cell and one accumulation (AC) cell in the main computation component; and (iv) a control unit in the control generating component. Note that the structure in Fig. 3 is presented in a general format, and the specific bit-width of each data path depends on the setup in the specific KEM Saber scheme.

Generally, the input processing component (two CSRs) is firstly loaded with the necessary coefficients from the two inputs, and then in the following cycles, it produces the correct outputs to the following components. Connecting with Algorithm 7, while the CSR-I is producing D_j ($j = 0$ to $v - 1$) in a sequential format, the CSR-II is responsible for generating the necessary values (in total $2u - 1$ values) to construct the corresponding $G_k^{(ju)}$. Of course, the sign processing component (sign block) assists with the sign assigning to all the delivered $2u - 1$ values (in two paths) to form the accurate $G_k^{(ju)}$. When all the necessary values have been fed to the main computation component, the MAA cell functions to execute the computation of $T_k^{(j)}$ and the following AC cell executes the related accumulation to deliver the desired W_k ($k = 0$ to $v - 1$) as the output of the AC cell has u parallel output coefficients to be further stored in the external memory. The overall operation is carried out through different types of control signals (some of them have already been specified in Fig. 3) generated from the control unit.

The Input Processing Component. As seen from Fig. 4, there are two CSRs contained in this component. The CSR-I is responsible for generating the proper D_j ($j = 0$ to $v - 1$) to the MAA cell in a repeated format (repeats every v cycles). To realize this specific function, we have used a multi-path based CSR, as shown in Fig. 4, where we have presented a case study example when $u = 2$. This multi-path based CSR actually consists of two sub-CSRs (each sub-CSR has $N/2$ registers), where the input to each sub-CSR is directly by a De-MUX (D-MUX) and two connected MUXes attached to the input of the sub-CSR. During the loading time, the control signal to the D-MUX operates according to the sequence of "$0101\cdots0101$", which splits the coefficients of the polynomial D into two groups (each group corresponds to the specific sub-CSR), i.e., group of $\{d_{N-2}, d_{N-4}, \cdots, d_0\}$ and another group of $\{d_{N-1}, d_{N-3}, \cdots, d_1\}$. When all the necessary values are

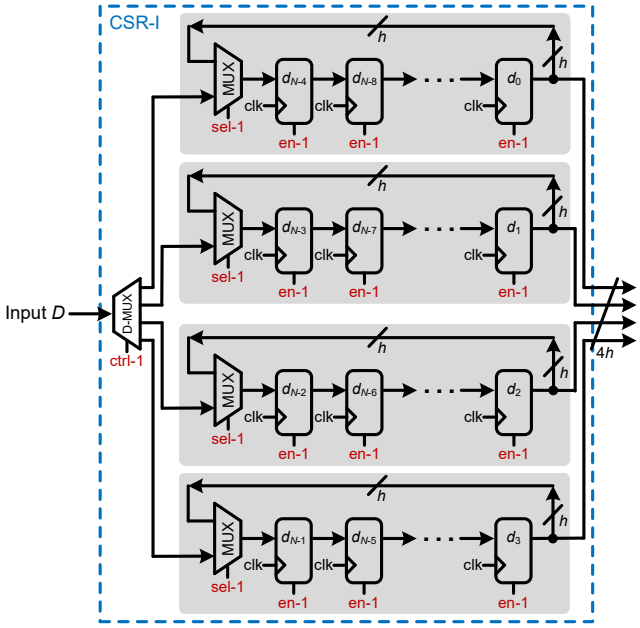

Fig. 5. The CSR-I ($u = 4$), where the registers are loaded with initial values.

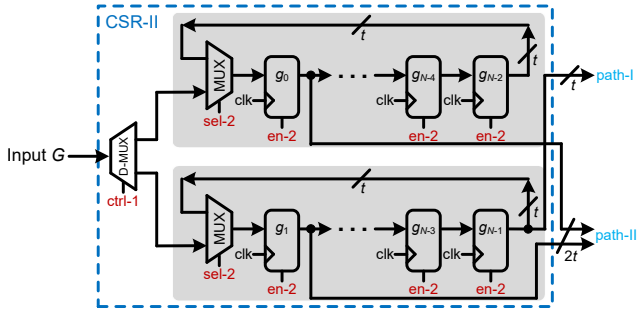

Fig. 6. The CSR-II ($u = 2$), where the values in the registers are initial values.

loaded into the sub-CSRs, the two MUXes then switch to close the loop such that in the following cycles, the output of the CSR-1 produces D_j ($j = 0$ to $v - 1$) correctly. The design of Fig. 4 can be extended to other values of u, such as the example shown in Fig. 5 for $u = 4$, where we have used 4 sub-CSRs (each with $N/4$ registers) and a 1-to-4 D-MUX for splitting the input coefficients into four groups as $\{d_{N-4}, d_{N-8}, \cdots, d_0\}$, $\{d_{N-3}, d_{N-7}, \cdots, d_1\}$, $\{d_{N-2}, d_{N-6}, \cdots, d_2\}$, and $\{d_{N-1}, d_{N-5}, \cdots, d_3\}$.

The CSR-II in the input processing component has a similar design structure as those in Figs. 4 and 5. For a clear demonstration, we have used $u = 2$ as the case study example again, as shown in Fig. 6. Comparing with the CSR-I in Fig. 4, the internal structure of the CSR-II is almost the same except on the output setup aspect. When connecting with the actual values contained in the two regions of Fig. 2 (applies to other matrices also), we firstly define that all the values in the upper-right region are generated by the path-I output of the CSR-II, while the values in the lower-left region, as well as the one in the main diagonal, are delivered out by the path-II.

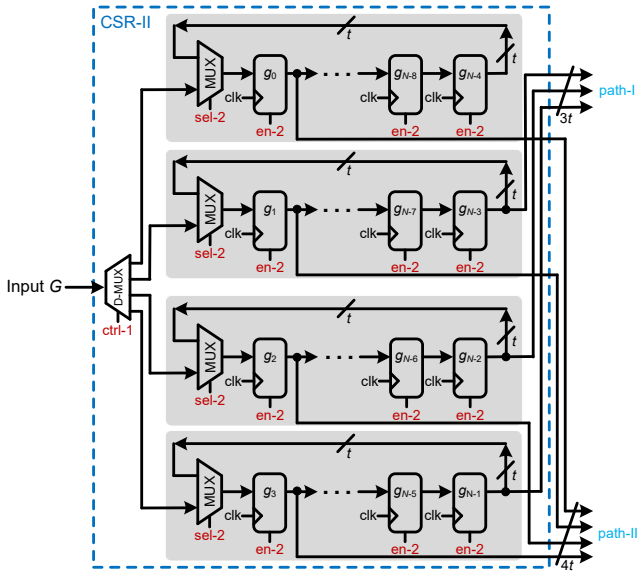

Fig. 7. The internal structure of the CSR-II ($u = 4$).

Considering the values contained within each $G_k^{(ju)}$, e.g., $G_0^{(0)}$ (G_0, see (12)), there are only $\{g_0, g_1, g_{255}\}$ involved (not counting the sign, which is done by the following sign block). Hence, as seen in Fig. 6, the far-right register's output (only bottom sub-CSR) is used for path-I delivering while both the far left registers' outputs (two sub-CSR) are used for path-II delivering. In second cycle, the CSR-II delivers the outputs of $\{g_{255}, g_{254}, g_{253}\}$, which is exactly the actual values contained in $G_0^{(2)}$ (connecting with (19)). When the desired output for $G_0^{(254)}$ (for the example here, at the $N/2$th cycle) is delivered (g_2, g_1, g_3), all the registers in the CSRs will be disenabled for one cycle, i.e., the same output values are delivered out for the next cycle, which matches the actual values contained within $G_0^{(2)}$ (see (19)). Then, the registers in the CSR-II will be enabled again in the following cycles (the disabling of registers in the CSR-II repeats every $N/2$ cycles until all the proper outputs are delivered).

In a more general sense, when u is set as other values, all the outputs of the far-right registers in all the sub-CSRs (not including the top one) are used to deliver the values required for path-I, while all the far-left registers (in all the sub-CSRs) are used to form the path-II output. We have also presented another example for $u = 4$, as seen in Fig. 7.

The Sign Processing Component. The sign block in the sign processing component functions to assign the delivered outputs from the CSR-II with proper signs according to the distribution within each $G_k^{(ju)}$. As shown in Fig. 8(a), there are basically two inverter cells (marked as $x = -x$) attached correspondingly to two MUXes. The inverter cell contains $(u-1)$ (or u) sign inverters according to the two's complement representation requirement that all the bits of a certain value are all inverted and then pass through the same number of half-adder (HD) (with one carry-in set as '1'), as shown in Fig. 8(b). The control signals (s-0 and s-1) of the MUXes are

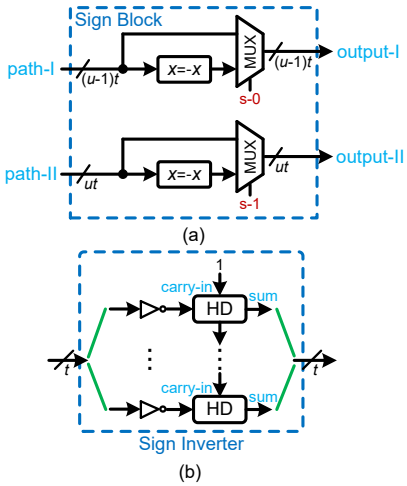

Fig. 8. The internal structures of: (a) the sign block; (b) the sign inverter.

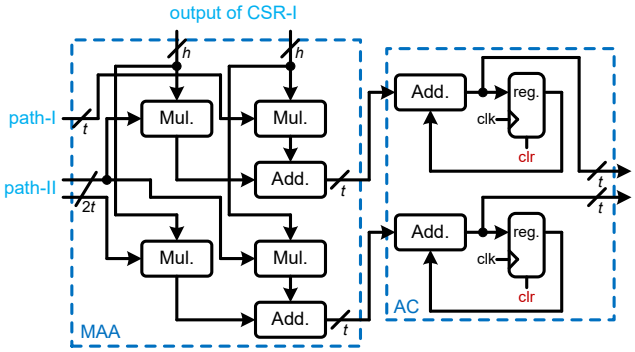

Fig. 9. The internal structure of the main computation component (for $u = 2$), where Mul. and Add. denote the multiplier and adder, respectively (reg. is the register).

generated by the control unit.

The Main Computation Component. The main computation component contains two cells, namely the MAA and the AC cells. The MAA cell is responsible for the calculation of corresponding $T_k^{(j)}$ in Step 2.7 of Algorithm 7, while the AC cell executes the following accumulation in the same step. As specified in Section 3 (see (18)), the MAA directly obtains the output from standard matrix-vector based calculation, and hence its general structure is shown in Fig. 9, based on the case study example of $u = 2$ (which applies to other values of u) as well as the internal structure of the AC cell.

As seen from Fig. 9, the MAA cell mainly consists of necessary multipliers and adders to perform the matrix-vector product (connecting (18)). In the case study example of $u = 2$, there is only one value contained in the upper-right region of the main matrix of $T_k^{(j)}$ (path-I) as well as the one element in the lower-left region (there are two values from path-II as the one in the main diagonal is also included). Following this setup, we can have the arrangement of multipliers and adders in the MAA cell, as shown in Fig. 9, where one input value (the element in the main diagonal) from path-II is reused twice as the input to the multiplier while the other input values

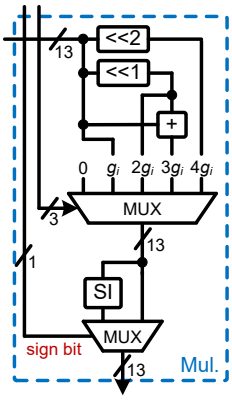

Fig. 10. The internal structure of the multiplier for Saber (SI: sign inverter, see Fig. 8).

(including the ones delivered from the CSR-I) are connected to the corresponding multipliers, respectively, following the principle of matrix-vector product (size of 2×2). The produced two outputs, namely the outputs of $T_k^{(j)}$, are then accumulated in the following AC cell through parallel processing to produce two outputs. Note that the outputs of the adders in the AC cell are directly connected to the outside as outputs of the main computation component for the sake of saving one extra clock cycle spent on the registers. The structure shown in Fig. 9 can easily be extended to the design of other values of u. **Note that** the registers can be inserted into the MAA cell to obtain pipelined processing to maintain high-speed performance.

As discussed in Section 1 (for Saber), the output value from the CSR-I (actually from the binomial sampler) lies in the range of [-4,4] [23], we hence can use its absolute magnitude to design the multiplier while the sign bit can be used in the following part to determine the actual multiplication result in either positive or negative status. As shown in Fig. 10, the absolute magnitude of the output from the CSR-I (3-bit) is used as the selecting signal to determine the multiplier's output result through the MUX (this part is similar to [23]). But after this, we have used another MUX (with the help of a sign inverter (SI)) to determine the actual result (where the sign bit is used as the selecting signal).

The Control Generating Component. This component plays a key role in the whole polynomial multiplication architecture as all the involved operations are under the coordination of all the related control signals generated from this component (control unit), including the sign control signals (sign block), clear signals (mainly for the registers), enable signals (mainly registers), and selecting signals (for MUXes/D-MUXes), etc. All the necessary control signals can be easily added/generated since we have used a double loop component centered control unit, as shown in Fig. 11, where the entire work status for the entire computation of $W = GD \bmod f(x)$ is divided into two stages, namely the loading and calculating stages. The details of the two working stages are introduced below.

During the load stage, the control unit takes N cycles to serially receive all the coefficients of input polynomials G and D into the corresponding registers in the CSRs. Note that the control signal (ctr-1, see Fig. 4) is set as '0' throughout this stage such that the two CSRs are all working in the loading mode. Once all the values are initiated in the related registers, the control unit switches to the calculating stage.

The overall calculating stage takes $(N/u)^2$ cycles to produce (N/u) batches of desired results, namely $[w_0, ..., w_{u-1}], ..., [w_{N-u}, w_{N-1}]$ (from W_0 to W_{v-1}), if no pipelined registers are inserted in the MAA cell. One part of the work for the control unit during this stage is to generate the necessary sign control signals for the sign block in Fig. 3 and Fig. 8. To achieve the accurate sign assigning to the correct coefficients, we have used a novel sign control generating strategy here: (i) first of all, we observe that the signs for all the element within a certain matrix $[G_k^{ju}]$ (connecting Algorithm 7) can be categorized into three conditions, namely a) all the values in the whole matrix have positive signs, b) the values in the lower-left region have positive signs but all negative signs in the upper-right region of the matrix, c) all the values in the whole matrix have negative signs; (ii) secondly, we hence propose to use indices y (horizontal) and z (vertical) to represent every element in the matrix $[G_k^{ju}]$ (e.g., $y = 0$ and $z = 0$ represent the element in the left-top corner of the matrix) such that we only need to consider three conditions of a) $y - z = 0$, b) $y - z = -1$, c) $z = (N/u) - 1$ (the transition between two states happens whenever one of three transition condition is satisfied), which can be realized by a three-state finite state machine (sign FSM). As shown in Fig. 11, the sign FSM produces the correct sign control signals, where the first bit of the sign signal determines the signs in the lower-left region (including the main diagonal), and the second bit of sign control signal determines the signs in the upper-right region of the matrix.

The control unit also sets the enable signal for the CSR-II to '0' when transition condition c) is satisfied because the main matrix in the last computation block of $T_k^{(v-1)}$ and the main matrix in the first computation block of $T_{k+1}^{(0)}$ consists of exactly the same values (with only different signs, as discussed in Section 3). Furthermore, the checking of the transition condition c) also enables the control unit to generate the clear signal for the AC cell as well as the loading signal for the final PISO component. The proposed control unit fully utilizes the reusability of checking transition of condition c) to reduce the area consumption for the entire control unit.

D. Polynomial Multiplication Input/Output Coordinator

This coordinator functions to facilitate the smooth input and output data processing between the data memory (data bus) and the polynomial multiplication building block. The input side is implemented with 2 FIFOs. One FIFO sends the secret coefficients to the multiplication core while another FIFO sends out the public coefficients. The output buffer takes responsibility to transfer u 13-bit output data into 64-bit memory words. (i) The first step is to expand each 13-bit data to 16 bit by inserting zeros at the head of the data. (ii)

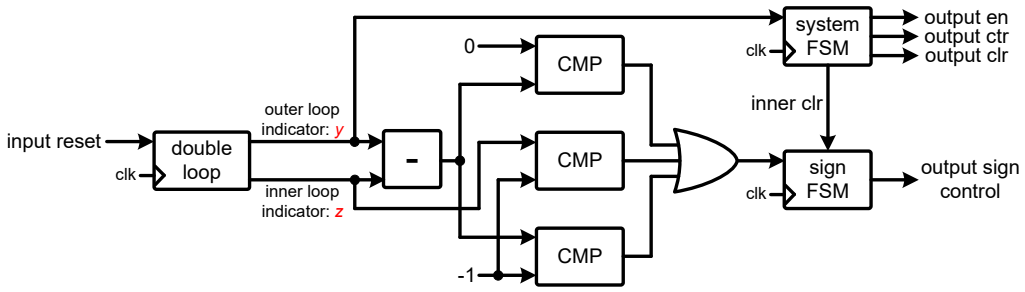

Fig. 11. The internal structure of the control unit, where y (horizontal) and z (vertical) are the indices for all the elements within one certain matrix $[G_k^{ju}]$ (connecting Algorithm 1). CMP: comparator. FSM: finite state machine.

While the next step depends on the value of u: (ii-a) when $u = 2$, the output data is not registered, instead of writing to a buffer; (ii-b) when $u = 4$, the output data is directly written to the memory; (ii-c) otherwise, when $u > 4$, the output data is registered and written back to the memory in $u/4$ cycles.

E. Keccak Core Building Block

The Hash functions SHA3-256 and SHA3-512 are necessary as required by the operations for KEM Saber, according to Algorithms 4, 5, and 6. We hence to use the open-source high-speed Keccak hardware core available in the literature [22] to realize this function, which is also similarly being used in other PQC schemes, including the recent hardware Saber of [16]. Besides that, as our proposed polynomial multiplication architecture possesses high-speed capability, we thus adopt the high-speed Keccak core used in the existing Saber coprocessor to generate high-speed and stable output, i.e., 1,344 bits of pseudo-random string every 28 cycles, to be used in other components within the coprocessor.

F. Binomial Sampler

The binomial sampler is used to generate the secret coefficients for the KEM Saber, including three different security levels as LightSaber [-5,5], Saber [-4,4], and FireSaber [-3,3]. Meanwhile, considering that the proposed polynomial multiplication architecture involves the assigning of signs to different values following the SMODS feature of the targeted specific "scalable matrix", we have decided to set the secret coefficients to be represented in the sign-magnitude format for the facilitating of sign installing. Based on this aspect of view, we follow the existing binomial sampler design [16] and have used it in our proposed design. This sampler's output is designed to be the values represented in the sign-magnitude form of 4-bit. This sampler's details can be seen at [16].

G. Data Bus & Manager

This building block functions to coordinate the other building components' working sequence and data transferring between the related blocks. Overall, this data bus & manager contains several FSMs and combinational circuit cells to realize the required function. This data bus & manager is controlled by the processor interface block through the program memory.

Specifically, for the proper operation of the proposed polynomial multiplication, the data bus component directs the data from RAM-1 to the polynomial multiplication I/O coordinator for further processing. When the polynomial multiplier delivers the desired output, the data bus & manager again coordinates with the I/O coordinator to transfer the output data to the data memory (RAM-1). Similar to operation operations required by the specific KEM Saber scheme.

Besides that, the data bus & manager always offers working priority to the processor interface & control unit component, i.e., the processor interface can reset any building block, and it also can write words into memory blocks at any time. While the other building blocks' working priorities are set as the same level, according to the data bus & manager building block's design, namely the component functions first is given priority.

H. Program Memory: RAM-2

Another block RAM is used to store and deliver corresponding control signals necessary for the proper operation of the coprocessor. Similar to RAM-1, RAM-2 is also designated as 37-bit for the ease of fast and simple read/write of required control signals. RAM-2 coordinates with the data bus building block and the processor interface component for the proper coordinating of generating correct processor-level control signals to the different building blocks such as sampler, polynomial multiplier and the related coordinator, Keccak core, and other components. In the actual instruction setup for RAM-2, all instruction words are 35-bit wide: 5-bit for instruction code, 10-bit for one input operand address while another 10-bit for another operand address and the last 10 bits are used for the result address.

I. Processor Interface & Control Unit

The processor interface component is used to help the coprocessor connect with the host processor embedded in the SoC FPGA device through AXI-4 interface. Inside of this interface component, seven 32-bit registers are being used, where four registers are used to transfer the data from the host processor to the coprocessor, and the remaining three are for delivering data from the coprocessor to the host processor. Note that the signals transferred from the processor are basically contained in two 64-bit words, one 64-bit word

contains the data information, and another word is used for containing the control signals. While the signals from the coprocessor side to be transferred to the host processor are just one 32-bit word that contains status information, and another 64-bit word contains data information. The key benefits of having this kind of setup in the processor interface component are that even when the host processor has a higher frequency than the coprocessor, both sides still can coordinate to work together.

While the involved control unit's function can be categorized into three main states: (i) handing the control priority to the host processor that it can read/write to the data memory or program memory directly; (ii) providing the necessary data context for the building blocks that are going to operate; (iii) waiting for the working/on-going building block to return the desired result for next step of the operation.

J. Other Components

We follow the existing coprocessor design [16] to setup the other components, namely AddPack, AddRound, Verify, CMOV, and Copy-Words components. The Verify component basically does the word-to-word comparison between the received ciphertext and re-encrypted ciphertext during a decapsulation process. The CMOV block is used to copy the decryption key or a pseudorandom string. While the AddRound block is designed to perform the point-wise addition of the constant h in Algorithms 1 and 2. The AddPack component performs the coefficient-wise addition of a constant in Algorithm 2 followed by the message. Since the involved computational complexity is relatively very small, the related resource usage is also minor when compared to the other building blocks.

VI. Complexity & Comparison

The whole coprocessor is coded in the way of mixed VHDL and Verilog, since some of the open-source core like the Keccak core is written in Verilog, but the proposed polynomial multiplication architecture is described in VHDL as well as related I/O coordinator. The binomial sampler and other component building blocks are also adopted from the open-source codes, but we have updated the related control signals and setups in the data bus & manager and the processor interface & control unit. Overall, the whole project is written in a generic format[1], and we have used the Vivado 2020.2 to synthesize and implement it on the targeted Xilinx UltraScale+ XCZU9EG-2FFVB1156 FPGA device. Due to the design nature of a compact coprocessor, we have selected $u = 2$, $u = 4$, and $u = 8$ for different security levels of KEM Saber (LightSaber, Saber, and FireSaber) for $N = 256$, and the detailed performance results are provided as follows.

A. Area Usage

The area usage for the mentioned PQC scheme, unified coprocessor for LightSaber, Saber, and FireSaber, with respect to different values of u are listed in Tables I, II, and III, respectively, where the frequency is set as 250MHz. We have also used extra registers in the MAA cell of the polynomial multiplication architecture to enhance the pipeline processing.

As seen from Tables I, II, and III, the proposed polynomial multiplication core (including the I/O coordinator) actually occupies very small resource usage when compared with other components involved within the coprocessor. For instance, even under the case of $u = 8$, the proposed polynomial multiplier still has very small resource usage, i.e., 2,162 LUTs and 1,656 FFs, which occupies only 21.4% and 21.5% LUT and FF usage in the whole coprocessor, respectively. Besides that, we have to mention that due to the using of pipelined technique in the MAA cell of the polynomial multiplication block, the frequency of the coprocessor is maintained at 250MHz, which is very ideal for practical usage.

TABLE I
AREA-COMPLEXITY FOR THE COMPACT COPROCESSOR WHEN $u = 2$, UNIFIED ARCHITECTURE FOR LIGHTSABER, SABER, AND FIRESABER. THE CLOCK FREQUENCY CONSTRAINT IS SET AS 250MHZ.

Building Block	LUTs	FFs	CLBs	DSPs	BRAMs
Keccak Core	5866	2982	920	0	0
Sampler	230	88	62	0	0
Polynomial Multiplier	324	58	70	0	0
I/O Coordinator	225	679	89	0	1
Other Blocks	1989	2890	498	0	2
Whole Coprocessor	8634	6697	1639	0	3
(% of overall FPGA)	3.15	1.22	4.78	0	0.33

TABLE II
AREA-COMPLEXITY FOR THE COMPACT COPROCESSOR WHEN $u = 4$, UNIFIED ARCHITECTURE FOR LIGHTSABER, SABER, AND FIRESABER. THE CLOCK FREQUENCY CONSTRAINT IS SET AS 250MHZ.

Building Block	LUTs	FFs	CLBs	DSPs	BRAMs
Keccak Core	5603	2985	886	0	0
Sampler	101	88	58	0	0
Polynomial Multiplier	996	634	217	0	0
I/O Coordinator	254	485	101	0	1
Other Blocks	2135	2892	477	0	2
Whole Coprocessor	9089	7084	1739	0	3
(% of overall FPGA)	3.32	1.29	5.08	0	0.33

TABLE III
AREA-COMPLEXITY FOR THE COMPACT COPROCESSOR WHEN $u = 8$, UNIFIED ARCHITECTURE FOR LIGHTSABER, SABER, AND FIRESABER. THE CLOCK FREQUENCY CONSTRAINT IS SET AS 250MHZ.

Building Block	LUTs	FFs	CLBs	DSPs	BRAMs
Keccak Core	5655	2984	888	0	0
Sampler	229	88	77	0	2
Polynomial Multiplier	2162	1656	448	0	0
I/O Coordinator	69	81	44	0	1
Other Blocks	1996	2890	483	0	2
Whole Coprocessor	10111	7699	1940	0	3
(% of overall FPGA)	3.69	1.40	5.66	0	0.33

[1]The source code of this work will be released upon acceptance for official publication in journal/conference.

TABLE IV
TIME-COMPLEXITY FOR THE UNIFIED COMPACT COPROCESSOR WHEN $u = 2$, $u = 4$, AND $u = 8$, RESPECTIVELY. THE CLOCK FREQUENCY CONSTRAINT IS SET AS 250MHZ.

Instruction	Cycles/Time (μs)		
	KeyGen	Encapsulation	Decapsulation
$u = 2$			
LightSaber	101840/407	135122/540	168670/675
Saber	151376/606	201170/805	251230/1005
FireSaber	200912/804	267218/1067	333790/1335
$u = 4$			
LightSaber	27632/111	36370/145	45374/181
Saber	40064/160	53042/212	66286/265
FireSaber	52496/210	69714/279	87198/345
$u = 8$			
LightSaber	9072/36	11538/46	14270/57
Saber	12224/49	15794/63	19630/79
FireSaber	15376/62	20050/80	24990/100

Meanwhile, one has to point out that the main resource usage of the proposed coprocessor is the Keccak core, which has more than 50% area occupation to the whole coprocessor. This indicates that if a higher efficient Keccak core is deployed, the overall area usage of the proposed coprocessor can be much less than the current version.

B. Timing Results

The time-complexities of the proposed compact coprocessor, in terms of the number of cycles and related computational time, with respect to different security levels of Saber, are calculated and listed in Table IV, where the frequency constraint is set as 250MHz.

As seen from Table IV, the proposed coprocessor has again very excellent performance on the timing performance, especially that the case of $u = 8$ achieves excellent time efficiency on three different operational phases.

C. Overall Performance Consideration

While taking the overall area-time complexities of the proposed coprocessor into careful consideration, we can find out that the proposed coprocessor under case of $u = 8$ achieves properly the best area-time efficiency among all the implemented cases, i.e., relatively low area usage yet with very efficient timing performance.

Meanwhile, we still have to mention that the cases of $u = 2$ and $u = 4$ are still promising in those resource-constrained application environments, especially that a lower complexity Keccak core is deployed for actual implementation at that time.

D. Comparison With The Existing Implementations

We have also listed the area-time complexities of the proposed coprocessor (case of $u = 8$) along with the existing implementations for a comparison, including the available KEM Saber implementations [13], [14], [16], [17] and other similar PQC designs [23]–[26] (Kyber).

As seen from Table V, when comparing with the existing KEM Saber implementations, the proposed coprocessor has significantly outperforms the existing designs of [13], [14], [16], [17]. Especially considering the most recent two fully hardware designs of [16], [17], respectively, the proposed coprocessor obtains much small resource usage while maintains relatively efficient time-complexity. For instance, when comparing with the design of [16], the proposed design involves more than 50% LUT and 20% FF usage, but only requires $\simeq 30\%$ longer time. While comparing with the design of [17], the proposed coprocessor has more balanced performance in terms of area usage and time-complexity. Note that the FPGA implementation results of [17] are based on the public-key encryption scheme, rather than the KEM scheme.

While considering the comparison with the existing implementations for Kyber, the proposed design again obtains very balanced performance on overall area-time complexity than the existing ones of [23]–[26]. Note that the design of [24] is not the cryptoprocessor style, since it does not involve the data memory and program memory as well as other related necessary components to support the designed instruction based operation in different phases. In fact, as seen from [24], it relies on the server and client structures to support the actual operation, which indicates that the actual resource usage and time-complexity are larger than the reported (only covers the server module). Besides that, considering the overall flexibility and resource usage aspect, the proposed compact coprocessor undoubtedly is more fit for various application environments where the resource is not that abundant.

E. Discussion

Overall, this is the FIRST paper about the compact coprocessor for KEM Saber, which achieves many unique aspects of performance: (i) the proposed SMOP strategy offers very low resource usage, as demonstrated by the performance data shown in Tables I, II, and III; (ii) the finalized KEM Saber coprocessor is suitable for various types of application environment, due to its flexible processing style; (iii) the designed SMOP based polynomial multiplication as well as the overall hardware coprocessor is generic and can be easily extended for actual deploying. Meanwhile, as the primary goal of the proposed work is to obtain compact implementation of KEM Saber, we do not take the other aspects of the hardware implementation into considering such as side-channel attack resistance, which can be seen as one of our future research directions. Finally, we also want to point out that the proposed KEM coprocessor has large area occupation on Keccak core, which can be replaced by the smaller-size one to obtain much higher efficiency in resource-constrained applications.

VII. CONCLUSION

This paper proposes a compact instruction-set coprocessor for KEM Saber based on a novel SMOP design strategy. Overall, we have proposed four innovative layers of coherent interdependent efforts to finalize the proposed compact coprocessor: (i) we have conducted a series of mathematical formulation process to derive a novel SMOP strategy for the

TABLE V
COMPARISON OF THE AREA-TIME COMPLEXITIES FOR THE PROPOSED AND EXISTING HARDWARE SABER IMPLEMENTATION ON THE FPGA PLATFORM

Design	Device	Freq. (MHz)	Time (KenGen./Enca./Deca.) (μs)	Area			
				LUT	FF	DSP	BRAM
Comparing with Existing Only KEM Saber Implementations							
[13]	Artix-7	125	3.2k/4.1k/3.8k	7.4k	7.3k	28	2
[14]	UltraScale+	322	-/60/65	$\simeq$12.5k	11.6k	256	4
[16]	UltraScale+	250	21.8/26.5/32.1	23.6k	9.8k	0	2
[17]	UltraScale+	160	6.7/7.2/2.6	28.1k	9.5k	85	6
Comparing with Existing Unified KEM Saber Implementations							
[16]	UltraScale+	150	18.4/26.9/33.6 36.4/44.1/53.6 60.2/68.4/82.0	24.9k	10.7k	0	2
[17][2]	UltraScale+	100	6.0/8.6/10.8 10.7/14.6/17.0 17.2/21.9/24.8	34.9k	9.9k	85	6
This Work ($u = 8$)	UltraScale+	250	36.3/46.2/57.1 48.9/63.2/78.5 61.5/80.2/100.0	10.1k	7.7k	0	3
With Other Implementations (Kyber)							
[24]*	Artix-7	161	23.4/30.5/41.3 39.2/47.6/62.3 58.2/67.9/86.2	7.4k	4.6k	3	2
[25]	Artix-7	210	-/14.3/20.9 -/19.2/26.5 -/27.4/35.2	11,864 11,884 12,183	10,348 10,380 12,441	15	8
[23]	-	-	-	24K	11k	32	18
[26]	Artix-7	59	12,034/16,458/14,746 - 37,339/44,390/41,169	1,842	1,634	34	5

[1]: The time refers to the delay time of the hardware structure/coprocessor at different security levels (from LightSaber to FireSaber, for instance), in a way of from top to bottom for one specific design.
[2]: The FPGA implementation results (area usage) shown here only supports the PKE scheme, not the KEM scheme. But the time is calculated based on the KEM scheme.
*: Note that the design of [24] is not a cryptoprocessor style, since it does not involve the data memory and program memory as well as other related necessary components such as the instruction-set and instruction analyser. In fact, as seen from [24], it relies on the server and client based operations to support the actual operation, which indicates that the actual resource usage and time-complexity are larger than the reported (only covers the server module), from this point of view.

polynomial multiplication of the KEM Saber in a general format; (ii) we have then proposed the SMOP strategy based polynomial multiplication algorithm as well as related arithmetic details and structural innovation for practical implementation; (iii) we have also constructed a new compact coprocessor for KEM Saber deploying the proposed polynomial multiplication architecture, following the existing coprocessor design style; (iv) we have provided thorough complexity and comparison to show that the proposed compact coprocessor has better balanced area-time complexity than the existing PQC hardware implementations.

The proposed compact coprocessor as well as the design strategy, offer many unique features: (a) very low-complexity; (b) flexible processing style; (c) design generic; and (d) fit for various application environments. The outcome of this work is expected to serve as an important reference for NIST PQC standardization process and a useful prototype for compact PQC hardware cryptoprocessor design.

ACKNOWLEDGEMENT

The authors would like to thank the support from NIST.

REFERENCES

[1] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th annual symposium on foundations of computer science*, pp. 124–134, Ieee, 1994.
[2] J. Xie, K. Basu, K. Gaj, and U. Guin, "Special session: The recent advance in hardware implementation of post-quantum cryptography," in *2020 IEEE 38th VLSI Test Symposium (VTS)*, pp. 1–10, IEEE, 2020.
[3] D. Micciancio and O. Regev, "Lattice-based cryptography," in *Post-quantum cryptography*, pp. 147–191, Springer, 2009.
[4] P. quantum cryptography round 3 submissions. https://csrc.nist.gov/projects/post-quantumcryptography/round 3-submissions.
[5] G. Alagic, J. Alperin-Sheriff, D. Apon, D. Cooper, Q. Dang, J. Kelsey, Y.-K. Liu, C. Miller, D. Moody, R. Peralta, *et al.*, "Status report on the second round of the nist post-quantum cryptography standardization process," *US Department of Commerce, NIST*, 2020.
[6] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *Journal of the ACM (JACM)*, vol. 56, no. 6, pp. 1–40, 2009.
[7] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 1–23, Springer, 2010.
[8] A. Banerjee, C. Peikert, and A. Rosen, "Pseudorandom functions and lattices," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 719–737, Springer, 2012.

[9] J.-P. D'Anvers, A. Karmakar, S. S. Roy, F. Vercauteren, J. Mera, M. Beirendonck, and A. Basso, "Saber: Mod-lwr based kem (round 3 submission),"

[10] 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," in *International Conference on Cryptology in Africa*, pp. 282–305, Springer, 2018.

[11] A. Karmakar, I. Verbauwhede, *et al.*, "Saber on arm. cca-secure module lattice-based key encapsulation on arm," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 243–266, 2018.

[12] J. M. B. Mera, A. Karmakar, and I. Verbauwhede, "Time-memory trade-off in toom-cook multiplication: an application to module-lattice based cryptography," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 222–244, 2020.

[13] 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 *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2020.

[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 *2019 International Conference on Field-Programmable Technology (ICFPT)*, pp. 206–214, IEEE, 2019.

[15] T. Fritzmann, G. Sigl, and J. Sepúlveda, "Risq-v: Tightly coupled risc-v accelerators for post-quantum cryptography," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 239–280, 2020.

[16] S. S. Roy and A. Basso, "High-speed instruction-set coprocessor for lattice-based key encapsulation mechanism: Saber in hardware," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 443–466, 2020.

[17] Y. Zhu, M. Zhu, B. Yang, W. Zhu, C. Deng, C. Chen, S. Wei, and L. Liu, "Lwrpro: An energy-efficient configurable crypto-processor for module-lwr," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 3, pp. 1146–1159, 2021.

[18] A. Basso and S. S. Roy, "Optimized polynomial multiplier architectures for post-quantum kem saber," *Cryptology ePrint Archive*, vol. Report 2020/1482, 2020.

[19] J. M. Pollard, "The fast fourier transform in a finite field," *Mathematics of computation*, vol. 25, no. 114, pp. 365–374, 1971.

[20] D. Hofheinz, K. Hövelmanns, and E. Kiltz, "A modular analysis of the fujisaki-okamoto transformation," in *Theory of Cryptography Conference*, pp. 341–371, Springer, 2017.

[21] C.-Y. Lee and J. Xie, "Efficient subquadratic space complexity digit-serial multipliers over $gf(2^m)$ based on bivariate polynomial basis representation," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 253–258, IEEE, 2020.

[22] K. T. K. in VHDL: High-speed core. https://keccak.team/hardware.html November.

[23] T. Fritzmann, G. Sigl, and J. Sepúlveda, "Risq-v: Tightly coupled risc-v accelerators for post-quantum cryptography," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 239–280, 2020.

[24] Y. Xing and S. Li, "A compact hardware implementation of cca-secure key exchange mechanism crystals-kyber on fpga," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 328–356, 2021.

[25] V. B. Dang, F. Farahmand, M. Andrzejczak, K. Mohajerani, D. T. Nguyen, and K. Gaj, "Implementation and benchmarking of round 2 candidates in the nist post-quantum cryptography standardization process using hardware and software/hardware co-design approaches," *Cryptology ePrint Archive: Report 2020/795*, 2020.

[26] E. Alkim, H. Evkan, N. Lahr, R. Niederhagen, and R. Petri, "Isa extensions for finite field arithmetic," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 219–242, 2020.