SMALL: Scalable Matrix OriginAted Large Integer PoLynomial Multiplication Accelerator for Lattice-based Post-Quantum Cryptography

Jiafeng Xie[1], Pengzhou He[1], Samira Carolina Oliva Madrigal[2], and Çetin Kaya Koç[3]

[1] Department of Electrical and Computer Engineering, Villanova University, Villanova PA, USA, {jiafeng.xie,phe}@villanova.edu
[2] San José State University, scolivamadrigal@gmail.com
[3] NUAA, Iğdır University, UCSB, cetinkoc@ucsb.edu

Abstract. Along with the rapid development in quantum computing, more attention has been switched to post-quantum cryptography (PQC) and related research including their hardware implementations. Following this trend, this paper presents a novel strategy to implement a special type of polynomial multiplication used in lattice-based PQC, where the coefficients of two input polynomials are unequal, and modulus and polynomial size are power-of-two numbers (not in favor of deploying number theoretic transform). In particular, we have proposed a Scalable Matrix originAted Large integer poLynomial multiplication Accelerator (SMALL) for flexible and compact implementation of the targeted polynomial multiplication that is constant-time. In total, our efforts include: (i) we have formulated and derived a scalable matrix originated computation strategy for the targeted polynomial multiplication in a general format; (ii) we have then presented the detailed internal structures for the proposed polynomial multiplication accelerator based on novel algorithm-to-architecture design techniques; (iii) we have implemented the proposed accelerator based on two case study PQC schemes to demonstrate the superior efficiency of the proposed design over the state-of-the-art solutions. We hope the outcome of this work will be useful for further PQC development.

Keywords: Hardware design · lattice-based PQC · polynomial multiplication accelerator · scalable matrix originated computation.

1 Introduction

In light of the fast advances in quantum technology, the post-quantum cryptography (PQC) related research and development have reached an all-time high [23]. Quite a number of cryptographic algorithms have been proposed for possible PQC candidates, and lattice-based cryptography is regarded as one of the most important categories of algorithms due to their strong security proof and relatively easy implementation complexity [18,15]. Along with this direction of

exploration, advances in the field have gradually switched to the hardware implementation side [14,23].

As polynomial multiplication over ring $\mathbb{Z}_q[x]/(x^n + 1)$ is one of the most important components for lattice-based PQC, a number of works have been released on related hardware implementation [20,5]. In particular, we notice a special type of polynomial multiplication where the coefficients of the two inputs are unequal, and the corresponding polynomial length and modulus q are numbers of power-of-two. This polynomial multiplication can be observed in one of the National Institute of Standards and Technology (NIST) PQC standardization third-round candidates Saber [1] and ring binary learning-with-errors (RBLWE)-based encryption scheme (RBLWE-ENC, a promising lightweight PQC suitable for lightweight applications) [2,4].

Motivation. Unlike the other large integer polynomial multiplications used in typical lattice-based PQC that can be implemented with popular fast algorithm number theoretic transform (NTT) [17], the targeted polynomial multiplication here is not in favor of deploying of NTT unless field extension is executed (which is out of the scope of this research). Meanwhile, due to its specific parameter sets that the two input polynomials' coefficients do not have the same size, deploying traditional fast algorithms such as Karatsuba or Toeplitz Matrix-Vector Product (TMVP) may cause the small coefficients involved operations to become larger-sized computations (more resource usage) because of the pre-addition related operations (may eventually offset the gain) [2]. As a result, the recent works based on traditional fast algorithms all have relatively large resource usage [2,11,21]. In this case, a compact design (also with flexibility) for this polynomial multiplication is highly desirable.

Proposal. Following these considerations, in this paper, we propose a novel polynomial multiplication algorithm to design the targeted accelerator, i.e., a Scalable Matrix originAted Large integer poLynomial multiplication Accelerator (SMALL). In particular, we have made three layers of innovative efforts:

(i) We present a detailed mathematical formulation process to propose a novel scalable matrix originated computation strategy for the targeted polynomial multiplication in a general format.

(ii) We then provide the architecture design process and related component details to obtain the proposed SMALL with the help of novel algorithm-to-architecture design techniques.

(iii) Finally, we give a thorough evaluation to showcase the superior performance of the proposed accelerator over the state-of-the-art solutions (based on two case study examples).

Overall Layout. The rest of the paper is arranged as follows. The proposed polynomial multiplication algorithm is formulated in Section 2. Details of the proposed accelerator are provided in Section 3. The evaluation is conducted in Section 4. And the conclusion is given in Section 5.

2 Related Work

This work follows a rather simple method from numerical linear algebra or a technique from polynomial arithmetic transformations (i.e., polynomial matrix multiplication). Specifically, the process of transforming the multiplication of two polynomials into a Matrix-Vector Product, which some works may describe as block-matrix multiplication. This type of transformation and strategies with similar decomposition is observed in numerous cryptographic works and the sciences in general and can be described as follows [7,8,12,13,16,19,22].

Consider an $(n \times n)$ matrix decomposed into v^2 total $(u \times u)$ sub-matrices and a vector decomposed into v total $(1 \times u)$ subvectors. In a similar manner, we can express the multiplication of two univariate polynomials $A(x)$ and $B(x)$ consisting of n coefficients each. First, each of these can be expressed as a finite sum of terms where each term consists of a coefficient and power according to the respective position (e.g., $A(x) = a_0 + a_1x + a_2x^2 + ... + a_{n-1}x^{n-1}$). Second, the product of two such polynomials can be decomposed in different ways according to the computational goal. For example, if our goal is a small Matrix-Vector Product decomposition, we can decompose, say, $A(x)$ into a sum of v smaller polynomials each of degree $u-1$. For instance, $A(x) = (a_0 + a_1x + ... + a_{u-1}x^{u-1}) + (a_u + a_{u+1}x + ... + a_{2u-1}x^{u-1}) + ... + (a_{uv-u} + a_{uv-u+1}x + ... + a_{uv-1}x^{u-1}) = (A_0 + A_1 + ... + A_{v-1})$. Clearly, we can multiply out with $B(x)$ to have $B(x)(A_0 + A_1 + ... + A_{v-1})$. Then, for each such sub-product $(e.g., B(x)(A_0))$ can we can decompose $B(x)$ and multiply out to have a sum of v^2 small products (e.g., $(B_0A_0 + B_0A_1 + ... + B_0A_{v-1}) + (B_1A_0 + B_1A_1 + ... + B_1A_{v-1}) + ... + (B_{v-1}A_0 + B_{v-1}A_1 + ... + B_{v-1}A_{v-1})$. Each such small product (e.g., B_0A_0) can be expressed in Matrix-Vector Product form where B_0 accounts for multiplication with respective power terms of A_0 and becomes a $(u \times u)$ matrix and A_0 is left as a vector of coefficients or a $(u \times 1)$ vector. Ideally, we would like to explore further optimizations in addition to small component decomposition and parallelism in hardware.

3 SMALL: Proposed Algorithm

Mathematical Formulation. Without loss of generality, we can just define the targeted polynomial multiplication over ring $\mathbb{Z}_q/(x^n + 1)$ 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 $D = \sum_{i=0}^{n-1} d_i x^i$ (w_i (t-bit), g_i (t-bit), and d_i (h-bit) are integers over ring such that $t = \log_2 q$ and $h < t$, and the actual h and t are determined by the specific PQC scheme).

Proposed Mathematical Derivation Strategy. For efficient implementation, it will be ideal that the original polynomial multiplication can be transformed into a number of small-size sub-components, where these sub-components can be realized through the form of serial accumulation (desirable for low-complexity implementation). Following this principle, we set our mathematical

formulation and derivation strategy with the following goals: (i) deriving the polynomial multiplication into the equivalent form of the additions of small-size sub-polynomial-multiplications (where each sub-polynomial-multiplication retains certain degree of similarity and modularity); (ii) looking for unique/common features from these sub-components such that they can be easily processed by a format of low-resource usage.

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

$$\begin{aligned} W =& (Gd_0 + Gd_1x + \cdots + Gd_{n-1}x^{n-1}) \bmod f(x) \\ =& G^{(0)}d_0 + G^{(1)}d_1 + \cdots + G^{(n-1)}d_{n-1}, \end{aligned} \tag{2}$$

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 substitute x^n with $x^n \equiv -1$ to have

$$\begin{aligned} G^{(1)} =& -g_{n-1} + g_0x + \cdots + g_{n-2}x^{n-1}, \\ & \cdots \ \cdots \ \cdots \\ G^{(n-1)} =& -g_1 - g_2x - \cdots + g_0x^{n-1}. \end{aligned} \tag{3}$$

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

$$D = D_0 + D_1x^u + D_2x^{2u} + \cdots + D_{v-1}x^{(v-1)u}, \tag{4}$$

where $D_0 = d_0 + d_1x + d_2x^2 + \cdots + d_{u-1}x^{u-1}$, $D_1 = d_u + d_{u+1}x + d_{u+2}x^2 + \cdots + d_{2u-1}x^{u-1}$, $\cdots$, $D_{v-1} = d_{uv-u} + d_{uv-u+1}x + \cdots + d_{uv-1}x^{u-1}$.

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 $1 \le i \le n-1$) $G_0 = g_0 + g_1x + g_2x^2 + \cdots + g_{u-1}x^{u-1}$, $G_1 = g_u + g_{u+1}x + g_{u+2}x^2 + \cdots + g_{2u-1}x^{u-1}$, $\cdots$, $G_{v-1} = g_{uv-u} + g_{uv-u+1}x + \cdots + g_{uv-1}x^{u-1}$. Then, we can rewrite (2) into

$$\begin{aligned} W =& G(D_0 + D_1x^u + \cdots + D_{v-1}x^{(v-1)u}) \bmod f(x) \\ =& GD_0 + G^{(u)}D_1 + \cdots + G^{(uv-u)}D_{v-1}, \end{aligned} \tag{5}$$

where the original polynomial multiplication has been decomposed into the addition of several sub-polynomial-multiplications. For further decomposition, we just cover GD_0 of (5) first (without loss of generality)

$$\begin{aligned} GD_0 =& (G_0 + G_1x^u + G_2x^{2u} + \cdots + G_{v-1}x^{(v-1)u})D_0 \\ =& G_0D_0 + G_1x^uD_0 + \cdots + G_{v-1}x^{(v-1)u}D_0, \end{aligned} \tag{6}$$

where we define $T_0^{(0)} = G_0D_0$, $\cdots$, $T_{v-1}^{(0)} = G_{v-1}x^{(v-1)u}D_0$. We can then substitute them into (6) to have $GD_0 = T_0^{(0)} + T_1^{(0)} + \cdots + T_{v-1}^{(0)}$, 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)

$$\begin{aligned} T_0^{(0)} =& G_0(d_0 + d_1x + d_2x^2 + \cdots + d_{u-1}x^{u-1}) \\ =& G_0d_0 + G_0^{(1)}d_1 + G_0^{(2)}d_2 + \cdots + G_0^{(u-1)}d_{u-1}, \end{aligned} \tag{7}$$

which can be transformed into a matrix-vector product form of (connecting (3))

$$[T_0^{(0)}] = \begin{bmatrix} g_0 & -g_{n-1} & \cdots & -g_{N-u+1} \\ g_1 & g_0 & \cdots & -g_{N-u+2} \\ \vdots & \vdots & \ddots & \vdots \\ g_{u-1} & g_{u-2} & \cdots & g_0 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{u-1} \end{bmatrix} = [G_0][D_0], \tag{8}$$

where $[G_0]$ is an $u \times u$ matrix. Then, from $[G_0]$ we observe that: (i) the elements in the main diagonal are identical (say g_0); (ii) the rest of 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 the 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 Section 3).

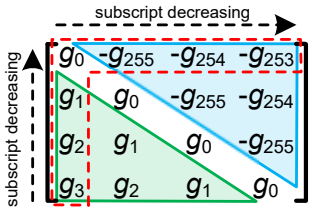

Fig. 1: Example of $n = 256$ and $u = 4$ ($[G_0]$), where the values are regularly distributed in the regions (colored areas).

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 contained in the dotted red area are in total $(2u - 1 = 7)$ numbers (where the subscripts are decreasing). Besides, the values in the respective region are symmetrically identical along with the line of the main diagonal.

In summary, one can conclude that these unique properties are very much related to the elements in the matrix ($[G_0]$) main diagonal and the other elements are distributed following a specific order. Besides that, the matrix size u is not a fixed number, i.e., scalable matrix originated computation.

For a more general conclusion, one can find that these observed unique features do not apply to $[G_0]$ only. In fact, these properties apply also to other

sub-products of (6). For instance, we can have $T_1^{(0)}$ as

$$T_1^{(0)} = \begin{bmatrix} g_u & g_{u-1} & \cdots & g_1 \\ g_{u+1} & g_u & \cdots & g_2 \\ \vdots & \vdots & \ddots & \vdots \\ g_{2u-1} & g_{2u-2} & \cdots & g_u \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{u-1} \end{bmatrix} = [G_1][D_0], \tag{9}$$

where all the elements within $[G_1]$ follow the same pattern (as specified above). Likewise, $T_2^{(0)}, \ldots, T_{v-1}^{(0)}$ can be transformed into similar matrix-vector products, and the involved matrices share the same features.

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

$$G^{(u)}D_1 = T_0^{(1)} + T_1^{(1)} + \cdots + T_{v-1}^{(1)}, \tag{10}$$

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

$$\begin{aligned} G^{(2u)}D_2 =& T_0^{(2)} + T_1^{(2)} + \cdots + T_{v-1}^{(2)}, \\ & \cdots \quad \cdots \quad \cdots \\ G^{(uv-u)}D_{v-1} =& T_0^{(v-1)} + T_1^{(v-1)} + \cdots + T_{v-1}^{(v-1)}, \end{aligned} \tag{11}$$

where we can find that each sub-polynomial-multiplication of (5) has now been further decomposed into v number of sub-components. Besides that, all these sub-components can be transformed into the matrix-vector product forms, following the examples presented in (8), (9), and Fig. 1.

The above steps, mainly from (4)-(11), undoubtedly have fully satisfied the mentioned two goals of the proposed mathematical derivation strategy. Hence, we can summarize the proposed decomposition strategy as follows:

Proposed Strategy. For a general polynomial multiplication over ring $\mathbb{Z}_q/(x^n+1)$, we propose a constant-time solution in which we can follow the above steps of (4)-(11) to decompose the polynomial multiplication into the addition of a total v^2 number of regular sub-components, where each sub-component is equivalent to a matrix-vector product involved with a main matrix sharing the pattern of scalable matrix based processing.

Overall, the whole polynomial multiplication can be computed as follows. Let us again decompose W into v sub-polynomials as

$$W = W_0 + W_1 x^u + W_2 x^{2u} + \cdots + W_{v-1}x^{(v-1)u}, \tag{12}$$

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 (5), one can further have $W_0 = T_0^{(0)} + T_0^{(1)} + \cdots + T_0^{(v-1)} = \sum_{j=0}^{v-1} T_0^{(j)}$, $W_1 = T_1^{(0)} + T_1^{(1)} + \cdots + T_1^{(v-1)} = \sum_{j=0}^{v-1} T_1^{(j)}$, $\cdots$, $W_{v-1} = T_{v-1}^{(0)} + T_{v-1}^{(1)} + \cdots + T_{v-1}^{(v-1)} = \sum_{j=0}^{v-1} T_{v-1}^{(j)}$, where each output sub-polynomial becomes the accumulation of v number of $T_k^{(j)}$ (for $W_k = \sum_{j=0}^{v-1} T_k^{(j)}$). We can thus have:

Algorithm 1 Proposed polynomial multiplication algorithm (general form)

Inputs: G and D are polynomials (the actual bit-width of the coefficients of G and D follows the specific PQC scheme).
Output: $W = GD \bmod f(x)$ (where $f(x) = x^n + 1$).
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 $\{G_0, G_1, \cdots, G_{v-1}\}$.
  2.4. for $k = 0$ to $v - 1$.
    2.5. for $j = 0$ to $v - 1$.
    2.6. obtain all the corresponding $G_k^{(ju)}$. // (13)
    2.7. $\overline{W} = \overline{W} + T_k^{(j)}$.// scalable matrix based processing strategy (14)
    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 1 are clearly expressed (see the above-detailed derivation processes) 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) 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 instance, we can have $[G_0^{(0)}]$ as

$$[T_0^{(u)}] = \begin{bmatrix} -g_{n-u} & -g_{n-u-1} & \cdots & -g_{n-2u+1} \\ -g_{n-u+1} & -g_{n-u} & \cdots & -g_{n-2u+2} \\ \vdots & \vdots & \ddots & \vdots \\ -g_{n-1} & -g_{n-2} & \cdots & -g_{n-u} \end{bmatrix}, \tag{13}$$

where there are actually $(2u-1)$ number of values involved within, i.e., $\{-g_{n-1}, \cdots, -g_{n-u}, \cdots, -g_{n-2u+1}\}$. 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 use 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 3). 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 1 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 proposed computational strategy 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 also be seen in the next section.

(b) Computation of Each $T_k^{(j)}$. The computation of each $T_k^{(j)}$ follows the regular calculation process, i.e., transform 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 1 is the serial accumulation of $T_k^{(j)}$). For example, (8) can be calculated as

$$[T_0^{(0)}] = \begin{bmatrix} g_0 d_0 - g_{n-1} d_1 - \cdots - g_{N-u+1} d_{u-1} \\ g_1 d_0 + g_0 d_1 - \cdots - g_{N-u+2} d_{u-1} \\ \cdots \quad \cdots \quad \cdots \\ g_{u-1} d_0 + g_{u-2} d_1 + \cdots + g_0 d_{u-1} \end{bmatrix}, \tag{14}$$

which applies to other $T_k^{(j)}$ of Algorithm 1.

4 SMALL: Proposed Accelerator

The overall structure of the proposed accelerator (SMALL) is shown in Fig. 2, where it consists of five major components, namely the input processing component, the sign processing component, the main computation component, the control generating component, and the output delivering component. In terms of the constitution of each component, there are: (i) two CSRs in the input processing component; (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; (iv) a control unit in the control generating component; and (v) a parallel-in serial-out (PISO) shift-register (SR) in the output delivering component. Note that the bit-width of the each data path depends on the setup in the specific PQC scheme.

The input processing component (two CSRs) is firstly loaded with the necessary coefficients from the two inputs (which takes N cycles) and then in the following cycles it produces the correct outputs to the following components. Connecting with Algorithm 1, 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

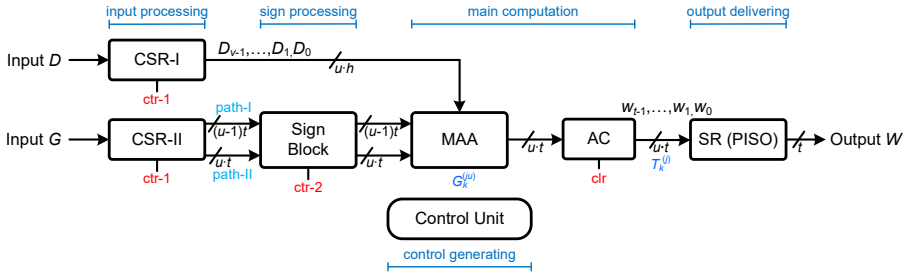

Fig. 2: The proposed polynomial multiplication accelerator (SMALL).

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, the final output delivering component transfers the parallel output into serial style to be stored in the external memory or for other usage. The overall operation, of course, is carried out through different types of control signals generated from the control unit (see control generating component for more details). The whole process, including the input loading and output delivering time, requires $(n + v^2 + u)$ cycles of operations. The detailed internal structures as well as related functions are presented below.

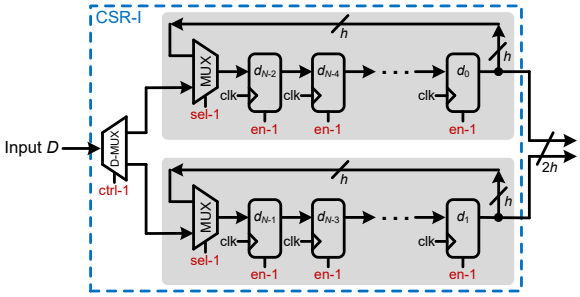

Fig. 3: The CSR-I (for $u = 2$), where the values in the registers are initial values.

The Input Processing Component. As seen from Fig. 2, 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 after all the input coefficients are loaded in the CSR-I). To realize this specific function, we have used a multi-path based CSR, as shown in Fig. 3, where we have presented a case study example when $u = 2$. This multi-path based CSR consists of two sub-CSRs (each sub-CSR has $n/2$ registers), where the input to each sub-CSR is directly from 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 loaded into the

corresponding registers, the two MUXes in the sub-CSRs 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. 3 can be extended to other u.

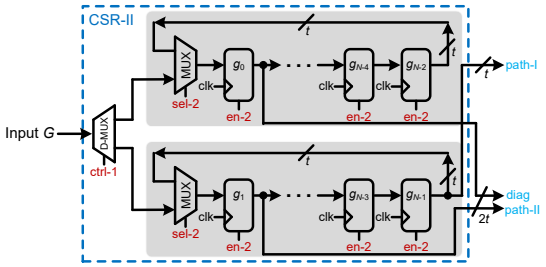

Fig. 4: The CSR-II ($u = 2$), where the values in the registers are initial values. Note "diag" refers to the value in the diagonal direction of the matrix, and is part of the path-II output.

The CSR-II has a similar design structure as that in Fig. 3 except the output setup. When connecting with the actual values contained in the two regions of Fig. 1 (applies to other matrices also), 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.

Considering the values contained within each $[G_k^{(ju)}]$, e.g., $[G_0^{(0)}]$ (for $[G_0]$, see (8)), 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. 4, 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 the 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 (13)). 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 disabled 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 (13)). 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 produced).

In a more general sense u is selected 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 (can be extended to other u).

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. 5, 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 (SIs) according to the two's complement representation requirement that all the bits

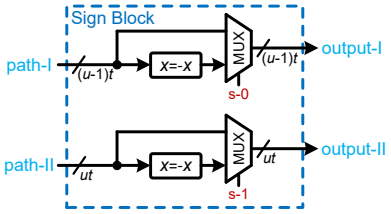

Fig. 5: The internal structure of the sign block.

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'). Note s-0 and s-1 are generated by the control unit.

The Main Computation Component. In this component, the MAA cell is responsible for the calculation of corresponding $T_k^{(j)}$ in Step 2.7 of Algorithm 1, while the AC cell executes the following accumulation in the same step. As specified in (14), the MAA directly obtains the output from standard matrix-vector based calculation, which applies to other values of u (see Fig. 6).

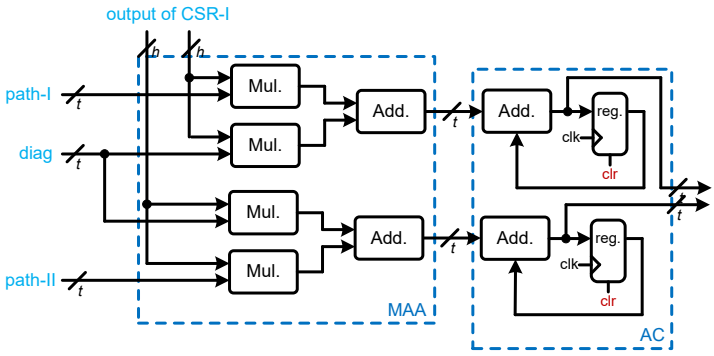

Fig. 6: The main computation component (for $u = 2$), where Mul. and Add. denote the multiplier and adder, respectively (reg. is the register).

As seen from Fig. 6, the MAA cell mainly consists of necessary multipliers and adders to perform the matrix-vector product (connecting (14)). 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. 6, 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 (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. 6 can easily extended to the design of other u.

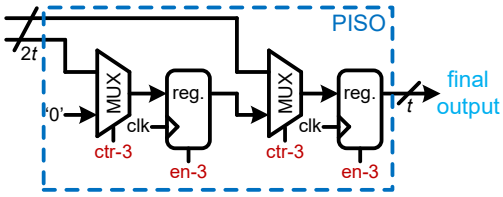

Fig. 7: The internal structure of the PISO SR for $u = 2$.

The Output Delivering Component. This component is relatively simple, and only a PISO SR is used to transfer the parallel output from the AC cell into a serial format for further processing (which is very important in practical applications). Fig. 7 gives an example for such SR when $u = 2$, which can be extended to other u. One can see that the two MUXes function to load the output from the AC cell into the registers in the SR for serial output delivery.

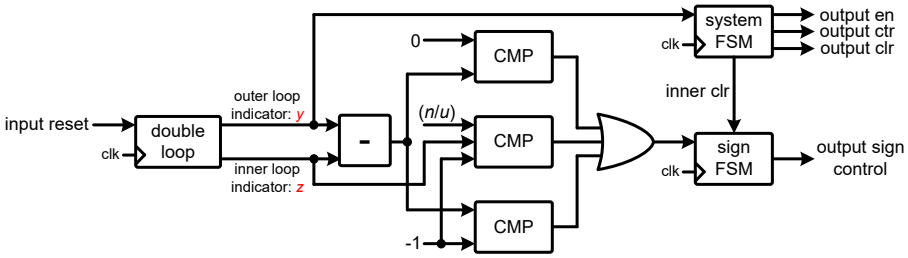

Fig. 8: The control unit, where y (horizontal) and z (vertical) are the indices for the elements within matrix $[G_k^{ju}]$. CMP: comparator. FSM: finite state machine.

The Control Generating Component. This component plays a key role in the whole accelerator, i.e., generating the sign controlling signals (sign block), clearing signals (mainly for the registers), enabling 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. 8, 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 (including the final output delivering) stages.

During the load stage, the control unit takes n cycles to serially receive all the coefficients of G and D into the corresponding registers in the CSRs. Note the control signal (ctr-1, see Fig. 2) 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}).

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. 2 and Fig. 5. 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 1) 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. 8, 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). 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 usage of the entire control unit.

5 Evaluation: Implementation and Comparison

This section focuses on the thorough complexity analysis and comparison of the proposed accelerator along with the state-of-the-art designs.

Complexity Analysis. In general, the proposed polynomial multiplication accelerator (SMALL), as shown in Fig. 2, uses two CSRs (where each CSR has n registers and $(v+1)$ MUXes/D-MUXes), one sign block ($(2u-1)$ MUXes and $(2u-1)$ SIs), one MAA cell (u^2 number of multipliers and $(u-1)u$ adders), one AC cell (u number of adders and the same number of registers), one PISO SR (u number of MUXes and the same number of registers), and one control unit. The polynomial multiplication structure requires n cycles of input loading into the related CSRs, $(n/u)^2$ cycles of calculation, and additional u cycles of output delivering (the first $(v-1)$ groups of outputs, from W_0 to W_{v-2}, are delivered out during the computation time). The actual critical-path of the accelerator is mainly determined by the selection of u and n.

Case Study Examples. For a detailed evaluation, we have used the polynomial multiplication used in NIST PQC third-round standardization candidate Saber [1] and RBLWE-ENC [2,4], respectively. Specifically, we notice that: (i) For

Table 1: Comparison of the Complexities of The Proposed Polynomial Multiplication Accelerator for Saber (FPGA Platform)

design	ALMs	Fmax (MHz)	latency	T[1] (μs)	ADP[2]
Stratix V device $n = 256$ (Proposed Design)					
$u = 1$	151	353	65,536	186	28,025
$u = 2$	340	240	16,384	68	23,211
$u = 4$	969	181	4,096	23	21,937
$u = 8$	2,766	142	1,024	7	19,966
Cyclone V device $n = 256$ (Proposed Design)					
$u = 1$	149	137	65,536	478	71,245
$u = 2$	403	109	16,384	151	60,732
$u = 4$	929	83	4,096	49	45,812
$u = 8$	2,908	64	1,024	16	46,311

[1]: T = total execution time ((latency cycle)$\times$(1/Fmax)). [2]: ADP=#ALM$\times$ T.

Saber, one polynomial has coefficients of either 13-bit or 10-bit (the 13-bit design covers the 10-bit one), while another polynomial has coefficients of [-4,4] (4-bit). The polynomial size n is fixed at 256. (ii) For RBLWE-ENC, one polynomial consists of integer coefficients of $\log_2 256 = 8$-bit. While another polynomial involves merely binary coefficients of $\in \{0, 1\}$. The polynomial size can be $n = 256$ and $n = 512$,

Experimental Setup. The experimental setup of our evaluation is set as follows: (a) we have coded the proposed polynomial multiplication accelerator (SMALL), based on Saber and RBLWE-ENC's respective parameters, with VHDL and have verified its functionality through ModelSim ($t = 13$, $h = 4$, and $n = 256$ for Saber and $t = 8$, $h = 1$, $n = 256$ and $n = 512$ for RBLWE-ENC, respectively, as well as $u = 1$, $u = 2$, $u = 3$, and $u = 4$); (b) we have synthesized and implemented the coded designs on the FPGA devices (both AMD-Xilinx and Intel-Altera FPGAs) to obtain their detailed area-time complexities along with the competing designs; (c) more detailed, we used the Intel Quartus Prime 17.0 to obtain the performance for all the coded designs on the Stratix V 5SGXMABN1F45C2 and/or Cyclone V 5CSXFC6D6F31I7ES devices; (d) we have also obtained the corresponding implementation results for Saber on the Artix-7 XC7A12TLCSG325-2L device through Xilinx Vivado 2019.2; (e) for the accelerator deployed with Saber's parameter, the coefficients of D are represented in the sign magnitude binary numbers (following [3]), while the coefficients in the design for RBLWE-ENC are denoted by the 2's complement.

FPGA based Implementation Results & Comparison. The obtained area-time complexities, in terms of the number of adaptive logic modules (ALMs) (or slice LUT/FF), maximum frequency (Fmax), latency cycles, total execution time T = ((latency cycle)$\times$(1/Fmax)), and area-delay product (ADP), of the proposed designs with different parameter settings are shown in Tables 2, and 3

Table 2: Comparison of the Complexities with the Existing Polynomial Multiplication Accelerators for Saber

design	LUT	FF	Fmax	latency	ADP[1] (μs)
Xilinx Artix-7 XC7A12TLCSG325-2L ($n = 256$)					
[3]	541	301	100	19,471	105,338
[25][2]	561	302	130	16,384	270,336
TW ($u = 1$)	202	605	178	65,536	74,372
TW ($u = 2$)	318	670	125	16,384	41,681
TW ($u = 4$)	664	798	109	4,096	24,952
TW ($u = 8$)	2,424	1,069	102	1,024	24,335

TW: This Work. Unit for Fmax: MHz.
[1]: ADP=#LUT×latency time.
[2]: This design also uses [25] uses 2 DSPs and 2 BRAMs (which need to be transferred into equivalent LUT usage, i.e., 1 BRAM (8k) equals 70 slices, 1 DSP equals 128 slices, and 1 slice has 4 LUTs).

along with those of the existing designs of [3,25], respectively. Note in Table 2, we have used the equivalent LUT calculation method to obtain the overall ADP for [25], for a fair comparison. Meanwhile, [3] indicated that external memory is used during the computation process, yet we don't count it here.

As seen from Tables 1 and 2, the proposed accelerator for Saber has superior performance on both Intel-Altera and AMD-Xilinx FPGA devices, not only limited to its scalability (but also the overall complexity). For instance, the proposed design ($u = 4$) has at least 76.3% less ADP than the recent one of [3] (also at least 90.8% less ADP than another recent report of [25]), not even counting the conditions that the existing design of [3] very much relies on the memory access (this part of resource is not included in Table 2). Meanwhile, one can notice that the proposed design overall maintains very low complexity, especially the ones of $u = 1, 2, 4$, which are desirable for lightweight applications.

For completeness, we highlight relevant NTT-based solutions. For instance, Saber work from [24] requires ∼1,680 clock cycles with 2,247 LUTs (with higher clock frequency) while our work has a latency of 1,024 clock cycles and 2,424 LUTs for $u = 8$. For RBLWE-ENC ($n = 512$) the work from [9] using t=8 and t=32 parallel processing units, has similar latency as this work ($u = 2$ and $u = 4$) but the ALMs are much higher, 5,073 and 6,076, respectively.

Meanwhile, as seen from Table 3, one can find that the proposed polynomial multiplication structure significantly outperforms the recent one in [10]. For example, the proposed design of $u = 4$ involves at least 98.7% less ADP than the one in [10] on the Stratix V device when $n = 256$ (the similar situation applies to nearly every case presented in Table 3). Meanwhile, considering that the existing one also requires external resource assistance and has a limited processing style, the proposed design has completely outperformed [10].

Discussion. The proposed algorithmic computation and accelerator design strategies overall are highly efficient: (i) the proposed scalable matrix originated

Table 3: Comparison of the Complexities of The Polynomial Multiplication Designs for RBLWE-ENC

design	ALMs	Fmax (MHz)	latency	T (μs)	ADP[1]
Stratix V device $n = 256$ (Existing Design)					
[10]	1,793	318.47	65,536	206	369,358
Stratix V device $n = 256$ (Proposed Design)					
$u = 1$	98	553	65,536	119	11,618
$u = 2$	179	421	16,384	39	6,959
$u = 4$	357	315	4,096	13	4,644
$u = 8$	914	202	1,024	5	4,624
Stratix V device $n = 512$ (Existing Design)					
[10]	3,491	288.77	262,144	908	3,169,828
Stratix V device $n = 512$ (Proposed Design)					
$u = 1$	104	562	262,144	466	48,501
$u = 2$	192	395	65,536	166	31,822
$u = 4$	430	290	16,384	56	24,294
$u = 8$	996	207	4,096	20	19,749

[1]: ADP=#ALM$\times$ T (total execution time).

strategy brings both flexibility and compactness to the polynomial multiplication's scalable processing and low complexity computation; (ii) the proposed novel algorithm-to-architecture design techniques have produced an exceptionally optimized accelerator (with resource usage significantly minimized).

The proposed design strategy can be further developed into other applications. For instance, we notice that the polynomial multiplication used in the homomorphic encryption scheme BFV [6] also involves a similar coefficients setup. Other research directions can also be further extending the proposed strategy into polynomial multiplication used for other PQC schemes.

6 Conclusion

In this paper, we propose a novel constant-time scalable matrix originated computation strategy for the efficient implementation of the targeted polynomial multiplication in important PQC schemes. In total, we have: (i) proposed a new polynomial multiplication algorithm; (ii) presented the details of the polynomial multiplication accelerator (SMALL); (iv) demonstrated the superior efficiency of the proposed polynomial multiplication accelerator through two case study examples. We hope the outcome of this work will produce significant impact on the PQC development and related computer arithmetic technique research.

Acknowledgement

J. Xie was supported by NIST-60NANB20D203 and NSF SaTC-2020625. Ç. Koç was supported by TUBITAK Project 1001-121F348.

References

1. Saber, https://www.esat.kuleuven.be/cosic/pqcrypto/saber/
2. Bao, T., He, P., Bai, S., Xie, J.: Tina: Tmvp-initiated novel accelerator for lightweight ring-lwe-based pqc. IEEE Transactions on Very Large Scale Integration (VLSI) Systems (01), 1–13 (2023). https://doi.org/10.1109/TVLSI.2023.3341037
3. Basso, A., Roy, S.S.: Optimized polynomial multiplier architectures for post-quantum kem saber. In: 2021 58th ACM/IEEE Design Automation Conference (DAC). pp. 1285–1290. IEEE (2021). https://doi.org/10.1109/DAC18074.2021.9586219
4. Buchmann, J., Göpfert, F., Güneysu, T., Oder, T., Pöppelmann, T.: High-performance and lightweight lattice-based public-key encryption. In: Proceedings of the 2nd ACM international workshop on IoT privacy, trust, and security. pp. 2–9 (2016). https://doi.org/10.1145/2899007.2899011
5. Choi, P., Kim, D.K.: Lightweight polynomial multiplication accelerator for ntru using shared sram. IEEE Transactions on Circuits and Systems II: Express Briefs **70**(12), 4574–4578 (2023). https://doi.org/10.1109/TCSII.2023.3290192
6. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive (2012)
7. Fawzi, A., Balog, M., Huang, A., Hubert, T., Romera-Paredes, B., Barekatain, M., Novikov, A., Ruiz, F., Schrittwieser, J., Swirszcz, G., Silver, D., Hassabis, D., Kohli, P.: Discovering faster matrix multiplication algorithms with reinforcement learning. Nature **610**, 47–53 (10 2022). https://doi.org/10.1038/s41586-022-05172-4
8. Han, J., Fan, H.: Toeplitz matrix-vector product based $GF(2^n)$ shifted polynomial basis multipliers for all irreducible pentanomials. Cryptology ePrint Archive, Paper 2013/427 (2013), `https://eprint.iacr.org/2013/427`, `https://eprint.iacr.org/2013/427`
9. He, P., Bao, T., Xie, J., Amin, M.: Fpga implementation of compact hardware accelerators for ring-binary-lwe-based post-quantum cryptography. ACM Trans. Reconfigurable Technol. Syst. **16**(3) (jun 2023). https://doi.org/10.1145/3569457, `https://doi.org/10.1145/3569457`
10. He, P., Guin, U., Xie, J.: Novel low-complexity polynomial multiplication over hybrid fields for efficient implementation of binary ring-lwe post-quantum cryptography. IEEE Journal on Emerging and Selected Topics in Circuits and Systems **11**(2), 383–394 (2021). https://doi.org/10.1109/JETCAS.2021.3075456
11. He, P., Tu, Y., Xie, J., Jacinto, H.: Kina: Karatsuba initiated novel accelerator for ring-binary-lwe (rblwe)-based post-quantum cryptography. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **31**(10), 1551–1564 (2023). https://doi.org/10.1109/TVLSI.2023.3302289
12. Hu, J., Wang, W., Cheung, R.C., Wang, H.: Optimized polynomial multiplier over commutative rings on fpgas: A case study on bike. In: 2019 International Conference on Field-Programmable Technology (ICFPT). pp. 231–234 (2019). https://doi.org/10.1109/ICFPT47387.2019.00035
13. Khayyat, A., Manjikian, N.: Analysis of blocking and scheduling for fpga-based floating-point matrix multiplication analyse du blocage et de l'ordonnancement d'une multiplication matricielle à virgule flottante sur un fpga. Canadian Journal of Electrical and Computer Engineering **37**(2), 65–75 (2014). https://doi.org/10.1109/CJECE.2014.2317983

14. Lucas, B.J., Alwan, A., Murzello, M., Tu, Y., He, P., Schwartz, A.J., Guevara, D., Guin, U., Juretus, K., Xie, J.: Lightweight hardware implementation of binary Ring-LWE PQC accelerator. IEEE Computer Architecture Letters **21**(1), 17–20 (2022). https://doi.org/10.1109/LCA.2022.3160394
15. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Advances in Cryptology–EUROCRYPT 2010: 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30–June 3, 2010. Proceedings 29. pp. 1–23. Springer (2010). https://doi.org/10.1145/2535925
16. Mouilleron, C.: Efficient computation with structured matrices and arithmetic expressions (11 2011)
17. Pollard, J.M.: The fast fourier transform in a finite field. Mathematics of computation **25**(114), 365–374 (1971)
18. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM (JACM) **56**(6), 1–40 (2009). https://doi.org/10.1145/1568318.1568324
19. Samsi, S., Helfer, B., Kepner, J., Reuther, A., Ricke, D.O.: A linear algebra approach to fast dna mixture analysis using gpus. In: 2017 IEEE High Performance Extreme Computing Conference (HPEC). pp. 1–6 (2017). https://doi.org/10.1109/HPEC.2017.8091027
20. Tan, W., Wang, A., Zhang, X., Lao, Y., Parhi, K.K.: High-speed vlsi architectures for modular polynomial multiplication via fast filtering and applications to lattice-based cryptography. IEEE Transactions on Computers **72**(09), 2454–2466 (2023). https://doi.org/10.1109/TC.2023.3251847
21. Wong, Z.Y., Wong, D.C.K., Lee, W.K., Mok, K.M., Yap, W.S., Khalid, A.: Karatsaber: New speed records for saber polynomial multiplication using efficient karatsuba fpga architecture. IEEE Transactions on Computers **72**(07), 1830–1842 (2023). https://doi.org/10.1109/TC.2023.3238129
22. Xie, J., He, P., Lee, C.Y.: Crop: Fpga implementation of high-performance polynomial multiplication in saber kem based on novel cyclic-row oriented processing strategy. In: 2021 IEEE 39th International Conference on Computer Design (ICCD). pp. 130–137 (2021). https://doi.org/10.1109/ICCD53106.2021.00031
23. Xie, J., Zhao, W., Lee, H., Roy, D.B., Zhang, X.: Hardware circuits and systems design for post-quantum cryptography–a tutorial brief. IEEE Transactions on Circuits and Systems II: Express Briefs **71**(3), 1670–1676 (2024). https://doi.org/10.1109/TCSII.2024.3357836
24. Xu, T., Cui, Y., Liu, D., Wang, C., Liu, W.: Lightweight and efficient hardware implementation for saber using ntt multiplication. In: 2022 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS). pp. 601–605 (2022). https://doi.org/10.1109/APCCAS55924.2022.10090310
25. Zhang, Y., Cui, Y., Ni, Z., Liu, D., Liu, W., et al.: A lightweight and efficient schoolbook polynomial multiplier for saber. In: 2022 IEEE International Symposium on Circuits and Systems (ISCAS). pp. 2251–2255. IEEE (2022). https://doi.org/10.1109/ISCAS48785.2022.9937496