Novel Schoolbook-Originated Polynomial Multiplication Accelerators for NTRU-based PQC

Yazheng Tu[1], Shi Bai[2], Jinjun Xiong[3], and Jiafeng Xie[1]
[1]: Department of Electrical and Computer Engineering, Villanova University, Villanova, PA, 19085
Email: {ytu1, jiafeng.xie}@villanova.edu
[2]: Department of Mathematics and Statistics, Florida Atlantic University, Boca Raton, FL, 33431
Email: sbai@fau.edu
[3]: Department of Computer Science and Engineering, University at Buffalo, NY 14260
Email: jinjun@buffalo.edu

Abstract—**NTRU (Number Theory Research Unit)-based post-quantum cryptography (PQC) has recently drawn significant attention from the research communities, e.g., the National Institute of Standards and Technology (NIST) PQC standardization process selected algorithm FALCON. As the recent trend in the field has gradually switched to the hardware implementation side, efficient accelerator design for polynomial multiplication (one of the critical components) within NTRU-based PQC is becoming ever more important. Traditionally, the commonly used method for NTRU-based PQC, e.g., FALCON, is the number theoretic transform (NTT). In this paper, however, we have presented an alternative method, i.e., novel SChoolbook-Originated Polynomial multiplication accElerators (SCOPE) design framework. Overall, we have proposed the schoolbook-based method in an innovative format to implement the targeted polynomial multiplication, first through a basic version and then through a Toeplitz matrix-vector product (TMVP)-based approach. In total, we have carried out four layers of coherent & interdependent efforts: (i) we have proposed a novel lookup table (LUT)-based point-wise multiplier along with a related modular reduction technique to obtain optimal implementation; (ii) we have then introduced a new hardware accelerator architecture for the targeted polynomial multiplication, deploying the proposed point-wise multiplier design; (iii) we have also extended the proposed architecture to a TMVP-based polynomial multiplication accelerator; (iv) thorough implementation and comparison have shown the efficiency of the proposed accelerators. e.g., they have even better area-time complexities than the existing NTT-based designs (for $n = 512$). The proposed design strategy is also extended to another NTRU-based scheme (the NIST third-round PQC standardization candidate, NTRU) and other schoolbook- and Toom-Cook-based polynomial multiplications used in other PQC schemes, and the same superior performance is again obtained. We hope the outcome of this research can impact the ongoing NIST PQC standardization process and related full-hardware implementation work for schemes like FALCON.**

Index Terms—**FALCON, FPGA, hardware implementation, NTRU, NTRU-based PQC, SChoolbook-Originated Polynomial multiplication accElerators (SCOPE), TMVP.**

I. INTRODUCTION

THE recent rapid progress in quantum computing has initiated many innovations related to post-quantum cryptography (PQC) [1]. Currently-deployed public-key cryptographic algorithms such as Rivest-Shamir-Adleman encryption (RSA) and Elliptic Curve Cryptography (ECC) are vulnerable to quantum attacks [2], [3]. Several computational problems which are believed to be quantum-resistant have been proposed, including the lattice-based and code-based assumptions [4], [5]. The NTRU (Number Theory Research Unit) problem can be phrased as an lattice-based computational problem that admits additional algebraic structure [6]. Notably, the NTRU assumption has been used in the recent selected algorithm FALCON [7] in the National Institute of Standards and Technology (NIST)'s PQC standardization process. The NIST third-round candidate, NTRU [8], also falls within this category.

Problem Background. Given the NIST-standardized algorithms for PQC, there is an increasing demand for investigating their efficient hardware implementation. In particular, efficient acceleration of critical components like polynomial multiplication has become an important research topic [9]–[17].

Related works [18], [19] on polynomial multiplication have been done for other lattice-based PQC schemes such as Kyber and Dilithium [20]. However, there is limited research on the hardware implementation in the literature for NTRU-based PQC (e.g., FALCON [7]). One plausible explanation is that the schemes Kyber and Dilithium convert their representations into the NTT (number theoretic transform) domain and hence the number of forward/inverse NTT is limited. By comparison, the polynomial multiplication in FALCON (for instance, in signature verification phase) requires the complete computation process (both NTT and inverse NTT (INTT) are needed in the same process) [21]. This requires a unified hardware platform that supports the NTT/INTT computations simultaneously which, to the best of our knowledge, is not fully investigated and remains as an open research question. Furthermore, an early work [22] has shown that the schoolbook-based approach can obtain better area-time complexities than the NTT-based designs (for general Ring-Learning-with-Errors (Ring-LWE)-based schemes). This finding has been further confirmed in a recent work [11] showing that the schoolbook-based polynomial multiplication can achieve high-performance operation while maintaining decent area-time complexities.

Motivation and Objective. In this paper, we follow the above-mentioned works' pace to explore the efficient implementation of polynomial multiplication accelerator for NTRU-based PQC. In particular, we use the polynomial multiplication of FALCON as a case study example since: (i) it requires a complete computation process even when the fast algorithm NTT is deployed (signature verification phase); (ii) not many works have been carried out specifically on this scheme. Meanwhile, we also want to explore if the schoolbook-based method is also efficient for FALCON, thus providing a better design option for the PQC implementation community.

Challenges. With these mentioned motivations, the proposed work aims to deliver a novel **SC**hoolbook-**O**riginated **P**olynomial multiplication acc**E**lerators (SCOPE) design framework for NTRU-based PQC. On the other hand, however, there are many bottleneck challenges for schoolbook-based polynomial multiplication (of FALCON). In particular, the schoolbook-based method naturally involves a computational complexity of $\mathcal{O}(n^2)$, and thus it is hard to compete with NTT-based design (complexity of $\mathcal{O}(n\log n)$) unless an innovative method is developed. Key challenges include: (a) the point-wise multiplier is large, and new method is needed for efficient implementation; (b) the modulus q is prime (modulo reduction is not free) that increases the implementation complexity; (c) new accelerator architecture is needed to obtain optimal design; (d) additional fast algorithm based design can be explored to achieve better performance.

Major Contributions. In this paper, we have carried out four layers of coherent interdependent efforts to obtain the proposed SCOPE. Major contributions include:

First, we have proposed a novel lookup table (LUT)-based point-wise multiplier along with an efficient modular reduction technique in which resource usage is reduced to the minimum.

Then, we introduced the proposed polynomial multiplication hardware accelerator by innovatively incorporating the developed point-wise multiplier to obtain optimal efficiency.

Besides, we have extended the proposed design to a Toeplitz matrix-vector product (TMVP)-based accelerator, with innovations from both algorithmic and architectural aspects.

Finally, we have conducted a thorough evaluation process to showcase the efficiency of the proposed accelerators over the existing designs, including schoolbook- and NTT-based ones.

Note that we have also applied the same techniques on the polynomial multiplication for NTRU [8], another NTRU-based NIST PQC standardization third-round candidate and obtained similar efficiency over the competing designs.

The rest of the paper is organized as follows. Section II gives the preliminaries. The proposed point-wise multiplier (along with the modular reduction technique) is presented in Section III. The first accelerator is then introduced in Section IV. The extended TMVP-based accelerator is presented in Section V. The evaluation process is provided in Section VI, and the conclusion is given in Section VII.

II. PRELIMINARIES

Notations. We use the following notations throughout the entire paper: (i) n is the size of the polynomial; (ii) the targeted FALCON polynomial multiplication relies on the operations over ring $\mathbb{Z}_q/(x^n+1)$ where n is a power-of-two, while the NIST PQC third-round candidate NTRU uses the operations over ring $\mathbb{Z}_q/(x^n-1)$ where n is a prime; (iii) q is the modulus; (iv) one input polynomial is denoted as G ($G = \sum_{i=0}^{n-1} g_i x^i$), another input is represented as $D = \sum_{i=0}^{n-1} d_i x^i$, and the output polynomial is $W = \sum_{i=0}^{n-1} w_i x^i$ (g_i, d_i, and w_i are coefficients). Other specific notations will be marked out in specific sections of the paper.

NTRU-based PQC. As one important branch of lattice-based cryptography, NTRU-based PQC can be traced back to 1996 [23]. The NTRU problem with standard parameters remains essentially unbroken after several decades of cryptanalysis. Although there exist other NTRU-based PQC schemes, in this paper, we mainly focus on the NIST-selected one, FALCON [7] (as well as the third-round PQC standardization candidate NTRU [8]). While FALCON stands for **Fa**st Fourier **l**attice-based **co**mpact signatures over **N**TRU, it is a lattice-based post-quantum signature scheme built on previous years' efforts [7]; NTRU is a merger of previous two algorithms NTRUEncrypt and NTRU-HRSS-KEM, and was selected as one of the NIST PQC standardization thrid-round candidates [8], [24].

Schoolbook-based Polynomial Multiplication. Without loss of generality, we can use the polynomial multiplication for FALCON (e.g., signature verification phase) to have

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

where $f(x) = x^n + 1$, and $G = \sum_{i=0}^{n-1} g_i x^i$, $D = \sum_{i=0}^{n-1} d_i x^i$, and $W = \sum_{i=0}^{n-1} w_i x^i$ (g_i, d_i, and w_i are coefficients of 14-bit over ring since modulus $q = 12,289$ for FALCON [7]).

Then, we can rewrite (1) into another form (note that $W = G\sum_{i=0}^{n-1} d_i x^i \bmod f(x)$)

$$W = \sum_{i=0}^{n-1} d_i (Gx^i \bmod f(x)), \tag{2}$$

where we can define $G^{[0]} = Gx^0 \bmod f(x) = G$, $G^{[1]} = Gx^1 \bmod f(x) = G^{[0]}x \bmod f(x)$, ..., and $G^{[n-1]} = Gx^{n-1} \bmod f(x) = G^{[n-2]}x \bmod f(x)$. And we can have

$$\begin{aligned} G^{[0]} =& g_0 + g_1 x + g_2 x^2 + \cdots + g_{n-1}x^{n-1}, \\ G^{[1]} =& -g_{n-1} + g_0 x + \cdots + g_{n-2}x^{n-1}, \\ \cdots& \cdots \cdots \\ G^{[n-1]} =& -g_1 - g_2 x - g_3 x^2 - \cdots + g_0 x^{n-1}, \end{aligned} \tag{3}$$

where $x^n \equiv -1$ is substituted (since $f(x) = x^n + 1 \equiv 0$).

Thus, (2) can be rewritten as

$$W = \sum_{i=0}^{n-1} d_i G^{[i]}, \tag{4}$$

which can be transferred into a matrix-vector product as

$$\begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_{n-1} \end{bmatrix} = \begin{bmatrix} g_0 & -g_{n-1} & \cdots & -g_1 \\ g_1 & g_0 & \cdots & -g_2 \\ \vdots & \vdots & \ddots & \vdots \\ g_{n-1} & g_{n-2} & \cdots & g_0 \end{bmatrix} \times \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{n-1} \end{bmatrix}, \tag{5}$$

which is equivalent to $[W] = [G] \times [D]$, where $[W]$, $[G]$, and $[D]$ are $n \times 1$, $n \times n$, and $n \times 1$ matrices, respectively.

Note that for the NIST PQC standardization thrid-round candidate NTRU, all the negative signs in (5) are not needed since its polynomial multiplication relies on the operations over ring $\mathbb{Z}_q/(x^n-1)$.

TMVP Method. TMVP is a subquadratic complexity method proposed for polynomial multiplication over $GF(2^m)$ (binary field) [25]. First of all, let us define an $n \times n$ Teoplitz matrix as $T = [t_{i,j}]_{0\leq i,j\leq n-1}$, where $t_{i,j} = t_{i-1,j-1}$ [26]. Meanwhile, we also have V_0 and V_1 as $\frac{n}{2} \times 1$ column vectors and T_0, T_1, and T_2 as $\frac{n}{2} \times \frac{n}{2}$ Toeplitz matrices ($V = (V_0, V_1)$ is an $n \times 1$ column vector). We thus have

$$Z = \begin{bmatrix} Z_0 \\ Z_1 \end{bmatrix} = \begin{bmatrix} T_0 & T_2 \\ T_1 & T_0 \end{bmatrix} \begin{bmatrix} V_0 \\ V_1 \end{bmatrix} = \begin{bmatrix} T_0(V_0+V_1) + (T_2+T_0)V_1 \\ T_0(V_0+V_1) + (T_1+T_0)V_0 \end{bmatrix}, \tag{6}$$

where one can see that the original computational complexity of $\mathcal{O}(n^2)$ is now reduced to three matrix-vector products of $\mathcal{O}((\frac{n}{2})^2)$. The format of (6) needs slight adjustment when extended for polynomial multiplication over integer field (such as the one shown in (5)).

Existing Works for Targeted NTRU-based PQC. The prior works for polynomial multiplication of the mentioned NTRU-based PQC are not many: (i) a hardware-implemented signature verification (mainly the NTT-based polynomial multiplication) for FALCON was presented in [27]; (ii) hardware-implemented polynomial multiplications for NIST third-round PQC candidate NTRU can be seen at [28], [10], and [9], respectively. Schoolbook- and Toom-Cook-based polynomial multiplication designs for other PQC are reported in [11], [12].

III. Proposed Novel Point-Wise Multiplier

As discussed in Section II, polynomial multiplication for FALCON requires a relatively complicated point-wise multiplier, i.e., two inputs of 14-bit and the output is then 28-bit (which needs an appropriate modular reduction to transfer the 28-bit output to 14-bit). Due to this setup, efficient implementation of the point-wise multiplier becomes crucial to the overall design efficiency of the polynomial multiplication accelerator. In this section, we have developed a novel point-wise multiplier combined with a modular reduction technique.

Consideration. Generally, large integer point-wise multiplier can be realized by DSP cores on the field-programmable gate array (FPGA), which is not efficient if standing on the point-view of equivalent Slice number [22]. In fact, since the targeted point-wise multiplier is integer-based, one can find an alternative method to implement it. First of all, let us consider

$$C = A \times B, \tag{7}$$

where A and B are 14-bit integers and C is a 28-bit integer. Define $B = \sum_{j=0}^{13} b_j 2^j$, where b_j is the actual value of each bit of B. One can then have

$$C = A \times \sum_{j=0}^{13} b_j 2^j = \sum_{j=0}^{13} (A \times 2^j) b_j, \tag{8}$$

where one can see that $(A \times 2^j)$ simply involves shifting, and the actual value of b_j ('0' or '1') determines the related $(A \times 2^j)$ needs to be accumulated or not.

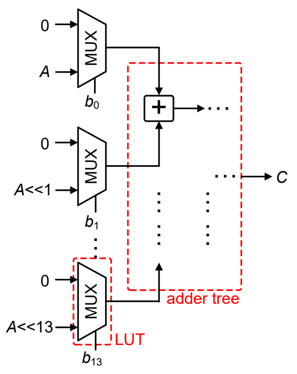

Fig. 1: Direct LUT-based point-wise multiplier based on (8).

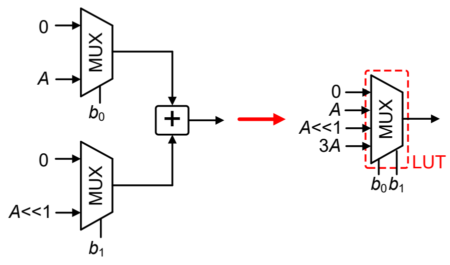

Fig. 2: Combined into a larger-size LUT.

Proposal. The above function of (8) can be seen as an equivalent function of adding 14 MUXes (or LUTs), as shown in Fig. 1, where b_j is used as the respective control signal. For practical implementation, however, this type of setup is not ideal as the overall structure involves too many MUXes and adders. To obtain optimal implementation, one can consider combining neighboring MUXes into a larger MUX (LUT). For instance, as shown in Fig. 2, two LUTs can be combined into one where one adder is saved at the cost of a larger-size LUT (assume $3A$ can be pre-computed). Following this strategy, the implementation of the point-wise multiplier becomes the finding of an optimal point of how large the combined LUT shall be and how many LUTs (and related types) are needed.

It is obvious that there is a trade-off between the size of the MUX and the calculation complexity of the summation. For example, if only single bits of B (b_j) are involved in the multiplication, then only 2-to-1 MUXes will be used. However, the maximum number of the MUXes, which is 14, will be used, and the following calculation will sum up 14 numbers shifted by 0, 1, 2, ..., and 13 positions, respectively. This summation is extremely resource-consuming. On the other hand, if all the bits of B are used as one selection signal together, a 2^{14}-to-1 MUX, which costs a huge area, will be used, and the calculation of the 2^{14} inputs of the MUX will be resource-intensive. Thus, based on the mentioned consideration, we propose to decompose B into four segments, which consist of 4, 4, 3, and 3 bits[1], respectively, starting from the least significant bit. This combination allows a decent amount of MUX input signal sharing. Meanwhile, involving 4 MUXes means that a 2-layer adder tree could implement the summation, which avoids a long critical-path.

[1]We obtain this segmentation through actual experimental testing.

Modular Reduction. Another important operation related to point-wise multiplication is modular reduction. Traditional multiplication executes the modular reduction at the end of the entire calculation. This setup, however, leads to a problem in hardware implementation, i.e., the length of the bit-shifted signals coming out from the MUXes are different, and we have to expand them all to 28 bits to execute the summation. Moreover, adding several 28-bit values can easily result in a long critical-path in the adder tree. In this case, we propose to move the modular reduction at the beginning of the multiplication, that is, right before they are fed to the MUXes. In this way, all the values attached to the MUXes are scaled back to 14 bits, more specifically, in the range of [0, q). Then, after the addition in the summation process, only a Simple Reduction is needed to keep the result in the desired range. Halving the length of the addends can significantly reduce the critical-path and enhance the point-wise multiplier's efficiency.

Algorithm 1: Proposed 14-bit point-wise multiplier with modular reduction (q = 12,289)

```
Input  : $A$ and $B$ are 14-bit integers;
Output: $C = AB \bmod q$; // where $C$ is also a 14-bit
        integer over $\mathbb{Z}_q$
Initialization step
1  Let $B = 2^{11} \cdot B_3 + 2^8 \cdot B_2 + 2^4 \cdot B_1 + 2^0 \cdot B_0$;
2  Let $P = [0, 4, 8, 11]$; // the elements in the array are
     the power of the coefficient in each segment of $B$
3  Define $a\_s[\cdot], \mathrm{sum}[\cdot]$;
Main step
4  for $i = 0$ to 3 do
5      $a\_s[i] = 2^{P[i]} \cdot A \bmod q$;
6  end
7  for $i = 0$ to 1 do
8      sum[$i$] = $a\_s[2i] \times B_{2i} + a\_s[2i+1] \times B_{2i+1}$;
9      if sum[$i$] $> q$ then
10         sum[$i$] = sum[$i$] - $q$;
11     end
12 end
13 $C$ = sum[0] + sum[1];
14 if $C > q$ then
15     $C = C - q$;
16 end
```

The proposed 14-bit point-wise multiplication algorithm is shown in Algorithm 1. As mentioned above, we split A into four parts, each with 4, 4, 3, and 3 bits, respectively. Then, we calculate the values of shifting B by those different bits and execute modular reductions. The reduced values are summed in two groups, followed by a Simple Reduction (subtracting q if the sum is larger than q). After summing up all the values, a final Simple Reduction is executed to make sure the result is scaled in the range of [0,q). Note the modular reduction used in the proposed algorithm is called Longa Reduction [29] (K-red, see Algorithm 2 of [29]).

Hardware Structure. Following Algorithm 1, we propose the point-wise multiplier as shown in Fig. 3. This point-wise multiplier consists of two 16-to-1 MUXes (4-input LUTs) and two 8-to-1 MUXes (3-input LUTs), one 2-layer adder tree, and three Simple Reduction units (attached to each adder in the adder tree). Meanwhile, different groups of shifted-and-reduced multiples (output of the Longa Reduction units), namely $\{0, a\_s[0], ..., 15 \cdot a\_s[0]\}$, $\{0, a\_s[1], ..., 15 \cdot a\_s[1]\}$, $\{0, a\_s[2], ..., 7 \cdot a\_s[2]\}$, and $\{0, a\_s[3], ..., 7 \cdot a\_s[3]\}$ are fed to the input of the four MUXes, with the selection being $b_3b_2b_1b_0$, $b_7b_6b_5b_4$, $b_{10}b_9b_8$, and $b_{13}b_{12}b_{11}$, respectively ($b_{13}b_{12}...b_0$ are the bits of B). The selection signals determine four correct multiples from the inputs as the output of the MUXes to be fed to the adder tree. The first layer of the adder tree performs two additions, i.e., two 8-to-1 MUXes and two 16-to-1 MUXes, corresponding to the sum of the upper/lower half of the inputs. A Simple Reduction unit is connected to the adders to reduce the sum to the regular range. Unlike the Longa Reduction, this Simple Reduction simply subtracts q from the sums if they are larger than q, otherwise keeps the sum unchanged (the selected multiples have been reduced to [0, q) and the sums cannot be larger than $2q$). The reduced sums are then added again, which requires another Simple Reduction to be scaled to the desired range.

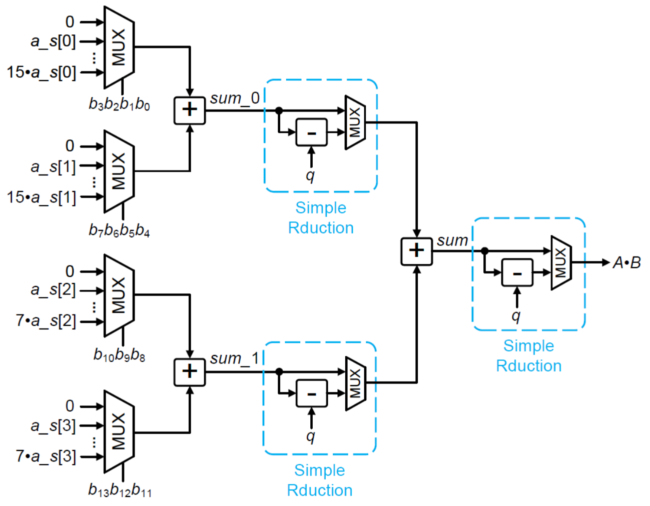

Fig. 3: Details of the proposed point-wise multiplier.

Algorithm 2: Longa Reduction on 28-bit value using K-red of [29] with $q = 12,289$, $k = 3$, and $m = 12$.

```
Input  : $C$ is a 28-bit integer;
Output: $k \cdot C \bmod q$;
1 Define sum[$\cdot$]; // where sum[$\cdot$] is the array of
    $kC_0 - C + iq$, $i = -3, -2, ..., 2$;
2 $C_0 = C \bmod 2^m$;
3 $C_1 = C \,//\, 2^m$;
4 $C_{red} = kC_0 - C$;
5 for $i = 0$ to 5 do
6     sum[$i$] $= kC_0 - C - (i-3)q$;
7 end
8 $k = arg_i$ sum[$i$] $\in [0, q)$;
9 Return sum[$k$]
```

Longa Reduction Unit (K-red). The details of the Longa

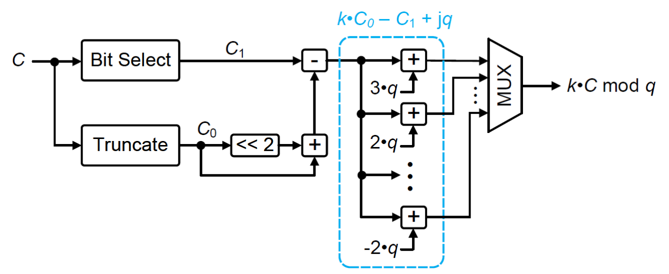

Fig. 4: The details of the Longa Reduction unit (K-red).

Reduction unit are described as follows. When executing a point-wise multiplication, the K-red unit is fed with a multiple of the shifted multiplicand, which is responsible for performing the Longa Reduction to scale the multiples of b_i back to the range of [0, q), following Algorithm 2, where $k = 3$ and $m = 12$. Note that the calculation of C_0 and C_1 can be implemented by bit truncation and selection. Also, three times of C_0 ($k = 3$) can be realized by $C_0 + 2 \cdot C_0$, which can be done by one-bit shifting and one addition. Therefore, the cost of the entire reduction operation is only two additions if we consider the subtraction as the addition based on two's complement representation. The result of the reduction operation falls into the range of (-3q, 3q), so a post-process of the reduction result is needed. we calculate different values of $k \cdot C_0 - C_1 + j \cdot q$, where $j = 3, 2, 1, 0, -1, -2$ and choose the one falling into [0, q) as the final output which will be delivered out. Note here the reduction unit returns the value of $k \cdot C$ mod q, so the inputs are assumed to be multiplied with the modular inverse of k prior to being input to the reduction unit (when they are generated/sampled) to return the value of C mod q. The overall structure can be seen in Fig. 4.

IV. SCOPE-I: The First Accelerator

Based on equations (4) and (5), we can have the first schoolbook-based polynomial multiplication algorithm, as shown in Algorithm 3. This algorithm calculates the product of one column of $[G]$ and one d_i at one time, and accumulates until all the products are obtained (time complexity of $\mathcal{O}(n)$).

Overview. Following Algorithm 3, we propose the hardware structure of the first accelerator as shown in Fig. 5. The proposed SCOPE-I contains five main components: Basic Input Process Component (BIPC), Basic Shift and Reduction Component (BSRC), Basic Point-wise Multiplier Component (BPMC), Basic ACcumulation Component (BACC), and Con-Trol Unit (CTU). Once all G coefficients are loaded into BIPC, they are fed in parallel into BPMC, along with the coefficients of D which are delivered serially from the input port. BPMC calculates the point-wise products between the G and D coefficients. These point-wise products are then sent to BACC to be accumulated to form the final output for serial delivering out. The entire calculation takes $(n + x)$ cycles, where x is the number of pipeline register layers inserted in/between components to increase the working frequency (see the last paragraph of this section).

Basic Input Process Component (BIPC). The BIPC uses a serial-in parallel-out (SIPO) shift register to receive and deliver the G coefficients. The SIPO contains n registers controlled by *en_sipo*. In load mode, BIPC takes one G input per cycle, feeding it to the first register while shifting previous values. In output mode, *en_sipo* stops shifting by disabling the registers, so the SIPO outputs the same values each cycle. This enables parallel delivery of the loaded G coefficients.

Basic Shift and Reduction Component (BSRC). The BSRC is responsible for pre-processing the input D by calculating multiples of the input and then shifting and performing the Longa reduction to the shifted multiples (Fig. 7). Having received an input coefficient of D (d_i), the BSRC firstly calculates its multiples up to 15 times of the input ($1 \cdot d_i$, $2 \cdot d_i$, ..., $15 \cdot d_i$), then the 15 multiples are left-shifted by 4 bits (d_i_s4, ..., $15 \cdot d_i\_s4$) while the 1 to 8-time multiples are shifted by 8 bits (d_i_s8, ..., $15 \cdot d_i\_s8$) and 11 bits (d_i_s11, ...,

Algorithm 3: Proposed first version of the schoolbook-based polynomial multiplication algorithm

```
Input  : $G$, $D$ are integer polynomials with 14-bit
         coefficients;
Output: $W = GD \bmod x^n + 1$ with $w_i \in \mathbb{Z}_q$.
Initialization step
1 Serially input $g_i$;
Main step
2 for $i = 0$ to $n-1$ do
3     for $j = 0$ to $n-1$ do
4         $w_i = w_i + G_i^{[j]} d_j$ mod $q$; // where $G_i^{[j]}$
          denotes the $i$-th coefficient of $G^{[j]}$;
5     end
6 end
Final step
7 Serially deliver the coefficients of output $W$;
```

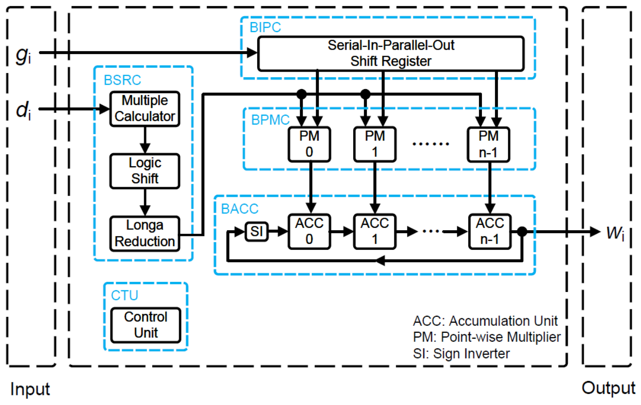

Fig. 5: Overview of the first accelerator (SCOPE-I).

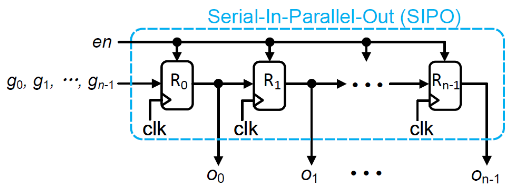

Fig. 6: Details of the BIPC ($o_0, ..., o_{n-1}$ are output bits).

TABLE I: Accumulation Process of The Proposed First Accelerator (SCOPE-I) with $n = 4$

Cycle	d_i	R_0	R_1	R_2	R_3
0	d_3	g_0d_3	g_1d_3	g_2d_3	g_3d_3
1	d_2	$g_0d_2 - g_3d_3$	$g_1d_2 + g_0d_3$	$g_2d_2 + g_1d_3$	$g_3d_2 + g_2d_3$
2	d_1	$g_0d_1 - g_3d_2 - g_2d_3$	$g_1d_1 + g_0d_2 - g_3d_3$	$g_2d_1 + g_1d_2 + g_0d_3$	$g_3d_1 + g_2d_2 + g_1d_3$
3	d_0	$g_0d_0 - g_3d_1 - g_2d_2 - g_1d_3$	$g_1d_0 + g_0d_1 - g_3d_2 - g_2d_3$	$g_2d_0 + g_1d_1 + g_0d_2 - g_3d_3$	$g_3d_0 + g_2d_1 + g_1d_2 + g_0d_3$

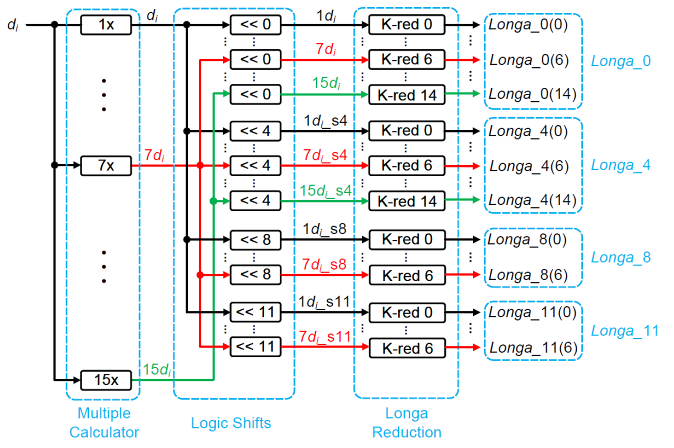

Fig. 7: Details of the BSRC.

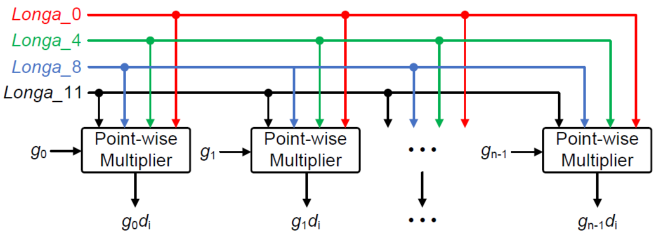

Fig. 8: Details of the BPMC.

$15 \cdot d_i\_s11$), respectively. Each set, which are called Longa_0, Longa_4, Longa_8, and Longa_11, respectively, of shifted multiples is then fed into 7 or 15 parallel Longa Reduction units (K-red) that they are reduced back to the range of [0, q).

Basic Point-wise Multiplication Component (BPMC). The BPMC calculates the multiplication between the input d_i and n coefficients in G in parallel and delivers the products to the BACC. This component contains n LUT-based point-wise multipliers. During the calculation, the BPMC takes in the shifted-and-reduced multiples d_i (Longa_0, Longa_4, Longa_8, Longa_11) from the Longa Reduction unit (K-red) and n parallel output of G from BIPC and feeds the coefficients into the point-wise coefficients. Each point-wise multiplier takes one coefficient g_i and the shared shifted-and-reduced multiples of d_i as inputs, calculates the product of g_i and d_i, and delivers the products to the following BACC.

Basic ACcumulation Component (BACC). As shown in Fig. 9, the BACC contains n registers, n adders, n Simple Reductions, and a sign inverter (14 NOT gates and an adder, based on the two's complement). The adder (before each register) adds the product of the BPMC and the output from the previous register. The sum is fed to the Simple Reduction unit to be reduced back into the range [0, q) and then transferred to the next register. The structure of Fig. 9 accomplishes the

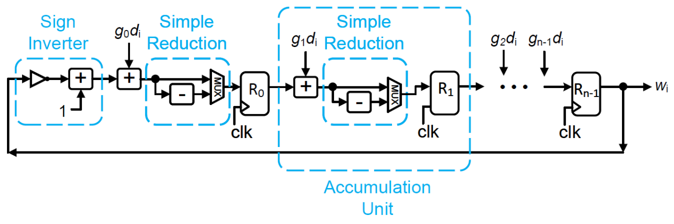

Fig. 9: Details of the BACC (R_0, R_1, ..., R_{n-1} are registers).

accumulation of all the desired products $(w_0, w_1, ..., w_{n-1})$ in n cycles. Table I shows the accumulation process of a simple example with $n = 4$. After the accumulation, the BACC serially outputs the products (note D is set to 0 at this time).

ConTrol Unit (CTU). The CTU, mainly a finite state machine (FSM), consists of five consecutive states: *reset*, *load*, *multi*, *output*, and *done*. In each state, different control signals related to the operation of the components are generated/adjusted, such as *csh_sipo* (input/output control signal), *en_sipo* (enable signal), *clr_reg* (clear signal), etc. The FSM enters the *reset* state when *clr* signal is received and it then enters the *load* state within one clock (n cycles). The *multi* state takes the FSM $(n + x)$ cycles to finish the calculation. After that, the FSM enters the *output* state that the output is serially delivered (n cycles). Finally, the FSM switches to the *done* state to complete the computation.

Register Insertion for Frequency Enhancement. Multiple layers of pipeline registers are inserted to enhance the working frequency of the proposed architecture. Although the register insertion slightly increases resource usage (and computation latency), the increased frequency can greatly improve the overall area-time complexities. Overall, we have inserted four layers of registers (through actual experimental testing), one layer after the MUXes in BPMC, one layer after both the first and second layers of adders in the adder trees of BPMC, and one layer before the Simple Reductions in BACC. Section VI demonstrates the efficiency of the proposed register insertion.

V. TMVP-based SCOPE: The Second Accelerator

Algorithm. By dividing $[G]$ following (6), we have

$$\begin{bmatrix} W_0 \\ W_1 \end{bmatrix} = \begin{bmatrix} G_0 & G_2 \\ G_1 & G_0 \end{bmatrix} \times \begin{bmatrix} D_0 \\ D_1 \end{bmatrix}, \tag{9}$$

where $[W_0]$, $[W_1]$, $[D_0]$, and $[D_1]$ are $\frac{n}{2} \times 1$ vectors; $[G_0]$, $[G_1]$, and $[G_2]$ are $\frac{n}{2} \times \frac{n}{2}$ matrices.

Note here $[G_2] = -[G_1]$, so by substitution we can have

$$\begin{aligned} \begin{bmatrix} W_0 \\ W_1 \end{bmatrix} &= \begin{bmatrix} G_0 & -G_1 \\ G_1 & G_0 \end{bmatrix} \times \begin{bmatrix} D_0 \\ D_1 \end{bmatrix} \\ &= \begin{bmatrix} G_0(D_0 + D_1) + (-G_1 - G_0)D_1 \\ G_0(D_0 + D_1) + (G_1 - G_0)D_0 \end{bmatrix}. \end{aligned} \tag{10}$$

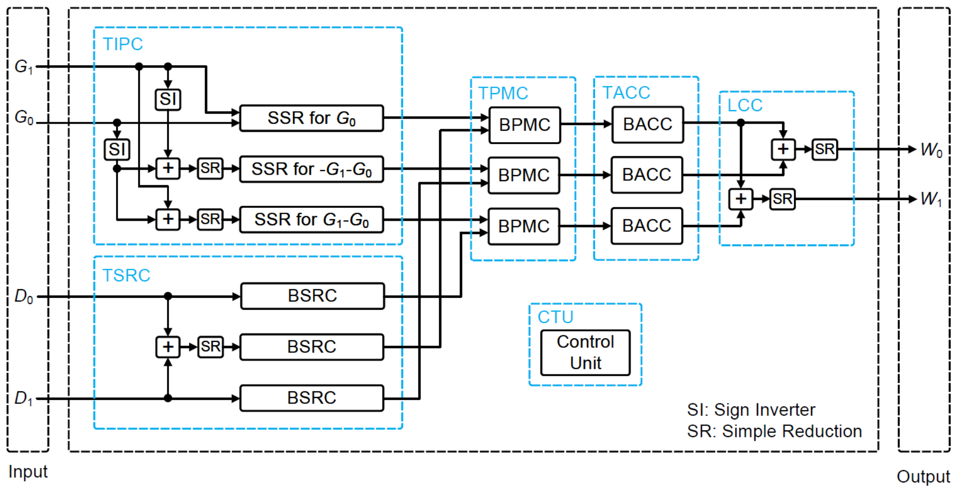

Fig. 10: Overview of the TMVP-based SCOPE-II.

Algorithm 4: Proposed TMVP-based algorithm

```
Input  : $G$, $D$ are integer polynomials with 14-bit coefficients;
Output: $W = GD \bmod x^n + 1$ with $w_i \in \mathbb{Z}_q$;
Main step
1 for $i = 1$ to $n/2$ do
2     for $j = 1$ to $n/2$ do
3         $w_0^j = [G_0^j]_i[D_0^j + D_1^j] + [-G_1^j - G_0^j]_i[D_1^j]$;
4         $[W_0{}^j] = [W_0{}^j] + w_0^j$;
5         $w_1^j = [G_0^j]_i[D_0^j + D_1^j] + [G_1^j - G_0^j]_i[D_0^j]$;
6         $[W_1^j] = [W_1^j] + w_1^j$.
7     end
8 end
Final step
9 Serially deliver the coefficients of output $W_0$ and $W_1$;
```

The proposed computation is thus shown in Algorithm 4. **Note**: $[G_0^i]_j$ denotes the element in the i-th row and the j-th column of $[G_0]$, $[W_0^j]$ denotes the j-th row of $[W_0]$, same notations apply for $[G_1]$ and $[W_1]$ ($[D_0], [D_1], [W_0]$, and $[W_1]$ have no subscript since they only have one column).

Overview. Accordingly, following the algorithmic operation of Algorithm 4, the proposed TMVP-based SCOPE-II involves six main components (Fig. 10): TMVP Input Process Component (TIPC), TMVP Shift and Reduction Component (TSRC), TMVP Point-wise Multiplier Component (TPMC), TMVP ACcumulation Component (TACC), ConTrol Unit (CTU), and Linear Combination Component (LCC). Having loaded all G coefficients into TIPC, TPMC then takes processed coefficients from G_0, G_1 and D_0, D_1 to calculate the needed point-wise products and accumulations. When the calculation finishes, the TACC outputs two sum-of-products (in parallel) to the LCC to produce the final output in serial. The major computation time is now only $n/2$ cycles (the register insertion can slightly increase the latency by a few cycles).

TMVP Input Process Component (TIPC). Due to the nature of the TMVP method, instead of outputting fixed coefficients of G ($g_0, g_1, ..., g_{n-1}$), TIPC for TVMP-based SCOPE provides dynamic linear combinations of coefficients in G_0 and G_1, corresponding to G_0, $(-G_1-G_0)$, and (G_1-G_0), respectively. As shown in Table II, outputs for G_0, $(-G_1-G_0)$, and $(G_1 - G_0)$ are different every cycle since coefficients in G_0 and G_1 change by time. To realize the changing format when outputting the coefficients during the calculation, two special Switch Shift Registers (SSRs) are designed (Fig. 11). The SSRs work in two operational modes, *load* and *output*, controlled by control signals *en_ctrl*, *csh*, and *switch*. When loading the coefficients ($[G_0^i]_0, [G_1^i]_0$, which are the coefficients in the first column of G_0 and G_1) from the input port of G_0 and G_1, the TIPC firstly loads $[G_0^i]_0$ into one set of the SSR. Meanwhile, the adders and sign inverter in TIPC calculate $-[G_1^i]_0 - [G_0^i]_0$ and $-[G_1^i]_0 + [G_0^i]_0$, and send the two values to the two SSRs in the other set. Having loaded all the coefficients (linear combinations), the TIPC starts to output the values to the TPMC during the multiplication process. During the calculation, one of the coefficients in each SSR is substituted with a new value at every clock cycle. For example, at the first cycle, all the coefficients are outputted in the order as they were loaded; at the second cycle, the last ($(n/2-1)$-th) coefficient in the SSR for G_0 is substituted with the last coefficient in G_1, while the last coefficient in the SSR for $(-[G_1]_0 - [G_0]_0)$ is substituted with $(-[G_1]_0 + [G_0]_0)$'s last coefficient $((-[G_1^{n/2-1}]_0 + [G_0^{n/2-1}]_0))$, so as the SSR for $(-[G_1]_0 + [G_0]_0)$. The coefficients in the two sets of SSRs will keep being substituted until all the last $(n/2 - 1)$ coefficients have been substituted in this manner. In TIPC, each SSR contains $n/2$ registers and $n/2$ 2-to-1 MUXes, where the MUXes choose either the output from the previous register or the value that is used for substitution. During the *load* mode, the TIPC loads and stores all the values for $(-[G_1]_0-[G_0]_0)$ and $(-[G_1]_0+[G_0]_0)$ chosen by the MUXes. During multiplication, the TIPC keeps receiving coefficients of $[G_1^j]_i$ and $[G_0^j]_i$, calculating linear combinations of the coefficients, and sending the results to the registers in parallel. The MUXes then choose the substituting values during the *ouput* mode. In the *load* mode, all the registers are enabled,

TABLE II: Theoretical Derivation and Formulation of Linear Combination of TIPC Outputs for Different Cycles

Index	$n/2-1$	$n/2-2$	...	i	...	1	0
Cycle 0							
G_0	$g_{n/2-1}$	$g_{n/2-2}$	...	g_i	...	g_1	g_0
G_1	g_{n-1}	g_{n-2}	...	$g_{n/2+i}$	...	$g_{n/2+1}$	$g_{n/2}$
$-G_1-G_0$	$-g_{n-1}-g_{n/2-1}$	$-g_{n-2}-g_{n/2-2}$	...	$-g_{n/2+i}-g_i$	...	$-g_{n/2+1}-g_1$	$-g_{n/2}-g_0$
G_1-G_0	$g_{n-1}-g_{n/2-1}$	$g_{n-2}-g_{n/2-2}$	...	$g_{n/2+i}-g_i$	...	$g_{n/2+1}-g_1$	$g_{n/2}-g_0$
Cycle 1							
G_0	$-g_{n-1}$	$g_{n/2-2}$	...	g_i	...	g_1	g_0
G_1	$g_{n/2-1}$	g_{n-2}	...	$g_{n/2+i}$	...	$g_{n/2+1}$	$g_{n/2}$
$-G_1-G_0$	$-g_{n/2-1}+g_{n-1}$	$-g_{n-2}-g_{n/2-2}$	...	$-g_{n/2+i}-g_i$	...	$-g_{n/2+1}-g_1$	$-g_{n/2}-g_0$
G_1-G_0	$g_{n/2-1}+g_{n-1}$	$g_{n-2}-g_{n/2-2}$	...	$g_{n/2+i}-g_i$	...	$g_{n/2+1}-g_1$	$g_{n/2}-g_0$
Cycle 2							
G_0	$-g_{n-1}$	$-g_{n-2}$	...	g_i	...	g_1	g_0
G_1	$g_{n/2-1}$	$g_{n/2-2}$	...	$g_{n/2+i}$	...	$g_{n/2+1}$	$g_{n/2}$
$-G_1-G_0$	$-g_{n/2-1}+g_{n-1}$	$-g_{n/2-2}+g_{n-2}$	...	$-g_{n/2+i}-g_i$	...	$-g_{n/2+1}-g_1$	$-g_{n/2}-g_0$
G_1-G_0	$g_{n/2-1}+g_{n-1}$	$g_{n/2-2}+g_{n-2}$	...	$g_{n/2+i}-g_i$	...	$g_{n/2+1}-g_1$	$g_{n/2}-g_0$
Cycle i							
G_0	$-g_{n-1}$	$-g_{n-2}$	...	$-g_{n/2+i}$	...	g_1	g_0
G_1	$g_{n/2-1}$	$g_{n/2-2}$	...	g_i	...	$g_{n/2+1}$	$g_{n/2}$
$-G_1-G_0$	$-g_{n/2-1}+g_{n-1}$	$-g_{n/2-2}+g_{n-2}$	...	$-g_i+g_{n/2+i}$	...	$-g_{n/2+1}-g_1$	$-g_{n/2}-g_0$
G_1-G_0	$g_{n/2-1}+g_{n-1}$	$g_{n/2-2}+g_{n-2}$	...	$-g_i-g_{n/2+i}$	...	$g_{n/2+1}-g_1$	$g_{n/2}-g_0$
...							
Cycle $n-1$							
G_0	$-g_{n-1}$	$-g_{n-2}$	...	$-g_{n/2+i}$	...	$-g_{n/2+1}$	g_0
G_1	$g_{n/2-1}$	$g_{n/2-2}$	...	g_i	...	g_1	$g_{n/2}$
$-G_1-G_0$	$-g_{n/2-1}+g_{n-1}$	$-g_{n/2-2}+g_{n-2}$	...	$-g_i+g_{n/2+i}$	...	$-g_1+g_{n/2+1}$	$-g_{n/2}-g_0$
G_1-G_0	$g_{n/2-1}+g_{n-1}$	$g_{n/2-2}+g_{n-2}$	...	$-g_i-g_{n/2+i}$	...	$-g_1-g_{n/2+1}$	$g_{n/2}-g_0$

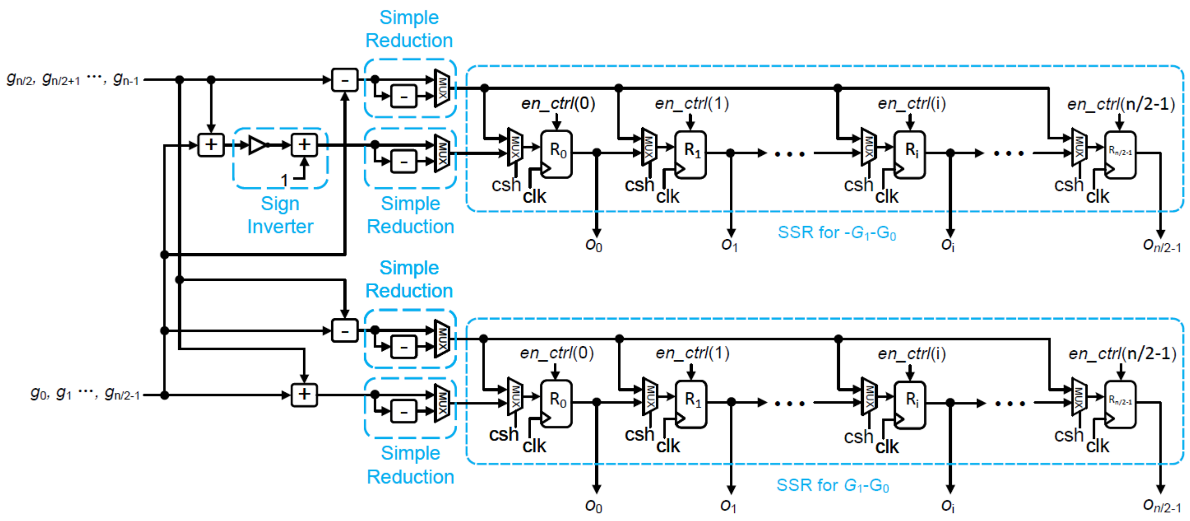

Fig. 11: Details of the SSRs for $(-G_1 - G_0)$ and $(G_1 - G_0)$ in TIPC for the second accelerator (SCOPE-II).

while only one register in each shift register is enabled to receive the coefficient to be substituted at each cycle.

TMVP Shift and Reduction Component (TSRC). As shown in Fig. 13, TSRC consists of three individual BSRCs, corresponding to the process of D_0, D_1, and $(D_0 + D_1)$, respectively. In particular, the TSRC calculates $(D_0 + D_1)$, followed by a Simple Reduction, and then sends the value to the third BSRC. When processing, the three BSRCs multiply, shift, and execute Longa Reduction on the input/calculated data simultaneously to produce the output data for the TPMC.

TMVP Point-wise Multiplication Component (TPMC). The TMPC also involves three BMPCs as basic units, responsible for the calculation of products of $[G_1 - G_0]_i[D_0^i]$, $[G_0]_i[D_{sum}^i]$, and $[-G_1 - G_0]_i[D_1^i]$ respectively, as shown in Fig. 14 (note we use D_{sum} to represent (D_0+D_1)). The three BMPCs receive the linear combinations of G_0 and G_1 output from TIPC as well as the Longa Reduction outputs from TSRC to calculate the point-wise products in one clock cycle. The products delivered from the BPMCs (grouped into three sets) are then fed to the TACC in parallel for accumulation.

TMVP ACcumulation Component (TACC). As shown in Fig. 15, the TACC consists of three sets of BACCs. Unlike the BACC in SCOPE-I (Fig. 5), the output of the last $((n/2-1)$-th) register will not be negated before being circularly shifted back to the adder before the first register (no need for any sign inverter). The three BACCs work at the same time, and the entire accumulation lasts $n/2$ cycles. When the accumulation is done, the BACCs output the accumulated sum-of-products in

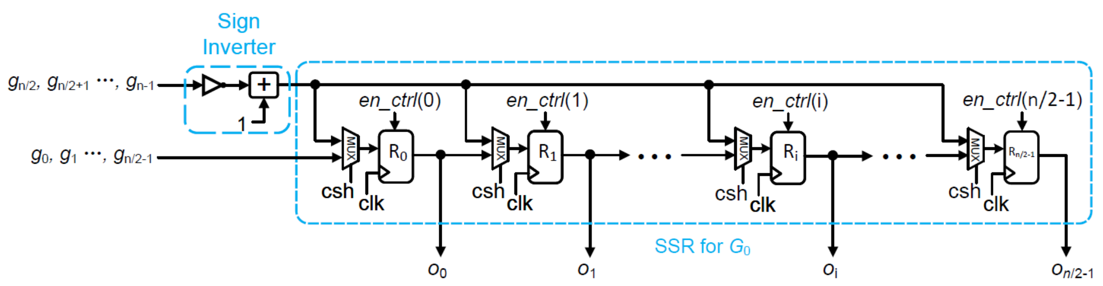

Fig. 12: Details of the SSR for G_0 in TIPC for the second accelerator (SCOPE-II).

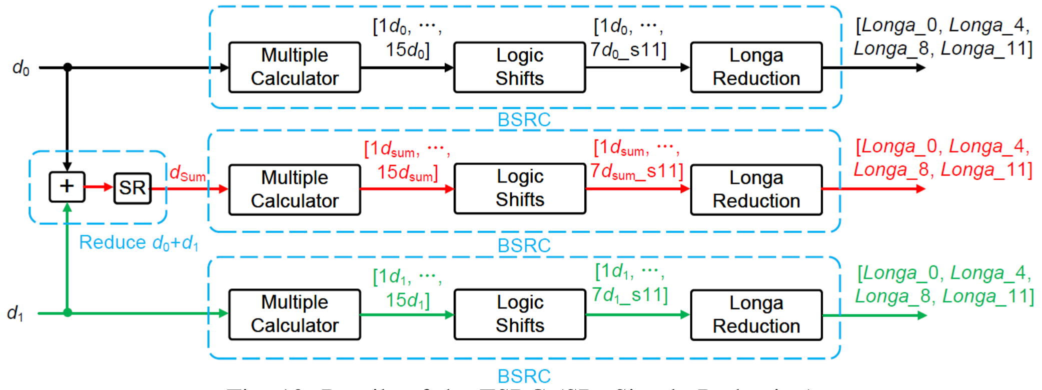

Fig. 13: Details of the TSRC (SR: Simple Reduction).

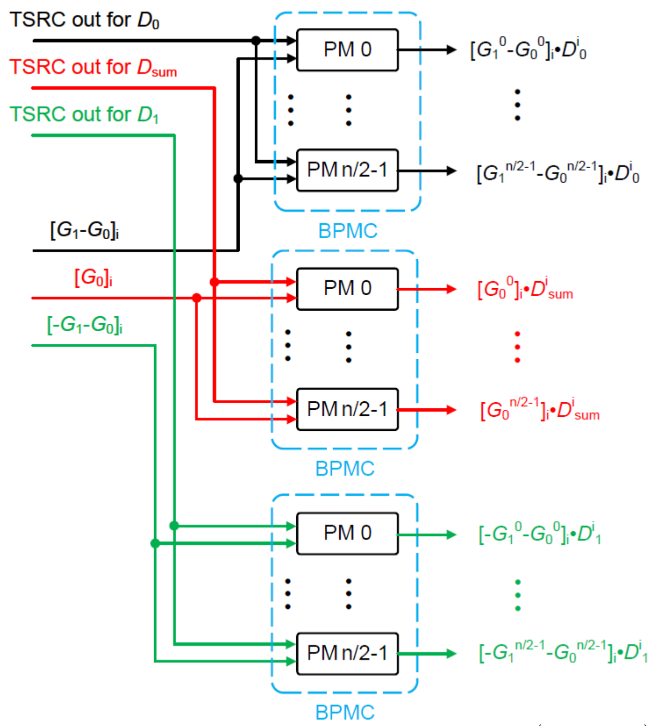

Fig. 14: Details of the TMPC (D_{sum} represents $(D_0 + D_1)$).

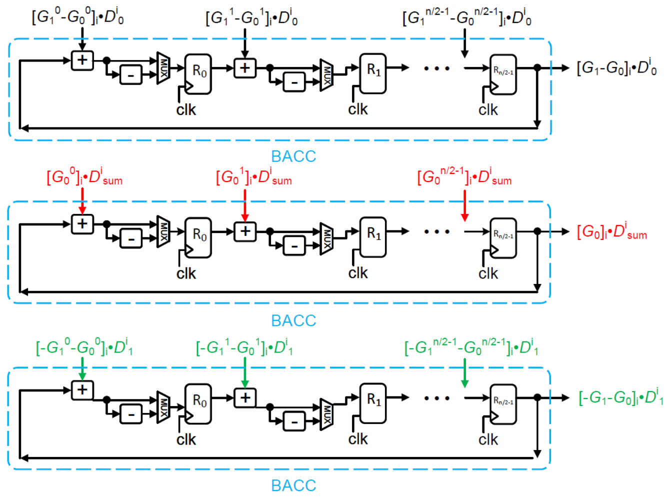

Fig. 15: Details of the TACC.

the order of $(w_{n/2-1}, w_0, ..., w_{n/2-2})$, which will take another $n/2$ cycles for the Linear Combination Component (LCC) to form the final outputs.

Linear Combination Component (LCC). The LCC executes two additions, one between $[G_1 - G_0]_i[D_0^i]$ and $[G_0]_i[D_{sum}^i]$ while the other between $[G_1 - G_0]_i[D_0^i]$ and $[-G_1 - G_0]_i[D_1^i]$ to produce the final results of the polynomial multiplication. The LCC works with the TACC synchronously, i.e., it calculates the final results as the TACC outputs the accumulated sum-of-products (two coefficients are produced at each cycle). Two Simple Reductions are also involved at the output end of the LCC to keep the final outputs in the desired range (after modulo reduction).

ConTrol Unit (CTU). The CTU generates all the required control signals for the proposed structure work properly. Overall, it contains consecutive states, i.e., $reset$, $load$, $multi$, $output$, and $done$, respectively. During the $load$ state, the CTU generates enable signals enabling all the registers in the SSRs. However, when entering $multi$ state, a signal $switch$ is pulled up, the enable signals becomes “10000...” (with $(n/2-1)$ 0’s), which means only the $(n/2 - 1)$-th register in the SSRs are enabled. Since the inputs of the registers are the outputs from the other SSR in the same set, the two $(n/2 - 1)$-th registers

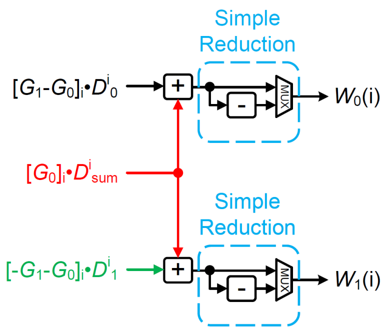
Simple Reduction
$[G_1\text{-}G_0]_i \bullet D^i_0$
$W_0(i)$
MUX
$[G_0]_i \bullet D^i_{sum}$
Simple Reduction
$[\text{-}G_1\text{-}G_0]_i \bullet D^i_1$
$W_1(i)$

Fig. 16: Details of the LCC.

in the same set will switch values while other registers remain unchanged. At the next cycle, the '1' in the enable signal will shift to the right by 1 bit, making the signal "01000...", and the next couple of registers in the same set will switch coefficients. This process will last for $n/2$ cycles until all the last $(n/2-1)$ registers in the SSRs have switched coefficients.

VI. Complexity and Comparison

This section provides an in-depth complexity analysis and implementation comparison to demonstrate the efficiency of the proposed accelerators. We further confirm the proposed strategy is well-suited for applications requiring the multiplication of relatively small-sized polynomials. The complexity and implementation results validate that our accelerators offer superior performance with faster calculation delay and lower area-time complexities than the existing works.

A. Complexity Analysis

For the proposed first accelerator, n registers are involved in the BIPC, while n registers and 2-to-1 MUXes are used in the BACC. Moreover, $3 \times n$ 8-to-1, n 16-to-1, and $3 \times n$ 2-to-1 MUXes are required to construct the point-wise multipliers deployed in the BPMC. The accelerator also needs $(7n+1)$ adders in the BPMC and BACC, along with 200 adders in the BSRC, to calculate and reduce the multiples of d_i. The calculation latency of SCOPE-I is $(n+4)$ cycles.

For the TMVP-based second accelerator, there are $3/2n$ 2-to-1 MUXes and registers in both the TIPC and TACC. Besides, to calculate the negation of the coefficients, there are also $3/2n$ sign-inverters in the TIPC. The number of adders and MUXes in the TSRC is also $3/2n$. Although its theoretical resource usage is larger than the first accelerator, the calculation latency of the second one drops to only $n/2$ cycles (plus the inserted pipelining register cycles of 4).

B. FPGA-based Implementation and Evaluation

To evaluate the actual performance of the proposed accelerators, we have implemented them on the FPGA platform with respect to different parameter sets. In particular, we have: (a) implemented the two accelerators based on one parameter set of Falcon ($n = 512$ and $q = 12,289$); (b) extended the implementations following the parameters of the NIST third-round PQC finalist NTRU [8] (another NTRU-based PQC); (c) extended to other schoobook- and Toom-Cook-based polynomial multiplication designs when $n = 256$.

Experimental Setup. We have set the experiment process following Falcon parameter as: (i) The designs are coded in VHDL, simulated and tested through ModelSim, and finally implemented using Vivado 2022.2; (ii) we have implemented the designs on the following FPGAs, i.e., Artix-7 (XC7a200t) and Ultrascale+ (XCZU9EG-FFVB1156-2), following the existing work in [27], [30]; (iii) we have chosen $n = 512$ and $q = 12,289$ for the implementation; (iv) key implementation results obtained from the implementation, including resource usage, maximum frequency, delay, ADP (area-delay product), and throughput, are listed and compared with the existing works of [27], [30]; (v) we have followed the approach used in [22], [31] to calculate equivalent ADP (EADP) for a balanced area-time overall consideration (see the footnote of Table III); (vi) note that [30] didn't provide the performance data for $n = 512$ but rather an NTT architecture of $n = 1,024$, which can be used to estimate the performance for $n = 512$ (a complete polynomial multiplication deployed with NTT method includes NTT, INTT, and point-wise multiplications)[2]; (vii) the power consumption is not reported since static power constitutes a significant portion of the whole FPGA power.

For the extension to the polynomial multiplication used in NTRU [8], we followed the parameter selection in [28], [9] and [28] to choose $n = 701$, $q = 2^{13}$ and $n = 821$, $q = 2^{12}$. Following the experimental procedure above, the extended work was coded, tested, and implemented on Zynq Ultrascale+ (XCZU9EG-FFVB1156-2) and Zynq-7000 (xc7z100ffg1156-2), respectively, to obtain the performance data. The collected results are listed in Table IV, along with the existing works.

Moreover, for the further verification of the efficiency of the proposed designs when comparing with other schoolbook- and Toom-Cook-based designs, we have implemented our designs ($n = 256$) on Kintex-7 (xc7k480tffv1156-3), Virtex Ultrasclae+ (xcvu9p-flga2577-3-e), Zynq Ultrascale+ (XCZU9EG-FFVB1156-2), Virtex-7 (xc7v2000tflg1925-2L) and Zynq-7000 (xc7z100ffg1156-2), respectively. The obtained performance is listed in Table V, along with [11] and [12].

Performance Discussion. As shown in Table. III, the resource usage (LUT, FF, Slice) increases as the proposed design is switched from SCOPE-I to SCOPE-II. This is because of the three parallel processing matrix-vector products, as seen in Algorithm 4. Meanwhile, though the inserted pipelined register layers bring an extra four cycles to the latency time, both accelerators' working frequencies are very high, indicating this technique's efficiency (similar situations happen with the extension on other PQC schemes, see Table V).

When implementing the designs based on NTRU parameters [8], the resource usage, especially the number of LUTs, drops significantly even though the used n is larger than 512. This is because NTRU implementations don't contain any K-red units or Simple Reductions (q is a power of two). The absence of modulo reductions also produces a higher working frequency for both accelerators as the critical-path is shortened.

[2]Due to the design format (butterfly-based) of an NTT-based polynomial multiplication, the resource usage of an NTT architecture ($n = 512$) is similar to the one of $n = 1,024$. We also use the time of an NTT of $n = 1,024$ for the whole polynomial multiplication of $n = 512$ (complexity of $\mathcal{O}(n\log n)$).

TABLE III: Comparison With The Existing Works (FALCON)

Design	n	Method	LUT	FF	Slice	DSP	BRAM	Fmax[1]	Latency[2]	Delay[3]	ELUT[4]	EADP[5]	EADPR[6]
Zynq Ultrascale+													
[27]	512	NTT	14,327	7,314	NA	4	2	314	2,100	6.7	16,895	112,992	NA
SCOPE-I	512	SB	88,267	35,159	14,598	0	0	525	516	1.0	88,231	86,718	30.30%
SCOPE-II	512	TMVP	157,686	84,226	26,937	0	0	529	260	0.5	157,686	77,502	31.41%
Artix-7													
[27]	512	NTT	14,500	7,287	NA	4	2	142	2,100	14.8	16,371	242,103	NA
SCOPE-I	512	SB	97,322	35,159	15,163	0	0	254	516	2.0	97,322	197,709	18.34%
Kintex Ultrascale+													
[30]*	512	NTT	22,648	15,030	NA	16	24	200	782	3.9	34,456	134,723	NA
SCOPE-I	512	SB	88,185	35,237	14,734	0	0	507	516	1.0	88,185	89,750	33.38%
SCOPE-II	512	TMVP	154,688	87,439	24,503	0	0	410	260	0.6	154,688	98,095	27.19%

Note: Due to the relatively large resource usage of the proposed second accelerator (TMVP-based), we don't implement it on the Artix-7 device.
SB: schoolbook.
*: The performance listed is an estimation since no specific data for $n = 512$ is provided in this work.
[1]: Fmax: Maximum frequency. Unit: MHz
[2]: Latency: Calculation latency (number of cycles). We roughly estimated the NTT-based polynomial multiplication in [27] as 2,100 for $n = 512$.
[3]: Delay = Latency/Fmax. unit: μs.
[4]: ELUT: Equivalent LUT, following [22]. 1 DSP = 102.4 Slices (7 series)/51.2 Slices (UltraScale+); one 18K BRAM = 116.2 Slices (7 series)/58.1 Slices (UltraScale+). UltraScale+ has 8 LUTs in one Slice/CLB while 7 series contains 4 LUTs in one Slice/CLB.
[5]: EADP: Equivalent ADP. EADP = #ELUT×delay (since the Slice number is not available for all designs, we use LUT as the main resource usage metric).
[6]: EADPR: EADP reduction (based on the same FPGA device with the same n).

TABLE IV: Comparison With The Existing Works on NTRU (NIST PQC Third-Round Finalist)

Design	n	q	Method	LUT	FF	Slice	DSP	BRAM	Fmax[1]	Latency[2]	Delay[3]	ELUT[4]	EADP[5]	EADPR[6]
Zynq Ultrascale+														
[10]	701	2^{13}	SB	71,028	18,994	11,661	0	0	223	701	3.14	71,028	223,276	NA
SCOPE-I	701	2^{13}	SB	87,190	41,843	15,069	0	0	577	705	1.22	87,190	106,532	52.29%
SCOPE-II	701	2^{13}	TMVP	150,266	59,677	26,920	0	0	549	354	0.64	150,266	96,893	56.60%
[10]	821	2^{12}	SB	72,430	21,172	11,300	0	0	236	821	3.48	72,430	251,970	NA
SCOPE-I	821	2^{12}	SB	74,760	44,360	12,268	0	0	556	825	1.48	74,760	110,930	55.98%
SCOPE-II	821	2^{12}	TMVP	122,677	64,132	22,863	0	0	513	414	0.81	122,677	99,002	60.71%
Zynq-7000+														
[9]	701	2^{13}	SB	1,463	NA	NA	0	86	76	247,104	3,251.37	21,449	69739901.81	NA
[10]	701	2^{13}	SB	71,321	19,554	20,270	0	0	201	701	3.49	71,321	248,736	NA
SCOPE-I	701	2^{13}	SB	87,191	41,845	25,339	0	0	452	705	1.56	87,191	135,995	45.33%
SCOPE-II	701	2^{13}	TMVP	153,170	58,980	45,710	0	0	416	354	0.85	153,170	130,342	47.60%
[9]	821	2^{12}	SB	1,463	NA	NA	0	86	76	338,664	4,456.11	21,449	95580784.23	NA
[10]	821	2^{12}	SB	71,990	21,202	11,647	0	0	210	821	3.91	71,990	281,447	NA
[28]	821	2^{12}	SB	56,218	21,406	NA	0	0	70	821	11.73	56,218	659,357	NA
SCOPE-I	821	2^{12}	SB	74,773	44,360	22,272	0	0	438	825	1.88	74,773	140,840	49.96%
SCOPE-II	821	2^{12}	TMVP	126,409	63,325	36,336	0	0	436	414	0.95	126,409	120,031	57.35%

SB: Schoolbook.
[1]: Fmax: Maximum working frequency. Unit: MHz.
[2]: Latency: Calculation latency (number of cycles).
[3]: Delay = Latency/Fmax. unit: μs.
[4]: ELUT: Equivalent LUT, following [22]. 1 DSP = 102.4 Slices (7 series)/51.2 Slices (UltraScale+); one 18K BRAM = 116.2 Slices (7 series)/58.1 Slices (UltraScale+). UltraScale+ has 8 LUTs in one Slice/CLB while 7 series contains 4 LUTs in one Slice/CLB.
[5]: EADP: Equivalent ADP. EADP = #ELUT×delay (Slice number is not available for all designs).
[6]: EADPR: EADP reduction (based on the same FPGA device with the same n).

Note we have used the cycle-count-based method combined with FSMs to implement the control units. A major advantage of such a setup is that no external resource is needed to manage data loading/storing operations. This design consideration can be regarded as shifting the workload from memory interaction into the structure, although it might affect the overall timing performance (and area usage). Nevertheless, this setup makes the accelerators more practical for real-world applications.

Comparison With The Existing Works on FALCON. Not many specific works have been released on polynomial multiplication for FALCON. To have a fair comparison, we have: (i) roughly estimated the polynomial multiplication latency (from the whole signature verification phase) as 2,100 for $n = 512$; (ii) estimated the resource usage of the polynomial multiplication of $n = 512$ from the provided NTT architecture performance data of $n = 1,024$ (from [30]); (iii) transferred all the implementation results into the EADP format for a balanced consideration. As seen from Table III, the proposed accelerators have significantly better area-time complexities than [27] and [30]. More precisely, on the Zynq Ultrascale+ device, the first and second accelerators have 30.30% and 31.41% lower EADP than [27], respectively. On the Artix-7 device, the first accelerator still holds an 18.34% lower EADP than [27]. While comparing with another design of [30], the proposed designs are 33.8% and 27.19% more efficient in EADP. The overall efficiency of the proposed accelerators comes from

TABLE V: Comparison With Existing Works on Other Lattice-based PQC (Schoolbook- and Toom-Cook-based Designs)

Design	n	q	Method	LUT	FF	Slice	DSP	BRAM	Fmax[1]	Latency[2]	Delay[3]	ELUT[4]	EADP[5]	EADPR[6]
Kintex-7														
[11]	256	7,681	SB	20,000	18,000	8,000	128	0	260	258	1.0	72,429	71,872	NA
SCOPE-I	256	12,289	SB	57,339	20,736	16,523	0	0	449	260	0.6	57,339	33,203	48.93%
SCOPE-II	256	12,289	TMVP	115,108	52,016	33683	0	0	345	132	0.4	115,108	44,041	38.72%
Virtex Ultrascale+														
[11]	256	7,681	SB	19,000	18,000	3,300	128	0	298	258	0.9	71,429	61,841	NA
SCOPE-I	256	12,289	SB	54,085	20,748	9,330	0	0	571	260	0.5	54,085	24,627	60.18%
SCOPE-II	256	12,289	TMVP	100,725	52,271	16,564	0	0	528	132	0.3	100,725	25,181	59.28%
Zynq Ultrascale+														
[12]	256	2^{13}	TM4	4,550	NA	NA	44	10	588	726	1.2	27,220	33,609	NA
SCOPE-I	256	2^{13}	SB	30,814	14,873	5,438	0	0	607	260	0.4	30,814	13,199	60.73%
SCOPE-II	256	2^{13}	TMVP	53,698	21418	8,766	0	0	540	132	0.2	53,698	13,126	60.94%
Virtex-7														
[12]	256	2^{13}	TM4	4,330	NA	NA	44	10	476	726	1.5	27,000	41,181	NA
SCOPE-I	256	2^{13}	SB	30,577	14,883	9,266	0	0	435	260	0.6	30,577	18,276	55.62%
SCOPE-II	256	2^{13}	TMVP	53,867	21,509	15,918	0	0	418	132	0.3	53,867	17,011	58.69%
Zynq-7000														
[12]	256	2^{13}	TM4	4,550	NA	NA	44	10	400	726	1.8	27,220	49,405	NA
SCOPE-I	256	2^{13}	SB	30,582	14,874	9,162	0	0	476	269	0.6	30,582	17,283	65.02%
SCOPE-II	256	2^{13}	TMVP	53,877	21,544	15,343	0	0	409	132	0.3	53,877	17,388	64.80%

SB: Schoolbook. TM4: Toom-Cook-4.
[1]: Fmax: Maximum working frequency. Unit: MHz
[2]: Latency: Calculation latency (number of cycles).
[3]: Delay = Latency/Fmax. Unit: μs.
[4]: ELUT: Equivalent LUT, following [22]. 1 DSP = 102.4 Slices (7 series)/51.2 Slices (UltraScale+); one 18K BRAM = 116.2 Slices (7 series)/58.1 Slices (UltraScale+). UltraScale+ has 8 LUTs in one Slice/CLB while 7 series contains 4 LUTs in one Slice/CLB.
[5]: EADP: Equivalent ADP. EADP = #ELUT$\times$delay (since Slice is not reported for every design).
[6]: EADPR: EADP reduction (based on the same FPGA device with the same n).

two major efforts. (i) Though the schoolbook/TMVP-based designs' theoretical complexities are not as good as the NTT-based ones, the proposed point-wise multipliers and architectures have significantly offset this drawback. For instance, the first accelerator has only 6.16x more area usage (LUTs) than [27] while its latency cycles are almost 4.15x faster, which indicates the two designs' implementation complexities are quite close. (ii) The simple architectural setup (NTT-based structure generally has a more sophisticated control setup than schoolbook-based ones) and pipelined register insertion have enhanced the timing performance of the proposed designs. Following the above-mentioned example, the first accelerator has 1.67x higher frequency than [27], which eventually leads to a 30.30% reduction in EADP when comparing with [27].

Comparison With Other Designs on NTRU. Having extended the proposed designs on NTRU with $n = 701$, $q = 2^{13}$ and $n = 821$, $q = 2^{12}$ (the same as [10], [28] and [9]), we re-obtained all the performance data including resource usage such as LUT, FF, Slice, (etc.) and timing performance criteria like latency and delay, and listed them in Table IV, as well as those of the existing designs. Again, we can see that the proposed designs obtained significant reductions in EADP over the existing ones. Particularly, on the Zynq Ultrascale+ device, the first accelerator involves 52.29% and 55.98% less EADP than [10] for $n = 701$ and $n = 821$, respectively, while the second accelerator has respective 56.6% and 60.71% less EADP for the two selections of n. On the Zynq-7000 device, for $n = 701$, the first and second accelerator obtains at least 45.33% and 60.71% less EADP than the existing designs. For $n = 821$, compared with the three existing designs, the proposed accelerators have at least 49.96% and 57.35% EADP than the existing ones, respectively.

Comparison with Other PQC Schemes. To better showcase the superior performance of the proposed designs, we have also listed the implementation complexities on other PQC schemes for comparison in Table V. Particularly, we have chosen with $n = 256$, $q = 12,289$ and $n = 256$, $q = 2^{13}$, for comparing the designs in [11] and [12]. Table V shows that the first and second accelerators have 38.72% and 59.28% less EADP on Kintex-7 and Virtex Ultrascale+ devices, respectively. For the schemes with a modulo that is a power of 2 (i.e., 2^{13}), the proposed designs implemented on the Zynq Ultrascale+ and Zynq-7000 devices have 60.73% and 64.80% less EADP than the existing design of [12].

Both designs have superior performance to the existing designs, indicating the proposed SCOPE design strategy is effective and efficient. Benefiting from the proposed point-wise multiplier and architectural arrangement, the proposed accelerators have controllable area usage but with very small latency cycles. Meanwhile, simple control setup and register insertion have also uplifted the working frequencies of the designs. These efforts, in total, have contributed significantly to the overall efficiency of the proposed accelerators.

Discussion and Future Work. The proposed SCOPE can be seen as an alternative solution to the NTT-based polynomial multiplication for the NTRU-based (or other lattice-based) PQC when n is relatively small. For large n (such as $n = 1,024$), however, the implementation will be very large (limited by the complexity of $\mathcal{O}(n^2)$ or $\mathcal{O}(3n^2/4)$) and hence unsuitable for practical applications. Thus, new solutions are needed to deploy the proposed strategy, which will be our future effort. Future work can also be deploying the proposed SCOPE in the actual cryptoprocessor building and developing new polynomial multiplication implementation strategies.

VII. Conclusion

This paper presents a novel schoolbook-based method for efficiently implementing polynomial multiplication in NTRU-based PQC. We have proposed a novel LUT-based point-wise multiplier (combined with modulo reduction techniques) for efficient implementation. Then, a novel architecture is presented, deploying the proposed point-wise multiplier to minimize related hardware usage. After that, we have also extended the proposed design to a TMVP-based accelerator to obtain lower-latency implementation. The final evaluation confirms the efficiency of the proposed design strategy. Overall, the proposed methodology provides an important advancement for hardware acceleration of schemes like Falcon. We hope this research can impact the ongoing PQC standardization and initiate a new framework for PQC hardware implementation.

VIII. Acknowledgments

The work of Jiafeng Xie was supported by NIST-60NANB20D203 and NSF SaTC-2020625.

References

[1] D. J. Bernstein, "Introduction to post-quantum cryptography," in *Post-quantum cryptography*, pp. 1–14, Springer, 2009.

[2] 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.

[3] B. J. Lucas and et al., "Lightweight hardware implementation of binary ring-LWE PQC accelerator," *IEEE Computer Architecture Letters*, vol. 21, no. 1, pp. 17–20, 2022.

[4] D. Micciancio and O. Regev, "Lattice-based cryptography," in *Post-quantum cryptography*, pp. 147–191, Springer, 2009.

[5] R. Chaudhary, G. S. Aujla, N. Kumar, and S. Zeadally, "Lattice-based public key cryptosystem for internet of things environment: Challenges and solutions," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4897–4909, 2018.

[6] J. Hoffstein, D. Lieman, J. Pipher, and J. H. Silverman, "NTRU: A public key cryptosystem," *NTRU Cryptosystems, Inc.(www. ntru. com)*, 1999.

[7] Falcon. https://falcon-sign.info/, 2022.

[8] C. Chen *et al.*, "NTRU," *NIST third round PQC submissions*, 2020.

[9] P. Choi and D. K. Kim, "Lightweight polynomial multiplication accelerator for NTRU using shared SRAM," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 70, no. 12, pp. 4574–4578, 2023.

[10] P. He, Y. Tu, A. Khalid, M. O'Neill, and J. Xie, "HPMA-NTRU: High-performance polynomial multiplication accelerator for ntru," in *2022 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pp. 1–6, 2022.

[11] D.-e.-S. Kundi, Y. Zhang, C. Wang, A. Khalid, M. O'Neill, and W. Liu, "Ultra high-speed polynomial multiplications for lattice-based cryptography on FPGAs," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 4, pp. 1993–2005, 2022.

[12] J. Wang, C. Yang, F. Zhang, Y. Meng, S. Xiang, and Y. Su, "A high-throughput Toom-Cook-4 polynomial multiplier for lattice-based cryptography using a novel Winograd-schoolbook algorithm," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2023.

[13] J. Xie, P. He, and C.-Y. Lee, "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, IEEE, 2021.

[14] Z.-Y. Wong, D. C.-K. Wong, W.-K. Lee, K.-M. Mok, W.-S. Yap, and A. Khalid, "KaratSaber: New speed records for Saber polynomial multiplication using efficient Karatsuba FPGA architecture," *IEEE Transactions on Computers*, 2023.

[15] P. He, Y. Tu, T. Bao, L. Sousa, and J. Xie, "COPMA: Compact and optimized polynomial multiplier accelerator for high-performance implementation of LWR-based PQC," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, no. 4, pp. 596–600, 2023.

[16] W. Tan, A. Wang, X. Zhang, Y. Lao, and K. K. Parhi, "High-speed vlsi architectures for modular polynomial multiplication via fast filtering and applications to lattice-based cryptography," *IEEE Transactions on Computers*, 2023.

[17] P. He, Y. Tu, Ç. K. Koç, and J. Xie, "Hardware-implemented lightweight accelerator for large integer polynomial multiplication," *IEEE Computer Architecture Letters*, pp. 1–4, 2023.

[18] F. Yaman, A. C. Mert, E. Öztürk, and E. Savaş, "A hardware accelerator for polynomial multiplication operation of CRYSTALS-KYBER PQC scheme," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1020–1025, IEEE, 2021.

[19] N. Gupta, A. Jati, A. Chattopadhyay, and G. Jha, "Lightweight hardware accelerator for post-quantum digital signature CRYSTALS-Dilithium," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2023.

[20] "NIST, post-quantum cryptography, https://csrc.nist.gov/projects/post-quantum-cryptography," 2023.

[21] J. Xie, W. Zhao, H. Lee, D. B. Roy, and X. Zhang, "Hardware Circuits and Systems Design for Post-Quantum Cryptography – A Tutorial Brief," *IEEE Transactions on Circuits and Systems II: Express Briefs*, pp. 1–7, 2024.

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

[23] J. Hoffstein, J. Pipher, and J. H. Silverman, "NTRU: A ring-based public key cryptosystem," in *International algorithmic number theory symposium*, pp. 267–288, Springer, 1998.

[24] Post-quantum cryptography round 3 submissions. https://csrc.nist.gov/projects/post-quantumcryptography/round-3-submissions.

[25] H. Fan and M. A. Hasan, "A new approach to subquadratic space complexity parallel multipliers for extended binary fields," *IEEE Transactions on Computers*, vol. 56, no. 2, pp. 224–233, 2007.

[26] J.-S. Pan, C.-Y. Lee, A. Sghaier, M. Zeghid, and J. Xie, "Novel systolization of subquadratic space complexity multipliers based on toeplitz matrix–vector product approach," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 7, pp. 1614–1622, 2019.

[27] L. Beckwith, D. T. Nguyen, and K. Gaj, "High-performance hardware implementation of lattice-based digital signatures." Cryptology ePrint Archive, Paper 2022/217, 2022. https://eprint.iacr.org/2022/217.

[28] Z. Qin, R. Tong, X. Wu, G. Bai, L. Wu, and L. Su, "A compact full hardware implementation of PQC algorithm NTRU," in *2021 International Conference on Communications, Information System and Computer Engineering (CISCE)*, pp. 792–797, 2021.

[29] P. Longa and M. Naehrig, "Speeding up the number theoretic transform for faster ideal lattice-based cryptography," in *Cryptology and Network Security: 15th Int. Conf.*, pp. 124–139, 2016.

[30] B. Li, Y. Yan, Y. Wei, and H. Han, "Scalable and parallel optimization of the number theoretic transform based on FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2023.

[31] Y. Tu, P. He, Ç. K. Koç, and J. Xie, "LEAP: Lightweight and efficient accelerator for sparse polynomial multiplication of HQC," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2023.