Optimized Implementation of Multi-channel Parallel Polynomial Multipliers for Crystal-Kyber

Yihao Zhang, Xiaoting Hu, Hanpeng Zhu, Zelong Dai

Abstract—**In the lattice-based post-quantum crypto scheme Crystal-Kyber, efficient implementation of polynomial multiplication is crucial to achieve high-performance system. This paper proposes an efficient multi-channel parallel polynomial multiplication accelerator designed to optimize the execution speed of polynomial multiplication. Specifically, it presents a novel butterfly computing unit that integrates 32 parallel butterfly units and employs a low-complexity Barrett modular reduction algorithm to reduce computational complexity. To further optimize performance, a cross-multiplexing architecture is utilized to construct the PWM hardware structure, effectively enhancing resource utilization and reducing computation cycles. In addition, during the butterfly processing unit stage, a dynamic data output mechanism is employed to simplify the RAM access control logic and improve memory access efficiency. Experimental results demonstrate that the proposed scheme completes NTT, INTT, and PWM operations in 58, 58, and 29 clock cycles, respectively. Compared with other schemes, the total number of clock cycles is significantly reduced, which is valuable for implementing efficient lattice-based post-quantum cryptographic system.**

Index Terms—**polynomial multiplication, Crystal-Kyber, butterfly operation, number theoretic transform, post-quantum cryptography**

I. INTRODUCTION

THE rapid development of quantum computing technology has presented unprecedented challenges to traditional public key cryptography systems. Quantum computers, employing the Shor algorithm [1], can efficiently solve complex problems such as large integer factorization and discrete logarithm in polynomial time, thereby posing a significant security threat to existing encryption standards. In response to this challenge, NIST launched their PQC project in 2016 with a global call for proposals. The goal was to solicit, evaluate and standardize new cryptographic algorithms designed to be secure against the threat of quantum computing. Following six years of rigorous evaluation, on August 13, 2024, NIST officially standardized CRYSTALS-Kyber as FIPS 203 to enhance resilience against quantum computer attacks [2].

Manuscript received March 21, 2025; revised August 16, 2025.

This work was supported by the Doctoral Fund Project of Jiangsu Normal University of China under Grant 20XSRX014.

Yihao Zhang is a postgraduate student of the College of Computer Science and Technology, Jiangsu Normal University, Xuzhou 221000, China (e-mail: zyh20230601@jsnu.edu.cn).

Xiaoting Hu is a lecturer of the College of Computer Science and Technology, Jiangsu Normal University, Xuzhou 221000, China (corresponding author. e-mail: hxt@jsnu.edu.cn).

Hanpeng Zhu is a postgraduate student of the College of Computer Science and Technology, Jiangsu Normal University, Xuzhou 221000, China (e-mail: 2020230596@jsnu.edu.cn).

Zelong Dai is a postgraduate student of the College of Computer Science and Technology, Jiangsu Normal University, Xuzhou 221000, China (e-mail: 2020220553@jsnu.edu.cn).

Although the Kyber algorithm offers notable security advantages, its implementation requires a substantial number of complex polynomial multiplication operations, leading to considerable resource consumption in hardware applications [3], [4], [5]. Consequently, optimizing computing speed and resource utilization within constrained hardware environments has emerged as a critical issue.

In recent years, extensive research has been conducted to optimize the implementation of the Kyber algorithm. Li [6] and Xing [7] utilize Barrett modular reduction to construct basic butterfly units, thereby enhancing the computational efficiency of these units. Bisheh-Niasar proposed a register-free butterfly cell based on the K^2-RED modular reduction algorithm and applied it to four parallel NTT running units, resulting in a high-speed design [8], [9]. Lü enhanced the K^2-RED algorithm and subsequently designed a new butterfly computing unit. This new computing unit not only supports Number Theoretical Transform (NTT) and its inverse (INTT) operations but also can efficiently handle Point-Wise Multiplication (PWM) operations [10]. In terms of storage optimization, Chen implemented a ping-pong structure to store polynomial coefficients and a pipeline architecture to reduce the resource consumption of butterfly computing units [11]. Yang shortened the computation cycle of critical operations by executing multiple butterfly units in parallel and enhanced memory access efficiency by optimizing multi-RAM channel storage [12]. Li explored a memory access scheme suitable for parallel processing through data reuse and memory grouping techniques and proposed a reconfigurable, high-speed, and area-efficient polynomial multiplication accelerator [13]. Dang designed a double butterfly structure to optimize the control circuit, reduce clock waiting times, and facilitate pointwise multiplication through interleaved iterations of pipelines [14]. Li introduced a novel Split Radix DGT algorithm that employs the Split Radix technique to diminish computational complexity while preserving the transformation length of the DGT. This algorithm achieves a reduction in multiplication operations by a minimum of 10% compared to the most recent NTT algorithm for a polynomial length of 128. Additionally, a specialized flow replacement network was developed to minimize idle periods and enable full pipeline operation [15]. Ni proposed an optimized modular multiplication architecture that integrates K^2-RED with lookup table algorithms [16]. Nguyen initially proposed a non-memory-based iterative NTT architecture, utilizing double butterfly cells for NTT/INTT and PWM operations, and designed configurable reordering cells capable of rearranging coefficients at each NTT/INTT stage [17].

The main contributions of our work can be summarized as follows:

1) We propose an architecture to accelerate polynomial multiplication within the Kyber algorithm. This architecture employs 32 Butterfly Units (BU) and

schedules the butterfly operation modules with parallel pipelining, achieving dynamic scheduling during the execution of NTT, INTT, and PWM operations.

2) We optimize PWM implementation by developing a runtime-reconfigurable core composed of four cross-multiplexing basic BUs, optimizing time efficiency in terms of area-time performance for polynomial multiplication.
3) Through the optimization of the scheduling mechanism for the output cache of the data storage and processing unit, we realize compact control logic, thus reducing the occupation of additional resources.

The remainder of this paper is organized as follows: Section II introduces the notations used in this paper, the Kyber public-key cryptography, and polynomial multiplication based on number-theoretic transforms. Section III presents our design, including the PWM architecture with multi-path parallel operations and the scheduling scheme under a constant structure. Section IV provides the implementation results and comparisons with state-of-the-art works. Finally, Section V concludes the paper.

II. PRELIMINARIES

Kyber is a lattice-based encryption algorithm whose security is based on the Module Learning with Errors (MLWE) problem. The Kyber public key encryption scheme, as shown in Figure 1, consists of three core algorithms: key generation, encryption, and decryption.

In the key generation stage, first select matrix $\hat{A}$ and private key s from the uniform distribution and binomial distribution, respectively. Subsequently, the product of $\hat{A}$ and $\hat{\mathrm{s}}$ is calculated in the NTT domain, and noise $\hat{e}$ is added to the result to generate the public key pk, which is $pk = \hat{A} \circ \hat{s} + \hat{e}$.

The encryption process can be described as three steps: 1) Calculate $v = \hat{t}^T \circ \hat{r} + e_2 + Decompress_q(m, 1)$, where m represents the message, e_2 represents the noise, $\hat{t}^T$ is the transpose of the public key vector, and $\hat{r}$ is the value on the NTT field generated through binomial distribution; 2) Calculate $u = \hat{A}^T \circ \hat{r} + e_1$, where at is the transpose of $\hat{A}^T$ in the key generation stage, and e_1 is the noise vector; 3) Compress u and v to generate ciphertext *ct*. Namely $ct = (Compress_q(u, d_u) \,||\, Compress_q(v, d_v))$.

The decryption process calculates the product of the private key $\hat{s}$ and the matrix u to recover the approximate value of the vector v, and then decrypts the original message, that is, $m = Compress_q(v - \hat{\mathrm{s}}^T u, 1)$.

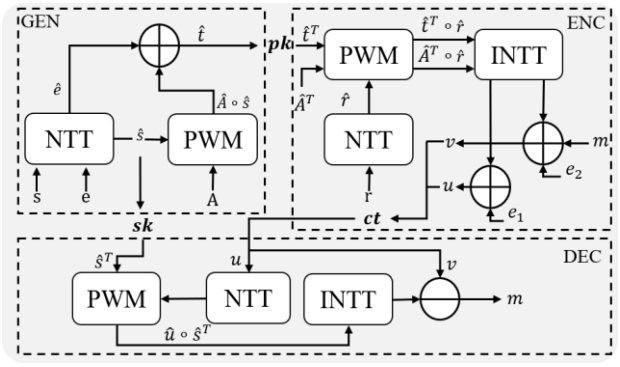


Fig. 1. Kyber Public Key Encryption Structure

In Kyber's encryption process, operations on polynomial rings are the core. This polynomial ring is defined as $R_q = \mathbb{Z}_q[x]/(x^n + 1)$, where $\mathbb{Z}_q$ The integer ring representing modulus q, $x^n + 1$ is an irreducible polynomial. Under these conditions, the coefficients of the polynomial are limited to the range of modulus q, which means that addition and subtraction operations are relatively simple, but multiplication rules are more complex. Matrix encryption and homomorphic operations rely on efficient multiplication over polynomial rings [18]. To optimize the efficiency of polynomial multiplication, Kyber typically employs polynomial multiplication operations based on NTT.

Polynomial multiplication based on NTT mainly includes three types of operations: the NTT operation that maps polynomial coefficients to the NTT domain, the point wise multiplication operation PWM, and the inverse NTT operation INTT on the elements in the NTT domain. The main advantage of NTT based polynomial multiplication is that it can reduce the complexity of polynomial multiplication from direct convolution calculation to more efficient coefficient point multiplication. For example, suppose a 256-degree polynomial $f(x) = f_0 + f_1 x + \cdots + f_{255} x^{255}$, then the NTT transformation of f can be expressed as (1):

$$\mathrm{NTT}(f) = \hat{f} = \hat{f}_0 + \hat{f}_1 X + \cdots + \hat{f}_{255} X^{255} \quad (1)$$

where $\hat{f}_{2i}$ and $\hat{f}_{2i+1}$ $(i = 0, \ldots, 127)$ are defined as,

$$\hat{f}_{2i} = \sum_{j=0}^{127} f_{2j} \omega^{(2br_7(i)+1)j} \quad (2)$$

$$\hat{f}_{2i+1} = \sum_{j=0}^{127} f_{2j+1} \omega^{(2br_7(i)+1)j} \quad (3)$$

with ω is a primitive 256th root of unity and $br_7(i)$ is a 7-bit bit reverse operation on *i*.

PWM is the multiplication operation of $\mathbb{Z}_{3329}[x]/\left(X^2 - \omega^{(2br_7(i)+1)}\right)$ on the ring. Assuming PWM multiplication is denoted as $\hat{h} = (\hat{f} \circ \hat{g})$, $\hat{h}$ can be calculated by (4).

$$\begin{cases} \hat{h}_{2i} = \hat{f}_{2i} \cdot \hat{g}_{2i} + \hat{f}_{2i+1} \cdot \hat{g}_{2i+1} \cdot \omega^{(2br_7(i)+1)} \\ \hat{h}_{2i+1} = \hat{g}_{2i} \cdot \hat{f}_{2i+1} + \hat{f}_{2i} \cdot \hat{g}_{2i+1} \end{cases} \quad (4)$$

Compared to traditional multiplication, the multiplication method based on NTT transform reduces the operational complexity from $O(n^2)$ to $O(n \log n)$. This not only simplifies the process of polynomial multiplication but also improves the efficiency of the entire encryption algorithm. Algorithms 1 and 2 outline the specific processes of NTT and INTT as utilized in the Kyber algorithm.

Algorithm 1: Forward Transform NTT

Input: $f = (f_0, f_1, \cdots, f_{n-1})$, ω, q;
Output: $NTT(f)$;
1. for $i = log_2\, n$ downto 1 do
2. $m \leftarrow 2^i, r \leftarrow 0$
3. for $k = 0$ to $n - 1$ by m do
4. $\omega \leftarrow ROM[r + br_7(m/2)]$
5. for $j = 0$ to $m/2 - 1$ do
6. $u \leftarrow f_{j+k}$; $t \leftarrow \omega \cdot f_{j+k+m/2}$
7. $f_{j+k} \leftarrow u + t$; $f_{j+k+m/2} \leftarrow u - t$
8. end for
9. $r \leftarrow r + 1$
10. end for
11. end for
12. Return $\mathrm{NTT}(f)$.

Algorithm 2: Inverse transformation INTT

Input: $\hat{f} = (\hat{f}_0, \hat{f}_1, \cdots, \hat{f}_{n-1}),\ \omega^{-1},\ q;$

Output: $INTT(\hat{f})$;

1. for $i = 1$ to $log_2\, n$ do
2. $m \leftarrow 2^i, r \leftarrow 0$
3. for $k = 0$ to $n - 1$ by m do
4. $\omega \leftarrow ROM[r + br_7(m/2)]$
5. for $j = 0$ to $m/2 - 1$ do
6. $u \leftarrow (\hat{f}_{j+k} + \hat{f}_{j+k+m/2})/2;$
   $t \leftarrow (\hat{f}_{j+k} - \hat{f}_{j+k+m/2})/2$
7. $\hat{f}_{j+k} \leftarrow u;\ \ \hat{f}_{j+k+m/2} \leftarrow \omega \cdot t$
8. end for
9. $r \leftarrow r + 1$
10. end for
11. end for
12. Return INTT($\hat{f}$).

III. POLYNOMIAL MULTIPLICATION UNIT BASED ON MULTI-CHANNEL PARALLEL NTT IMPLEMENTATION

A. *Overall Architecture*

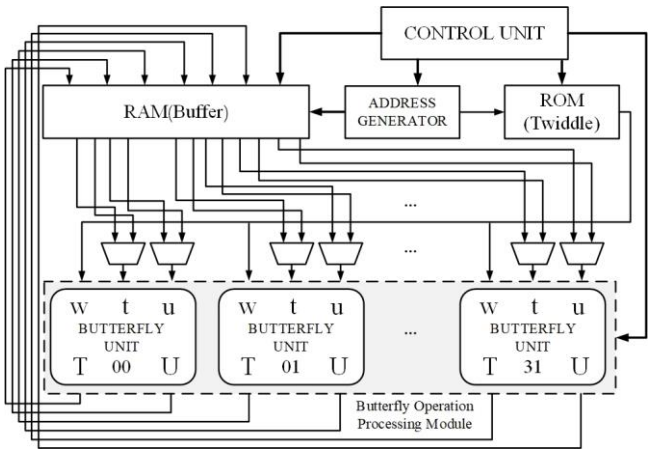


Fig. 2. Polynomial Multiplication Unit Based On NTT

The polynomial multiplication unit, based on the multi-channel parallel NTT implementation presented in this article, is illustrated in Figure 2. It comprises a butterfly operation processing module, an address generator, memory, and a control unit. The butterfly operation unit serves as the core component, significantly enhancing data processing speed. Within this structure, the butterfly operation processing module contains 32 BUs, specifically designed for the parallel processing of high-complexity computational tasks, which integrate NTT, INTT, and PWM modules. The Buffer module is intended to store polynomial coefficients and cache intermediate results throughout the iteration process. Furthermore, the pre-calculated value of ω is stored in the ROM module, which allows rapid retrieval and output to the butterfly operation module during calculations. The control unit oversees the input and output operations of the butterfly operation processing module and ensures that data read from RAM in each clock cycle is accurately directed to this module, synchronized with the twiddle factor obtained from ROM. Concurrently, state variables and state machine control are utilized to generate mapping addresses, update data transmission statuses, and guarantee the coordinated operation of the overall structure. In summary, the polynomial multiplication architecture employs a loosely coupled structural design, enabling flexible execution of NTT, INTT, and PWM operations according to varying parameter configurations, thereby facilitating efficient polynomial multiplication calculations.

B. *Butterfly Unit*

As illustrated in Figure 2, the butterfly operation processing module comprises 32 independent BUs. Each BU, as shown in Figure 3, consists of a modular addition module, a modular subtraction module, multiple cache registers, a DSP multiplier, and a module reduction unit, which are utilized to perform NTT, INTT, and PWM transformations.

$$forward\ \text{NTT}: \quad U = (u + t\omega)\ mod\ q; \qquad T = (u - t\omega)\ mod\ q \tag{5}$$

$$Inverse\ \text{NTT}: \quad U = (u + t)\ mod\ q; \qquad T = (u - t)\omega\ mod\ q \tag{6}$$

NTT transformation and INTT transformation are illustrated in (5) and (6).

In the forward NTT pipeline operation, the modular multiplication is initially executed over four clock cycles (as illustrated in the shaded part of Figure 3), followed by the modular addition and subtraction operation, which requires one clock cycle. The sequence of operations for INTT is reversed; it begins with the modular addition and subtraction module, followed by the modular multiplication operation. To facilitate synchronous data transfer between modules, several cache registers are implemented. Specifically, in both NTT and INTT, the outputs U and T necessitate five clock cycles. In the PWM algorithm, due to the collaboration of multiple modular multiplications, the results of the intermediate modular multiplications are directly output to M for buffering over four clock cycles. Figure 4 illustrates each phase of the single BU pipeline.

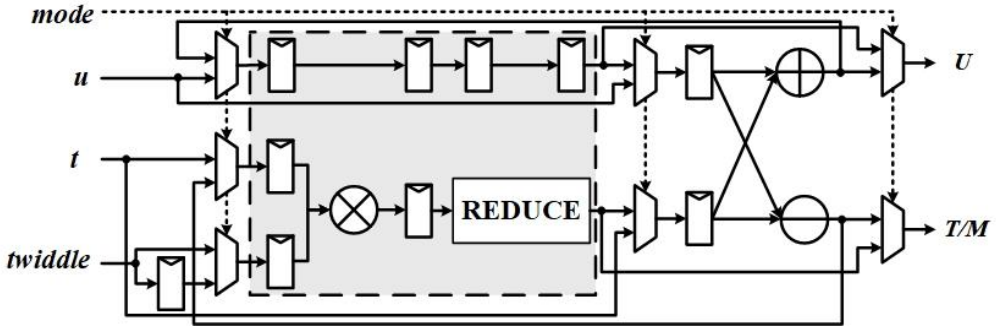


Fig. 3. Single BU Structure

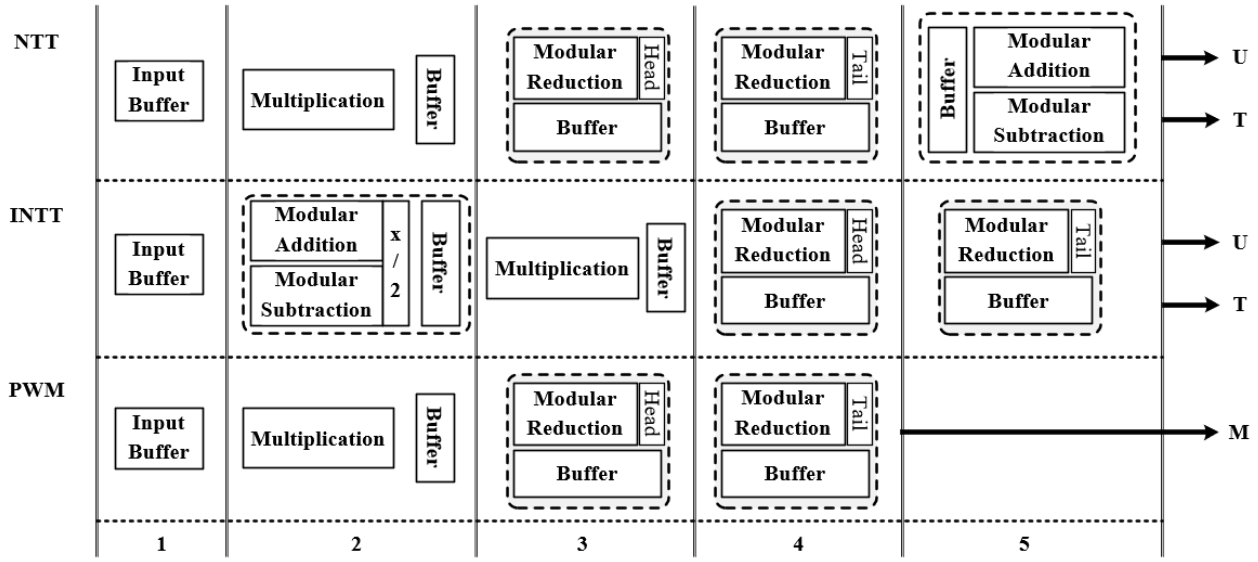


Fig. 4. BU Pipeline

In the BU, the input ports *u* and *t* receive 12-bit coefficients, while the twiddle port synchronously acquires the twiddle factor (ω) from the ROM. Based on the mode signal, data flows along a designated path to execute the corresponding function, with the processed data ultimately being output through the U and T/M ports. The mode signal is configured to 00, 01, and 10, which correspond to the single BU mode configurations in NTT, INTT, and PWM operations, respectively. We utilize the DSP multiplier of the FPGA chip to carry out 12×12 bits multiplication and apply the low-complexity Barrett algorithm to reduce the multiplication result to the NTT domain, as elaborated below.

24-bit low-complexity Barrett modular reduction

In the realization of polynomial multiplication, modular reduction operation (handled by REDUCE module in Figure 3) is the most time-consuming operation. To optimize this operation, the following low-complexity, high-efficiency Barrett modular reduction algorithm is adopted in this paper.

In the Kyber algorithm, since the modulus q of NTT satisfies the specific form $q = k \times 2^m + 1$, where $k = 13$ and $m = 8$, the 24-bit result of the multiplication operation in the shaded part of Figure 3 can be represented as $c = c_1 \cdot 2^8 + c_0$, where $c_0 = c[7:0]$ represents the lower 8 bits of c, $c_1 = c[23:8]$ represents the upper 16 bits. $c \bmod q$ can be converted into (7):

$$\begin{aligned} c \bmod q &= (c_1 \cdot 2^m + c_0) \bmod q \\ &\equiv \left(c_1 \cdot 2^m \bmod (q-1) - \left\lfloor \frac{c_1 \cdot 2^m}{q-1} \right\rfloor + c_0\right) \bmod q \\ &\equiv \left(\left(c_1 \cdot 2^m \bmod (k \cdot 2^m)\right) - \left\lfloor \frac{c_1 \cdot 2^m}{q-1} \right\rfloor + c_0\right) \bmod q \\ &= (r_1 \cdot 2^m - Q_1 + c_0) \bmod q \end{aligned} \tag{7}$$

where $Q_1 = \lfloor c_1/k \rfloor$, $r_1 = c_1 \bmod k$. This transformation converts the modulo q operation into a modulo k operation. Because k is much smaller than q, the computational complexity is significantly reduced.

In order to minimize the complexity of hardware implementation, we further simplified the calculation of Q_1 and r_1.

From equation (7), r_1 is the remainder of c_1 divided by k, and Q_1 is the quotient. Q_1 and r_1 can be respectively rewritten as:

$$Q_1 = \left\lfloor \frac{c_1 \cdot (2^{16}/13)}{2^{16}} \right\rfloor, \qquad r_1 = c_1 - Q_1 \cdot 13.$$

This can be further simplified using the fact that as $13 = 2^3 + 2^2 + 1$ and $2^{16} / 13 = (2^{-3} + 2^{-5})(1 - 2^{-6})\, 2^{15}$. By this conversion, Q_1 and r_1 can be computed using addition, subtraction, and shift operations, thereby minimizing hardware complexity to the greatest extent possible. The specific low-complexity Barrett modular reduction process is detailed in Algorithm 3 as follows.

Figure 5 shows an illustration of the modular reduction structure. In this structure, we utilize fast carry logic element CARRY4 to implement multi-bit addition and subtraction. In this implementation, multiple LUT6 units are employed to handle multi-bit XOR operations, multiple CARRY4 units is used to generate look-ahead multiple parallel carry signals which effectively avoids the delay caused by cascading in traditional carry chains.

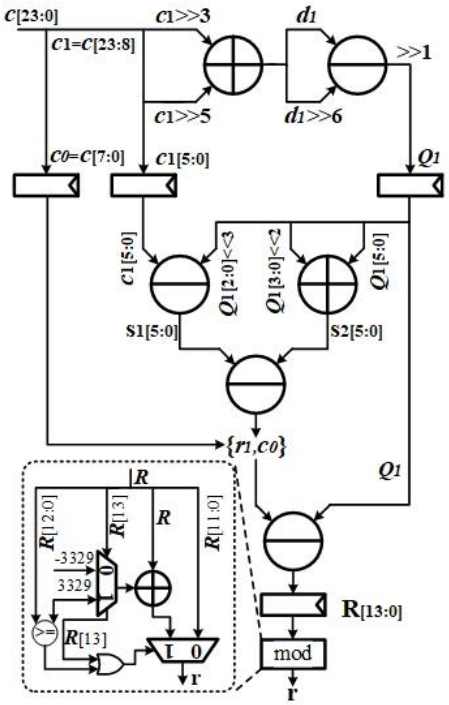


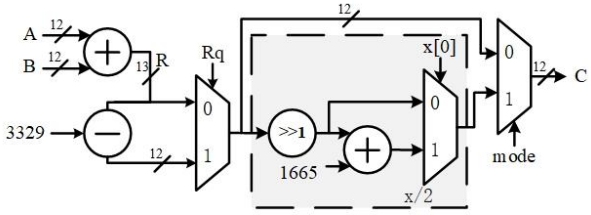
Fig. 5. Modular Reduction Structure

Algorithm 3: Low-Complexity Barrett Modular Reduction

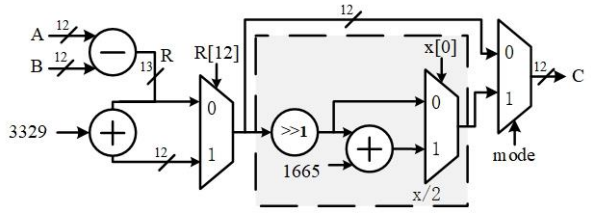
Input : $c = a \cdot b \in [0: 2^{24})$;
Output : $r \equiv c \ mod\ 3329$;
1. $c_1 \leftarrow c[23:8], c_0 \leftarrow c[7:0]$;
2. $d_1 \leftarrow c_1 \gg 3 + c_1 \gg 5$;
3. $d_2 \leftarrow d_1 - d_1 \gg 6$;
4. $Q_1 \leftarrow d_2 \gg 1$;
5. $s_1 \leftarrow c_1[5:0] - Q_1[2:0] \ll 3$;
6. $s_2 \leftarrow Q_1[3:0] \ll 2 + Q_1[5:0]$;
7. $r_1 \leftarrow s_1[5:0] - s_2[5:0]$;
8. $R \leftarrow r_1 \ll 8 + c_0 - Q_1$;
9. $r = mod(R, 3329)$; mod ensures that *r* is smaller than 3329
10. Return $r \in \mathbb{Z}_{3329}$.

Modular addition/subtraction and $x/2$ modular multiplication combined module

In a single BU, the modular addition and modular subtraction operations are used to perform the addition and subtraction of two 12-bit input data. To ensure the temporal synchronization of the two data streams, the modular addition and modular subtraction operations are designed as two independent modules. Additionally, the $x/2$ operation is introduced as an additional step during the execution of Algorithm 2. To avoid module redundancy, the $x/2$ operation is embedded within the modular addition and subtraction modules, and a modular compensation strategy is employed for its design.



(a) Modular addition and $x/2\ mod\ q$



(b) Modular subtraction and $x/2\ mod\ q$

Fig. 6. Modular Addition/Subtraction and $x/2\ mod\ q$ Combination Module

Figure 6(a) illustrates the structure of the combined module for modular addition and $x/2$ modular multiplication. The left part of Figure 6 is used to calculate modular addition, the shaded part is responsible for executing the $x/2\ mod\ q$ operation, the right multiplexer is used to choose the final output c according to working mode. Note that, the $x/2\ mod\ q$ is needed only during INTT in the BUs when in mode signal 01. In this paper, this operation is implemented based on (8).

$$x/2\ \ mod\ q = (x \gg 1) + x[0] \cdot \left(\frac{q+1}{2}\right) \tag{8}$$

As shown Figure 6,when the least significant bit of x, $x[0]$, is 0, the result of $x/2\ mod\ q$ can be derived by shifting x to the right by one bit. Conversely, when $x[0]$ is 1, an addition operation must be conducted based on the shifted result. Because q is a fixed constant, the $x[0] \cdot [(q+1)/2]$ in (8) can be efficiently executed using this method, thereby avoiding complex multiplication and modular operations.

Using a similar approach, we design the modular subtraction module as shown in Figure 6(b). The details are similar to those of the modular addition and will not be repeated here.

C. Optimization of PWM Scheduling

In Kyber algorithm implementation, PWM module is used to calculate $\hat{h}$ in (4). To optimize the implementation of PWM module, Xing [7] converted (4) into (9) applying factorization techniques, reducing the number of multiplication operations from 5 to 4, which significantly decreases DSP resource consumption and enhancing computational efficiency.

$$\begin{cases} \hat{h}_{2i} = \hat{f}_{2i} \cdot \hat{g}_{2i} + \hat{f}_{2i+1} \cdot \hat{g}_{2i+1} \cdot \omega^{(2br_7(i)+1)} \\ \hat{h}_{2i+1} = (\hat{f}_{2i+1} + \hat{f}_{2i})(\hat{g}_{2i} + \hat{g}_{2i+1}) - (\hat{f}_{2i} \cdot \hat{g}_{2i} + \hat{f}_{2i+1} \cdot \hat{g}_{2i+1}) \end{cases} \tag{9}$$

Building upon (9), this paper presents a novel PWM hardware architecture, as illustrated in Figure 7. This architecture reconfigures the original 32 BUs into 8 sets of binomial multiplication units, each of which is referred as a "Core", as illustrated within the dashed box in Figure 7. Each Core comprises 4 BUs, with each unit configured to operate in mode 10. To enhance resource utilization and computational efficiency, a cross-multiplexing mechanism is employed to control and coordinate the operations among the various BUs in each Core. Additionally, we introduce modular addition/subtraction modules and cache registers that collaborate with the BUs to execute multiplication operations on a set of polynomial coefficients.

The computation process in a Core is divided into three stages: Stage 0, Stage 1, and Stage 2. The operations performed at each stage are as follows:

Stage 0: $M_1 = \hat{f}_{2i} \cdot \hat{g}_{2i}, M_2 = \hat{f}_{2i+1} \cdot \hat{g}_{2i+1},$
$a_0 = \hat{f}_{2i+1} + \hat{f}_{2i}, a_1 = \hat{g}_{2i} + \hat{g}_{2i+1};$
Stage 1: $a_2 = M_1 + M_2, M_3 = M_2 \cdot \omega^{rev_{log\,n-1}(2i+1)},$
$M_0 = a_0 \cdot a_1;$
Stage 2: $\hat{h}_{2i} = M_1 + M_3, \hat{h}_{2i+1} = a_2 - M_0.$

Note that the operations in the BU require four clock cycles to complete, while the modular addition operation only requires two clock cycles, therefore, 2-level registers need to be added after modular addition for synchronization.

In addition, in our implementation, a dual-clock preload cache mechanism is employed for the preprocessing of input data, which takes two clock cycles. Subsequently, the loaded data is divided into four groups and fed sequentially into eight Cores over the course of four clock cycles. These eight Cores then process date in pipeline. Finally, the PWM module takes a total of 29 clock cycles to complete the entire operation. Compared to the 56 clock cycles reported in the current optimal literature [13], our approach achieves a 48% reduction in clock count, thereby significantly enhancing data processing speed.

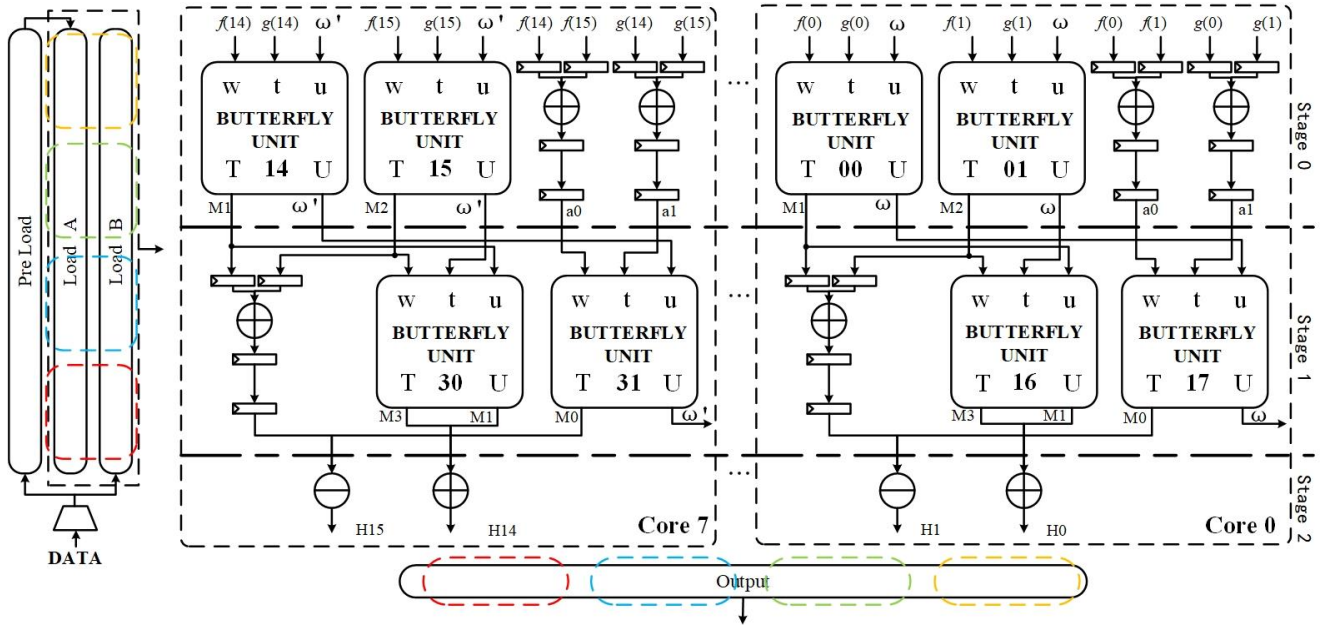


Fig. 7. PWM Module

D. *Storage Solutions*

In Kyber's NTT/INTT algorithm, a degree-256 polynomial is decomposed into two degree-128 polynomials for independent operations, requiring seven rounds of butterfly operations. Since distinct polynomial coefficients are needed in each round, the strategies for storing and controlling coefficient read-write operations significantly impact the algorithm's overall performance. For instance, in an 8-point NTT operation shown in Figure 8, the accessed coefficient pairs differ in each round. In the first round, the pairs are (0,4), (2,6), (1,5), and (3,7). In the second round, they are (0,2), (4,6), (1,3), and (5,7). Finally, in the third round, the pairs are (0,1), (4,5), (2,3), and (6,7). To implement the NTT operation, effective storage and control strategies must be designed to ensure accurate data read/write operations, with processed data stored at specific locations for correct and efficient retrieval in subsequent rounds. A single buffer is insufficient due to risks of data overwriting.

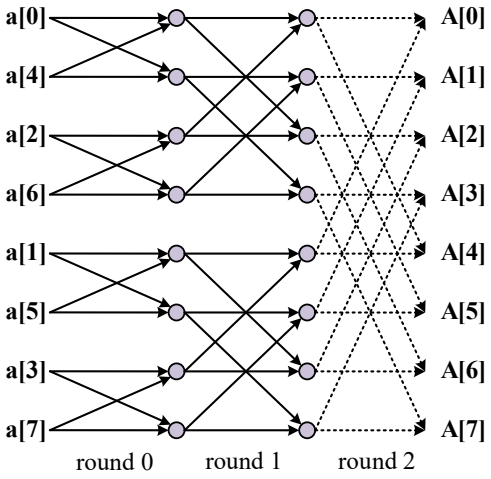


Fig. 8. 8-point NTT Butterfly Operation

To address the limitation of single buffer, we adopt a double buffer mechanism in the output stage of the butterfly computing unit. Specifically, the first buffer stores data processed in the previous clock cycle. In the subsequent cycle, this data is transferred to the second buffers while the second buffers simultaneously receives new data from the current cycle's processing. This dual-buffering mechanism allows data from two consecutive clock cycles to be output

Algorithm 4: RAM Read Address

```
Input ： stage, mode, cycle_counter (clct);
Output： raddr;
1. WHEN ( mode = NTT ) Begin
2. while ( clct = (stage << 3) || ( stage << 3 + 1)
                || (stage << 3+ 2) || (stage << 3 + 3) )
3.    raddr = (stage = 0) ?
                (clct [1] +( clct [0] ? 2 : 0)) : clct [1:0];
4. End
5. WHEN ( mode = INTT ) Begin
6. while( clct = (stage << 3) || (stage << 3 + 1)
                ||(stage << 3+ 2) || (stage << 3 + 3) )
7.    raddr = (stage = 5) ?
                (clct [1]+( clct [0] ? 2: 0)) : clct [1:0];
8. End
9. Return raddr;
```

Algorithm 5: RAM Write Address

```
Input ： stage, mode, raddr, cycle_counter (clct);
Output： waddr;
1. waddr_shift_reg[5:0] <= 0;
2. waddr_shift_reg[0] <= raddr;
3. FOR ( i = 1; i < 6; i = i + 1)
4.    waddr_shift_reg[i] <= waddr_shift_reg[i-1];
5. WHEN ( mode = NTT || INTT ) Begin
6. IF ( stage = 0 )
7.    while ( clct == 6 || clct == 7)
8.       waddr = waddr_shift_reg[5];
9. ELSE IF
10.    while ( clct = (6 + stage << 3) || (7 + stage << 3)
                ||(stage << 3) || (stage << 3+ 1 )|| 56 || 57)
11.       waddr = waddr_shift_reg[5];
12. End
13. Return waddr;
```

synchronously, ensuring an uninterrupted data stream, eliminating the inherent data overwriting problem of single buffer, and ensuring that the output results maintain the correct coefficient order for subsequent read operations.

This mode facilitates a flexible transformation between output and temporary data at the output port, controlled by state variables, thereby ensuring that the output results conform to the required coefficient arrangement order for the following reading operation.

Furthermore, to maintain the accuracy of read and write addresses at each stage, we propose an address generation method based on the value of the current computational stage and cycle counter. The detailed process for generating read and write addresses using this method is outlined in Algorithm 4 and Algorithm 5.

In the address generation process, during the read address generation phase, the read address is determined based on the specific number of bits of the cycle counter value. The output state of the read and write address is controlled by the stage and counter signal. Given that the addresses corresponding to read and write operations exhibit consistency within the same stage, the address generator employs a shift register-based caching strategy to temporarily store the generated read address information. After the data is read, there are five delay cycles required to implement the butterfly operation. Once the butterfly operation is completed, the address value cached in the shift register will be officially passed to the write address terminal to ensure accurate data addressing during storage and operation. Figure 9 shows the RAM read/write control mechanism during the NTT phase as an example to elucidate this process.

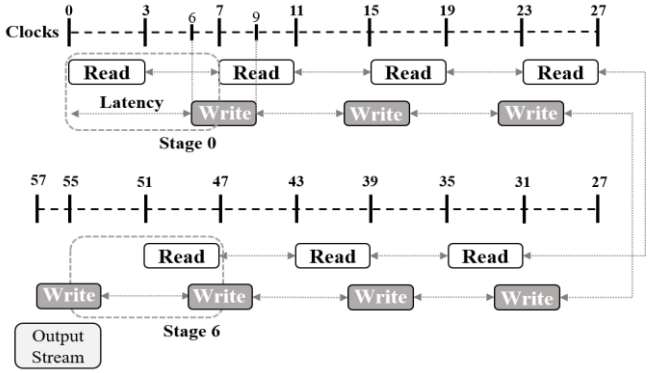


Fig. 9. NTT's RAM Read Write Control

While reading and writing polynomial coefficients, we consolidate the twiddle factors to enhance parallel processing efficiency. The power term of ω^k makes up the twiddle factor. These values are typically precalculated and stored to increase computational performance.

For $0 \le k \le (n/2) - 1$ and $n = 256$, the twiddle factor for NTT and INTT are ω^k and ω^{-k} respectively. Since the multiplication group created by ω^k is symmetric in group theory, ω^{-k} can be represented by ω^k. Given $\omega^{n/2} \equiv -1\ (mod\ q)$, we have $\omega^{-k} = -\omega^{(n/2)-k}$.

Figure 10 shows the twiddle factors required in different stages of NTT computation. In this figure, a rectangle block represents a twiddle factor and the number on the left side in, while the number on the right side is the hexadecimal representation of the twiddle factor. The stage numbers on the left denote the twiddle factors utilized in the respective NTT period. The number in the upper right corner of each cube indicates its depth, which represents the number of twiddle factors needed for one clock cycle during single stage. Various block styles denote the quantity of BUs that a singular twiddle factor needs to encompass within a single clock cycle. As shown in the figure, different stages need different numbers of twiddle factors. For example, during stage 0, only one twiddle factor 64:6C1 is needed and it will be loaded into 32 BUs and reused cyclically over 4 clock cycles in the operation at this stage. During stage 5, 32 distinct twiddle factors are needed. Each clock cycle requires the retrieval of eight distinct twiddle factors, with each factor encompassing four BUs.

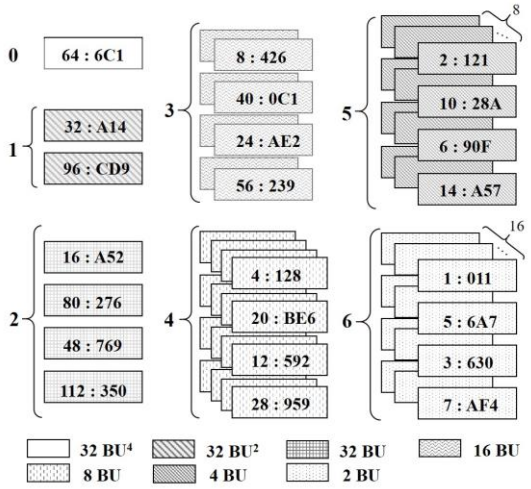


Fig. 10. Example of twiddle factor grouping

IV. Experimental Results and Analysis

To validate the optimization scheme proposed in this article, we implemented the design on the Artix-7 FPGA platform using the Xilinx Vivado 2018.3 software suite. To ensure consistency in area assessment, we employed Slice Equivalent Cost (SEC) as a metric. According to [13] and [19], a DSP48E1 and a 36-KB BRAM are equivalent to 102.4 and 196.2 *SLICE*, respectively. The area is expressed as *SECs = BRAM × 200 + DSPs × 100 + SLICE*, where *SLICE* is defined as *SLICE = LUTs × 0.25 + FFs × 0.125*. The Area Time Product (ATP) objectively reflects the relationship between resource consumption and algorithm performance, with lower values indicating a better balance between the two under constrained conditions. The specific ATP is expressed as *ATP = Area(SECs) × Time(PLMult)*, which includes 2 NTT, 1 INTT, and 1 PWM.

The resource usage of different polynomial multiplication modules is shown in Table I. The control unit uses a serial execution mode, whereas the NTT unit and one BU use a pipelined implementation. Both ROM storage and multi-channel buffer RAM use a parallel implementation.

A single BU necessitates multiplication, modular addition, and modular subtraction operations (as specified in Table I), leading to substantial resource consumption. The NTT unit exhibits higher overall resource usage due to its multi-BU cooperative architecture for polynomial multiplication. Conversely, the control unit consumes minimal resources under low-complexity control schemes, as its primary functions are generating counters, addresses, and control signals.

TABLE I
MODULE RESOURCE CONSUMPTION

Module	LUTs	FFs	DSPs
NTT Unit	9711	7112	32
Single BU	224	114	1
Control Unit	54	21	0
ROM	524	0	0
Buffer(RAM)	772	3840	0

Table II presents the implementation results of the NTT accelerator based on the Kyber algorithm, alongside a comparison with state-of-the-art polynomial multiplication accelerators. Experimental results demonstrate that the proposed NTT core accelerator outperforms existing implementations in polynomial multiplication (PLMult), achieving at least 32.3% fewer clock cycles and 40.3% shorter execution time. This yields the shortest PLMult clock cycles among all schemes, exhibiting significant advantages in overall runtime performance.

In contrast to Scheme [6], a 32-channel multi-channel BU architecture is proposed for concurrent data processing, with enhanced the PWM processing flow via a cross-multiplexing structure. The improved PWM operation can be executed in 29 clock cycles, significantly reducing clock cycle usage relative to traditional techniques. Although the clock cycle counts for individual NTT and INTT operations in our design are marginally more than those documented in [6], the latter necessitates supplementary clock cycles for PWM operations, resulting in an extended overall execution duration. The operational duration of this study has been significantly diminished by 60.2%, and ATP has been reduced by 63.5% relative to the findings in [6]. In comparison to Scheme [16], which demonstrates the highest overall efficiency, this study attains a moderate increase in ATP while incurring a significant decrease in operation time. Compared to Scheme [17], which demonstrates minimal resource consumption, the proposed design increases resource usage but reduces latency by 90.3% and improves ATP by 35.8%, achieving a synergistic enhancement in speed and comprehensive performance.

This work demonstrates significant enhancements in two key metrics: ATP and operational time, as illustrated in Figure 11. This improvement incurs a higher resource consumption cost. Among all the evaluated schemes, the operational time was reduced to its minimum, achieving a speed that was 40.4% faster than the suboptimal design [13]. Parallel computing and pipeline optimization techniques ensure that ATP performance surpasses most comparable schemes, such as [6], [13], and [14].

In summary, the proposed solution significantly enhances the computational efficiency of PLMult, achieving a reduction in computation time of at least 40.3% and a 24% decrease in the time required for a single NTT operation compared to the previously fastest solution. By maintaining a high operating frequency, this scheme effectively utilizes resources to attain optimal Time values in the comprehensive evaluation.

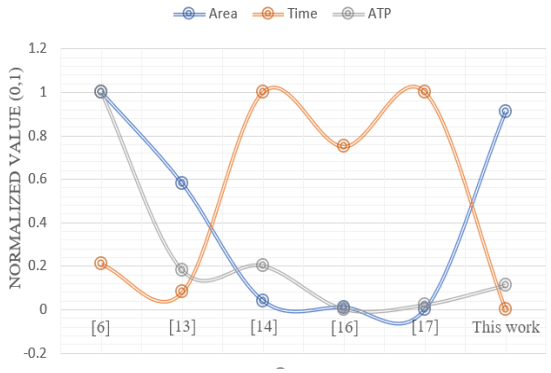


Fig. 11. Comprehensive performance comparison

V. CONCLUSION

With the advent of the quantum computing era, lattice-based cryptography has garnered significant attention due to its potential for post-quantum security. This article optimizes butterfly operations by employing a cross-multiplexing architecture and integrates pipeline technology to achieve an efficient design of BUs at the hardware level. Furthermore, the operations of NTT, INTT, and PWM have been accelerated through a 32-channel parallel processing mechanism. The implementation technique proposed in this research provides significant benefits in improving processing speed and possesses tremendous applicability in postquantum cryptography.

TABLE II
COMPREHENSIVE COMPARISON BETWEEN THE IMPLEMENTATION RESULTS OF NTT AND PREVIOUS WORK

Work	Freq [MHz]	LUTs	FFs	BRAMs	DSPs	SECs	Latency(cc) and Time(μs)				ATP
							NTT	INTT	PWM	PLMult	
[6]	175	8428	3979	11	32	8003	44/0.25	49/0.28	163/0.93	300/1.71	13685
[7]	161	1579	1058	3	2	1327	512/3.18	576/3.58	256/1.59	1856/11.53	15300
[9]	115	737	290	4	6	1621	474/4.12	602/5.23	1289/11.21	2839/24.68	40006
[13]	273	4619	4166	8	16	4875	84/0.31	84/0.31	56/0.21	308/1.14	5558
[14]	229	880	999	1.5	2	845	448/1.96	448/1.96	256/1.12	1600/7.00	5915
[16]	300	1154	1031	0	2	617	456/1.52	456/1.52	265/0.88	1633/5.44	3356
[17]	227	1005	559	0	2	522	448/1.97	448/1.97	256/1.13	1640/7.04	3675
This work	303	11061	10973	0	32	7337	58/0.19	58/0.19	29/0.09	203/0.68	4989

REFERENCES

[1] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring." Proceedings 35th annual symposium on foundations of computer science. IEEE, pp124–134, 1994.

[2] National Institute of Standards and Technology (2024) Module-Lattice-Based Key-Encapsulation Mechanism Standard. (Department of Commerce, Washington, D.C.), Federal Information Processing Standards Publication (FIPS) NIST FIPS 203.

[3] Yiming Huang, Miaoqing Huang, Zhongkui Lei and Jiaxuan Wu, "A pure hardware implementation of CRYSTALS-KYBER PQC algorithm through resource reuse. " IEICE Electron. Express 17 (2020): 20200234.

[4] A. C. Mert, E. Karabulut, E. Öztürk, E. Savaş and A. Aysu, "An Extensive Study of Flexible Design Methods for the Number Theoretic Transform," in IEEE Transactions on Computers, vol. 71, no. 11, pp 2829-2843, 2022.

[5] F. Yaman, A. C. Mert, E. Öztürk and E. Savaş, "A Hardware Accelerator for Polynomial Multiplication Operation of CRYSTALS-KYBER PQC Scheme," 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp1020-1025, 2021.

[6] Bin Li, Xiao-Jie Chen, Feng Feng and Qing-Lei Zhou, "FPGA multi-unit parallel optimization and implementation of post-quantum cryptography CRYSTALS-Kyber".Journal on Communications, vol. 43, no. 2, pp196-207, 2022.

[7] Yufei Xing and Shuguo Li, "A compact hardware implementation of CCA-secure key exchange mechanism CRYSTALS-KYBER on FPGA." IACR Transactions on Cryptographic Hardware and Embedded Systems, pp328-356, 2021.

[8] M. Bisheh-Niasar, R. Azarderakhsh and M. Mozaffari-Kermani, "High-Speed NTT-based Polynomial Multiplication Accelerator for Post-Quantum Cryptography," 2021 IEEE 28th Symposium on Computer Arithmetic (ARITH), Lyngby, Denmark, pp94-101, 2021.

[9] M. Bisheh-Niasar, R. Azarderakhsh and M. Mozaffari-Kermani, "Instruction-Set Accelerated Implementation of CRYSTALS-Kyber," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 68, no. 11, pp4648-4659, 2021.

[10] Shun-Sen Lü, Bin Li, Jia-Qi Zhai, Song-Qi Li and Qing-Lei Zhou, "FPGA efficient parallel optimization of Crystal-Kyber". Acta Electronica Sinica, pp1-11, 2024.

[11] Zhao-Hui Chen, Yuan Ma and Ji-Wu Jing, "Hardware optimization and evaluation for crucial modules of lattice-based cryptography." Acta Scientiarum Naturalium Universitatis Pekinensis, vol. 57, no. 4, pp595-604, 2021.

[12] Yang H, Chen R, Wang Q, et al, "Hardware acceleration of NTT-based polynomial multiplication in CRYSTALS-KYBER." International Conference on Information Security and Cryptology. Singapore: Springer Nature Singapore, pp111-129, 2023.

[13] M. Li, J. Tian, X. Hu and Z. Wang, "Reconfigurable and High-Efficiency Polynomial Multiplication Accelerator for CRYSTALS-Kyber," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 42, no. 8, pp2540-2551, 2023.

[14] V. B. Dang, K. Mohajerani and K. Gaj, "High-Speed Hardware Architectures and FPGA Benchmarking of CRYSTALS-Kyber, NTRU, and Saber," in IEEE Transactions on Computers, vol. 72, no. 2, pp306-320, 2023.

[15] G. Li, D. Chen, G. Mao, W. Dai, A. I. Sanka and R. C. C. Cheung, "Algorithm-Hardware Co-Design of Split-Radix Discrete Galois Transformation for KyberKEM," in IEEE Transactions on Emerging Topics in Computing, vol. 11, no. 4, pp824-838, 2023.

[16] Z. Ni, A. Khalid, W. Liu and M. O'Neill, "Towards a Lightweight CRYSTALS-Kyber in FPGAs: an Ultra-lightweight BRAM-free NTT Core," 2023 IEEE International Symposium on Circuits and Systems (ISCAS), Monterey, CA, USA, 2023, pp. 1-5.

[17] T H. Nguyen, D T. Dam, P P. Duong, B. Kieu-Do-Nguyen, C K. Pham and T T. Hoang, "Efficient Hardware Implementation of the Lightweight CRYSTALS-Kyber," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 72, no. 2, pp610-622, 2025.

[18] Prabhavathi Krishnegowda and Anandaraju M Boregowda, "Fully Homomorphic Encryption of Floating-Point Matrices for Privacy-Preserving Image Processing." IAENG International Journal of Computer Science, vol. 50, no. 4, pp1460-1469, 2023.

[19] 7 Series FPGAs Data Sheet:Overview.(2020).[Online].Available: https://www.xilinx.com/content/dam/xilinx/support/documents.