Efficient Polynomial Multiplier Architectures for ML-KEM on Resource-Constrained FPGAs

Hossein Naderi Varandi[a], Raziye Salarifard[a,*]

[a]*Faculty of Computer Science and Engineering, Shahid Beheshti University, Tehran, Iran*

Abstract

The Number Theoretic Transform (NTT) is a key component of polynomial multiplication in Kyber, making its efficient hardware implementation essential for post-quantum cryptographic applications. This paper presents two hardware architectures for Kyber polynomial arithmetic, supporting the NTT, inverse NTT (INTT) and point-wise multiplication (PWM) operations. The first architecture employs a single processing element (PE) to reduce hardware area, while the second uses two PEs to achieve higher throughput and efficiency. Both architectures incorporate a low-complexity modular multiplication and reduction method tailored to the Kyber modulus $q = 3329$, a unified butterfly unit supporting NTT, INTT, and PWM, and constant-geometry data access with dedicated address-generation algorithms. The dual-PE architecture is implemented without dedicated DSP blocks or BRAMs. FPGA implementation results demonstrate that the dual-PE architecture achieves the lowest area–time product and highest frequency among the evaluated state-of-the-art designs, while providing an effective trade-off between hardware area and execution time.

Keywords:
Kyber, ML-KEM, Number-theoretic transform, Polynomial multiplication, Post-quantum cryptography

*Corresponding author.

Email addresses: h_naderivarandi@sbu.ac.ir (Hossein Naderi Varandi), r_salarifard@sbu.ac.ir (Raziye Salarifard)

1. Introduction

Kyber is a post-quantum cryptographic key encapsulation mechanism (KEM) selected by NIST for standardization [1]. Its security relies on the hardness of the Module Learning with Errors (MLWE) problem, which is believed to be resistant against attacks from both classical and quantum computers [2]. As quantum computing advances, traditional public-key cryptosystems like RSA and ECC will become vulnerable; Kyber offers a practical, efficient, and quantum-safe alternative. It is already being deployed in real-world applications, including hybrid TLS connections and secure communication protocols.

The most computationally intensive operation in Kyber is polynomial multiplication in the ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$, where $n = 256$ and $q = 3329$. Kyber uses the NTT to perform this multiplication in $O(n \log n)$ time instead of $O(n^2)$ [2]. This process involves transforming the input polynomials into the NTT domain, performing PWM, and applying the INTT to obtain the resulting polynomial. The NTT-based multiplication accounts for over 80% of the total execution time in Kyber's key generation, encapsulation, and decapsulation routines. Therefore, efficient implementation of NTT multiplication while keeping resource usage low is critical for efficient hardware implementations of Kyber, especially in constrained environments such as IoT devices, smart cards, and embedded systems.

Edge devices in quantum-safe networks may need to support multiple concurrent Kyber sessions while operating under strict hardware-area constraints. Similarly, data-center accelerators require high throughput while maintaining area efficiency to control costs. These competing requirements are reflected in existing NTT architectures, which span single-PE designs with low area but limited throughput and highly parallel designs that achieve high throughput at the cost of greater hardware resources [3, 4, 5, 6, 7].

In this work, we address the area–performance trade-off in Kyber polynomial arithmetic by proposing two architectures for NTT, INTT, and PWM. The first is a low-area architecture based on a single PE, which supports all three operations while minimizing hardware resources. The second is a high-throughput architecture employing two PEs to achieve higher performance while maintaining efficient resource utilization. Both architectures incorporate two key innovations: (1) a low-complexity modular reduction method tailored to the Kyber modulus $q = 3329$, which reduces the required hardware resources and power consumption; and (2) a novel butterfly unit

that natively supports NTT, INTT, and PWM operations within a compact hardware structure. In addition, both designs employ constant-geometry data access to simplify address management and provide a regular and predictable memory-access pattern.

The main contributions of this work are summarized as follows:

- We propose a general low-complexity modular multiplication and reduction method for the Kyber modulus ($q = 3329$), which can be exploited across NTT, INTT, and PWM operations, enabling a hardware-efficient implementation with low computational complexity.
- We propose a novel butterfly unit that natively supports NTT, INTT, and PWM operations within a unified architecture while requiring low hardware area.
- We develop efficient address-generation and data-access algorithms based on the constant-geometry structure, simplifying address management and enabling regular memory-access patterns for NTT and INTT operations.
- We propose two complementary architectures for Kyber NTT/INTT/PWM operations: a single-PE architecture targeting low area and a dual-PE architecture targeting high throughput and efficiency. The dual-PE architecture is implemented without dedicated DSP blocks or BRAMs while supporting the complete set of required polynomial operations.

FPGA implementation results demonstrate that the proposed dual-PE architecture achieves the lowest area–time product (ATP) among the compared state-of-the-art designs, offering an efficient trade-off between hardware area and execution time.

The remainder of this paper is organized as follows. Section 2 reviews related work on Kyber NTT hardware architectures. Section 3 presents the preliminaries, including the Kyber polynomial multiplication, memory and data-access management, and PWM. Section 4 describes the two proposed architectures in detail. Section 5 presents the experimental setup, implementation results, and comparisons with previous work. Finally, Section 6 concludes the paper and outlines future research directions.

2. Previous Works

Modular arithmetic, particularly modular multiplication and reduction, plays a critical role in the computational efficiency of ML-KEM polynomial multiplication.

In the CRYSTALS-Kyber reference software implementation [8], Montgomery and Barrett reduction are employed for modular multiplication and reduction. Montgomery modular multiplication requires the operands to be represented in the Montgomery domain. In contrast, Barrett reduction approximates the quotient with respect to the modulus (q), thereby reducing the number of required subtraction operations. These reduction techniques have also been adopted in hardware implementations [9, 10, 11], where different optimization methods have been proposed specifically for Kyber.

Another effective approach to modular reduction in FPGA implementations is lookup-table (LUT)-based reduction, as employed in [12, 13]. In general, LUT-based reduction partitions the input into groups of bits, precomputes the modular remainders corresponding to all possible values of each group, and stores these values in lookup tables. During the reduction process, the bit groups are used as addresses for the LUTs, and the retrieved values are combined with the remaining bits to obtain the reduced result.

In addition to reviewing existing modular-reduction techniques, the work in [13] proposes a lookup-table (LUT)-based reduction method combined with K-RED. K-RED is applicable when the modulus can be expressed as $q = k \cdot 2^m + 1$ which implies $k \cdot 2^m \equiv -1 \pmod{q}$. This congruence can be exploited to simplify modular reduction. Applying K-RED twice results in the K^2-RED technique, which has been employed in [6, 14].

K-RED and K^2-RED are particularly effective in NTT and INTT computations because one operand of each modular multiplication, namely the twiddle factor, is known in advance. Therefore, the twiddle factors can be precomputed by multiplying them by K^{-1} or K^{-2}, respectively, to compensate for the scaling introduced by K-RED or K^2-RED. Consequently, the resulting products remain correctly represented modulo (q). However, this precomputation strategy cannot be directly applied to PWM, since both operands are generally unknown in advance and vary during execution.

In addition to the aforementioned methods, [7] employs a lazy-reduction strategy in which intermediate values are allowed to grow to 15 bits, while full modular correction is deferred until the final output stage. This approach reduces the number of reduction operations required during intermediate com-

putations. Furthermore, [3] proposes a Dadda-tree-based reduction method to accelerate the modular reduction process in hardware.

Previous designs can also be categorized according to the number of butterfly units or PEs employed. The selected degree of parallelism is typically determined by hardware-resource constraints and the requirements of the target application. The designs presented in [3, 4, 5, 15] employ a single PE to perform the forward NTT, INTT, and, in some cases, PWM. In contrast, [16] presents both single- and dual-butterfly architectures, whereas [17, 18] employ dual-parallel radix-2 butterfly PEs.

The polynomial-arithmetic architecture proposed in [9] supports both radix-2 and radix-4 PEs. At a much higher level of parallelism, [6] employs 128 PEs operating in parallel, enabling all 256 coefficients of an NTT stage to be processed within a single clock cycle. Furthermore, the architecture in [7] employs 896 PEs and reportedly achieves a throughput of one complete NTT per clock cycle.

Several existing designs employ constant-geometry NTT architectures [19, 20, 21, 22]. These architectures simplify memory addressing and control logic through regular butterfly interconnections and data-access patterns. However, depending on the implementation, these benefits may come at the cost of additional memory or data-reordering overhead.

Most of the aforementioned designs employ a unified PE to perform PWM. A straightforward implementation of each base-case multiplication requires five modular multiplications and two modular additions to generate a pair of output coefficients. To reduce the multiplication complexity, the architectures presented in [17, 9] employ a Karatsuba-based reformulation, which reduces the number of modular multiplications from five to four at the cost of additional modular additions and subtractions.

3. Preliminaries

3.1. Polynomial Multiplication

Polynomial multiplication is a fundamental operation in ML-KEM, particularly in polynomial-vector and polynomial-matrix arithmetic. As specified in NIST FIPS 203 [23], ML-KEM operates over the polynomial ring

$$R_q = \mathbb{Z}_q[x]/(x^{256} + 1), \tag{1}$$

where the polynomial length is $n = 256$ and the modulus is $q = 3329$. Each polynomial in R_q can be represented as

$$a(x) = \sum_{i=0}^{255} a_i x^i, \qquad a_i \in \mathbb{Z}_q. \tag{2}$$

To enable efficient polynomial multiplication, ML-KEM employs the NTT. The product of two polynomials $a, b \in R_q$ can be expressed as

$$c = \mathrm{INTT}\left(\mathrm{PWM}\left(\mathrm{NTT}(a), \mathrm{NTT}(b)\right)\right), \tag{3}$$

where c denotes the resulting polynomial, $\mathrm{NTT}(\cdot)$ and $\mathrm{INTT}(\cdot)$ denote the forward and inverse NTT operations, respectively, and $\mathrm{PWM}(\cdot, \cdot)$ denotes multiplication in the NTT domain.

3.2. Memory Organization and Data-Access Management

Two important considerations in NTT architecture design are memory organization and data-access geometry. Based on memory organization, NTT architectures can be classified as in-place or out-of-place. In an in-place NTT, intermediate results overwrite the input coefficients, reducing storage requirements. In contrast, an out-of-place NTT stores the results of each computational stage separately from its input data, typically using a ping-pong memory organization in which two memory buffers alternate between reading and writing.

Based on data-access geometry, NTT architectures can be classified as variable-geometry or constant-geometry. In variable-geometry architectures, butterfly connectivity and data-access patterns vary across computational stages. In contrast, constant-geometry architectures maintain fixed butterfly connectivity and regular data-access patterns across all stages.

An in-place variable-geometry NTT is commonly used in software implementations due to its relatively low memory requirements. For hardware implementations, an out-of-place constant-geometry NTT can simplify memory addressing and control logic by providing regular data-access patterns, at the cost of additional storage. In the proposed architectures, this additional memory capacity is also utilized during PWM computation, improving overall memory utilization.

3.3. Point-Wise Multiplication

In ML-KEM, multiplication in the NTT domain is not performed through simple coefficient-wise multiplication. Although $q = 3329$ admits a primitive 256th root of unity, it does not admit a primitive 512th root of unity. Consequently, multiplication in the NTT domain is performed through 128 independent base-case multiplications of degree-one polynomials [23]. The i-th base-case multiplication is expressed as

$$c_i(x) \equiv (a_0 + a_1 x)(b_0 + b_1 x) \pmod{x^2 - \gamma_i}, \tag{4}$$

where

$$\gamma_i = 17^{2\,\mathrm{BitRev}_7(i)+1} \bmod q, \quad 0 \le i < 128, \tag{5}$$

and $\mathrm{BitRev}_7(i)$ denotes the seven-bit reversal of i. Here, a_0, a_1, b_0, and b_1 denote the coefficients of the corresponding pair of degree-one polynomials in the NTT domain. The resulting coefficients are given by

$$\begin{aligned} c_0 &= (a_0 b_0 + \gamma_i a_1 b_1) \bmod q, \\ c_1 &= (a_0 b_1 + a_1 b_0) \bmod q. \end{aligned} \tag{6}$$

Therefore, an efficient polynomial multiplier architecture must account for the arithmetic and memory-access requirements of PWM in addition to those of the NTT and INTT.

4. Proposed Resource-Efficient ML-KEM Polynomial Multiplier Architectures

This work proposes two polynomial multiplier architectures for ML-KEM targeting resource-constrained FPGAs. The first architecture employs a single PE along with BRAM and DSP resources, whereas the second incorporates two PEs and eliminates the need for both BRAM and DSP resources. Both architectures follow a bottom-up design methodology, as illustrated in Fig. 1.

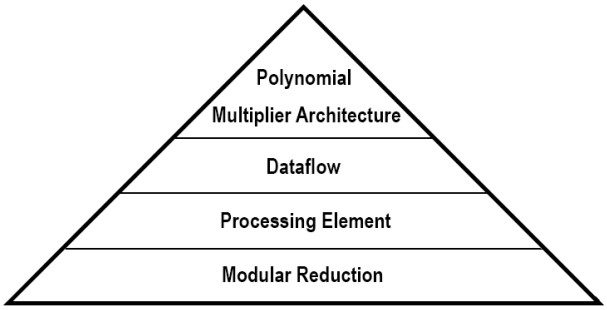

Figure 1: Bottom-up design methodology for the proposed polynomial multiplier architectures.

The NTT core consists of three fundamental arithmetic operations: modular addition, modular subtraction, and modular multiplication. Modular addition and subtraction can be implemented using simple arithmetic operations, followed by conditional subtraction or addition of (q) to ensure that the results remain within the range ($[0,q)$). In contrast, modular multiplication constitutes a critical computational component of the PE and can significantly affect its critical path delay. Therefore, we first propose a low-complexity modular reduction technique tailored to the modulus used in ML-KEM.

Building upon the proposed modular reduction technique, we develop an ultra-unified PE capable of supporting the NTT, INTT, and PWM. Unlike conventional coefficient-wise multiplication, which independently multiplies polynomial coefficients at corresponding indices, the PWM employed in ML-KEM requires additional arithmetic operations. Accordingly, the proposed PE is designed to efficiently accommodate these operations while maximizing hardware resource sharing.

At the architectural level, an efficient dataflow organization is essential for integrating the proposed PE into a complete polynomial multiplication architecture. The dataflow strategy for PWM is discussed in detail in the proposed BRAM- and DSP-free architecture. For the NTT and INTT, we adopt an out-of-place constant-geometry architecture, which provides regular and consistent memory read and write access patterns across computational stages. This regularity simplifies memory addressing and control logic at the cost of increased memory requirements, as the out-of-place approach requires two memory buffers. However, the additional memory capacity is also utilized during PWM computation, improving overall memory utilization. The

adopted out-of-place constant-geometry algorithms for the NTT and INTT are presented in Algorithms 1 and 2, respectively, based on the approaches reported in [20] and [22].

Algorithm 1 Out-of-place constant-geometry NTT for ML-KEM

Require: $A = (a_0, a_1, \ldots, a_{255}) \in \mathbb{Z}_q^{256}$; $q = 3329$; $\zeta[z] = 17^{\mathrm{BitRev}_7(z)} \bmod q$, $0 \leq z < 128$ [23]
Ensure: $\hat{A} = (\hat{a}_0, \hat{a}_1, \ldots, \hat{a}_{255}) \in \mathbb{Z}_q^{256}$

```
for i ← 0 to 6 do
    for j ← 0 to 127 do
        z ← 2^i + (j mod 2^i)
        t ← A[j + 128] · ζ[z] mod q
        Â[2j] ← (A[j] + t) mod q
        Â[2j + 1] ← (A[j] − t) mod q
    end for
    if i ≠ 6 then
        for k ← 0 to 255 do
            A[k] ← Â[k]
        end for
    end if
end for
return Â
```

4.1. Proposed General Modular Reduction Architecture

In this work, we propose a general-purpose modular reduction architecture that can be efficiently utilized for NTT, INTT, and PWM operations. To reduce an unsigned 24-bit number $C[23:0]$, obtained from multiplication, to a 12-bit unsigned value modulo $q = 3329$, the input is first decomposed as

$$C[23:0] = C[23:12] \cdot 2^{12} + C[11:0].$$

Since

$$2^{12} \bmod 3329 = 767 = 13 \times 59,$$

the input can be rewritten as

$$C[23:0] = (C[23:12] \times 59) \times 13 + C[11:0].$$

Algorithm 2 Out-of-place constant-geometry INTT for ML-KEM

Require: $\hat{A} = (\hat{a}_0, \hat{a}_1, \ldots, \hat{a}_{255}) \in \mathbb{Z}_q^{256}$; $q = 3329$; $\zeta[z] = 17^{\mathrm{BitRev}_7(z)} \bmod q$, $0 \leq z < 128$ [23]

Ensure: $A = (a_0, a_1, \ldots, a_{255}) \in \mathbb{Z}_q^{256}$

1: **for** $i \leftarrow 0$ **to** 6 **do**
2: **for** $j \leftarrow 0$ **to** 127 **do**
3: $z \leftarrow 2^{7-i} - 1 - (j \bmod 2^{6-i})$
4: $t \leftarrow (\hat{A}[2j+1] - \hat{A}[2j]) \bmod q$
5: $A[j] \leftarrow ((\hat{A}[2j] + \hat{A}[2j+1]) \cdot 2^{-1}) \bmod q$
6: $A[j+128] \leftarrow (t \cdot \zeta[z] \cdot 2^{-1}) \bmod q$
7: **end for**
8: **if** $i < 6$ **then**
9: **for** $k \leftarrow 0$ **to** 255 **do**
10: $\hat{A}[k] \leftarrow A[k]$
11: **end for**
12: **end if**
13: **end for**
14: **return** A

First, an intermediate value $d[17:0]$ is computed as

$$d = C[23:12] \times 59.$$

To eliminate the need for a dedicated multiplier, multiplication by 59 is implemented using its non-adjacent form (NAF) representation, as employed in our previous work [24]:

$$59 = 2^6 - 2^2 - 1.$$

Therefore,

$$d = C[23:12] \ll 6 - C[23:12] \ll 2 - C[23:12].$$

Next, based on the k-reduction approach, the intermediate value d is multiplied by $k = 13$. This operation can be efficiently implemented by decomposing d into its lower and upper parts:

$$e[12:0] = 13d[7:0] - d[17:8].$$

The resulting e is a 13-bit signed value within the range $[-943, 3315]$. After adding $C[11:0]$, the intermediate result lies within the range $[-943, 7410]$.

Finally, a correction step involving the appropriate multiples of the modulus, 3329, -3329, and -6658, maps the result to the desired range $[0, q)$. Overall, the proposed low-complexity reduction architecture requires only five subtraction and two addition operations, in addition to shift operations. The complete modular reduction procedure is summarized in Algorithm 3, and the corresponding hardware architecture is illustrated in Fig. 2.

Algorithm 3 Proposed General Modular Reduction

Require: $C[23:0]$
Ensure: $g = C \bmod q$, where $0 \leq g < q$
1: $d \leftarrow (C[23:12] \ll 6) - (C[23:12] \ll 2) - C[23:12]$
2: $e \leftarrow (d[7:0] \ll 4) - (d[7:0] \ll 1) - d[7:0]$
3: $f \leftarrow e - d[17:8] + C[11:0]$
4: $g \leftarrow Correct(f)$ ▷ Final correction (one addition)
5: **return** g

Furthermore, the proposed reduction architecture does not rely on FPGA-specific hardware resources, making it potentially suitable for efficient ASIC implementation. In our implementation, the reduction unit is pipelined to achieve a higher operating frequency. However, the number of pipeline stages differs between the two proposed NTT architectures to accommodate their respective target operating frequencies.

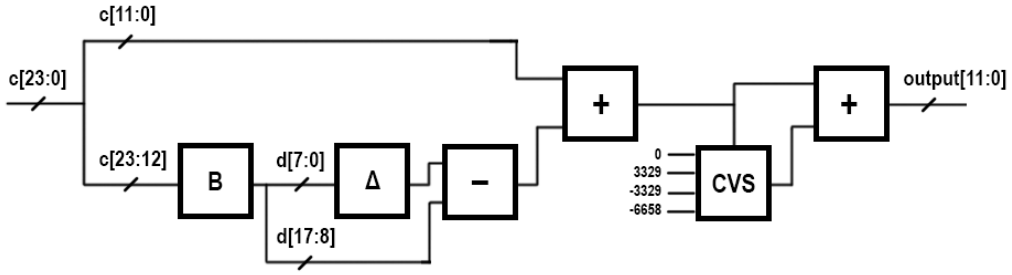

Figure 2: Hardware architecture of the proposed modular reduction unit, where B and Δ denote multiplication by 59 and 13, respectively, and CVS denotes the correction value selector.

4.2. Proposed Ultra-Unified Processing Element Architecture

The proposed PE employs a unified architecture capable of supporting NTT, INTT, and PWM operations through extensive hardware resource sharing. The PE supports three operating modes, and its corresponding hardware architecture is illustrated in Fig.3.

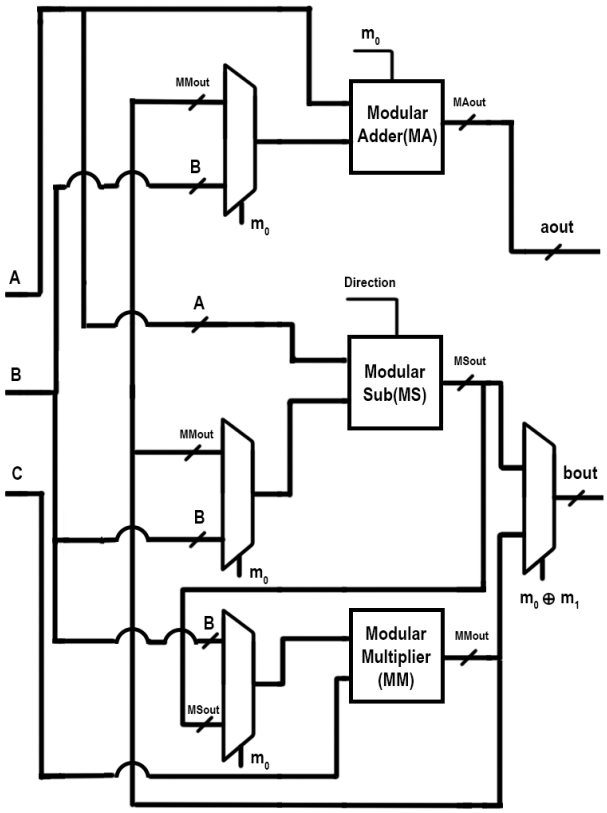

Figure 3: Hardware architecture of the proposed ultra-unified PE, where $m_1 m_0$ selects the operating mode: 00 for NTT, 01 for INTT, and 10 for PWM.

In NTT mode, the two outputs of the PE are computed as

$$A_{\text{out}} = A + B \cdot C, \tag{7}$$

$$B_{\text{out}} = A - B \cdot C. \tag{8}$$

The coefficient C is retrieved from a ROM containing the precomputed ζ values required for the NTT operation.

The INTT mode requires a slightly different computational approach. In the conventional Kyber INTT, with $q = 3329$, each coefficient is ultimately multiplied by 2^{-7}. Previous studies have proposed an optimization in which multiplication by 2^{-1} is incorporated into each butterfly stage. In this approach, the butterfly initially computes

$$A_{\text{out}} = (A + B), \tag{9}$$

$$B_{\text{out}} = (B - A) \cdot C, \tag{10}$$

after which both outputs are multiplied by 2^{-1}. Consequently, the required multiplication by 2^{-7} is achieved after seven butterfly stages, eliminating the need for a separate division-by-2^7 operation at the INTT output. Previous implementations have further integrated this scaling operation into the modular adder and modular multiplier. Since multiplication by 2^{-1} modulo q can be efficiently realized using an addition-based operation rather than a general-purpose multiplier, this technique reduces hardware overhead [25].

In the proposed PE, this optimization is also incorporated into the modular adder path. For the modular multiplier path, rather than explicitly multiplying the result by 2^{-1}, the ROM stores modified twiddle factors $2^{-1}\zeta$ instead of the conventional ζ values. Accordingly, the required scaling is absorbed into the multiplication operation, reducing the hardware complexity and area of the PE while preserving the correctness of the INTT computation. Therefore, in INTT mode, the PE outputs are expressed as

$$A_{\text{out}} = (A + B) \cdot 2^{-1}, \tag{11}$$

$$B_{\text{out}} = (B - A) \cdot C. \tag{12}$$

The third operating mode supports PWM. In this mode, the PE outputs are configured as

$$A_{\text{out}} = A + B \cdot C, \tag{13}$$

$$B_{\text{out}} = B \cdot C. \tag{14}$$

This configuration enables the PE to perform the modular addition and multiplication operations required for Kyber PWM without modifying its internal hardware structure. Consequently, the proposed ultra-unified PE supports NTT, INTT, and PWM operations while efficiently reusing the same computational resources, thereby minimizing hardware overhead.

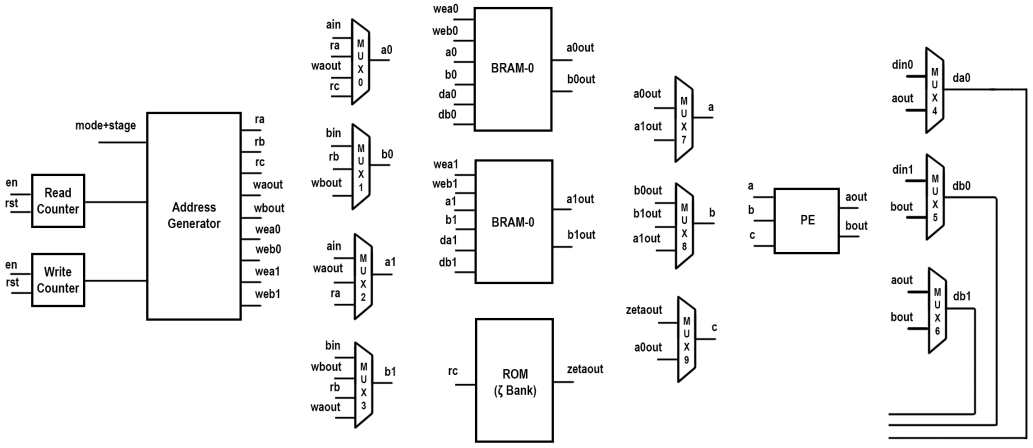

Figure 4: Hardware architecture of the proposed single-PE polynomial multiplier for ML-KEM.

The proposed PE employs a pipelined architecture to improve operating frequency. The two proposed polynomial multiplier architectures utilize different numbers of pipeline stages to meet their respective operating-frequency requirements.

4.3. Proposed Single-PE Polynomial Multiplication Architecture

The proposed NTT architecture employing a single PE is illustrated in Fig. 4. The modular multiplier utilizes one DSP block for the multiplication operation. Two true dual-port BRAMs are employed to store and access the polynomial coefficients during the computation.

The first memory, denoted as BRAM_0, has a capacity of 512×12 bits and serves as the main input/output (I/O) memory. For NTT and INTT operations, the first 256 addresses are used to store the input and output coefficients. For PWM operations, this memory is used to store the coefficients of the two input polynomials. During the NTT and INTT computations, BRAM_0 also serves as one of the two memories required by the ping-pong memory organization associated with the constant-geometry approach.

The second memory, denoted as BRAM_1, has a capacity of 256×12 bits. Unlike BRAM_0, BRAM_1 is dedicated to internal computation and is not directly accessible through the external I/O interface. It serves as the second memory in the ping-pong memory organization and is alternately used with BRAM_0 to store intermediate coefficients during the computation.

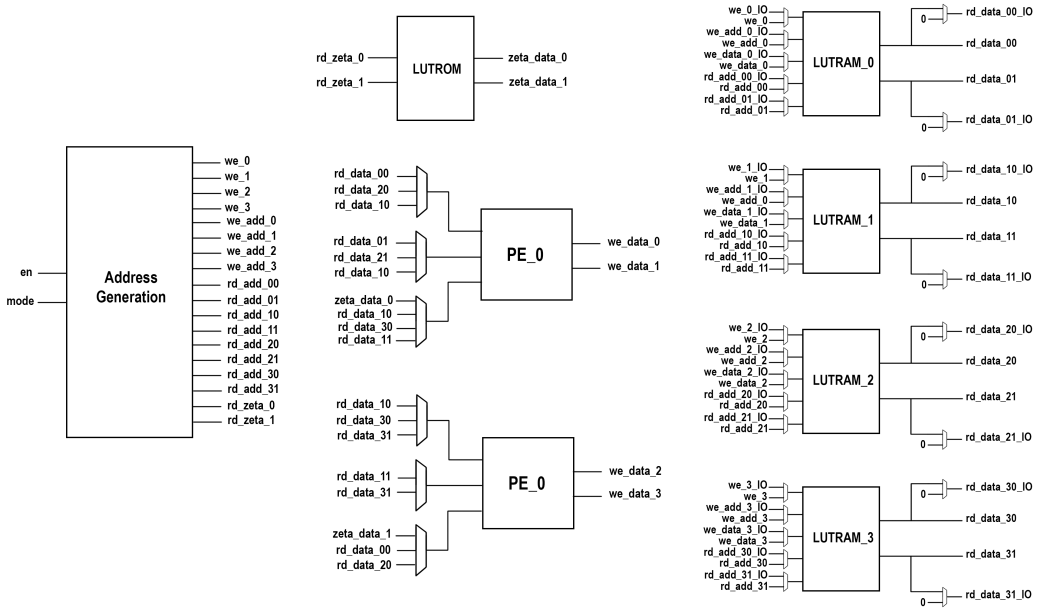

Figure 5: Hardware architecture of the proposed dual-PE polynomial multiplier for ML-KEM.

The use of two true dual-port BRAMs enables simultaneous memory accesses required by the constant-geometry NTT and INTT operations while allowing the single PE to process the coefficients sequentially.

4.4. Proposed Dual-PE Polynomial Multiplication Architecture

The proposed dual-PE architecture eliminates the need for both BRAM and DSP resources, as illustrated in Fig. 5. Instead of BRAMs, four LUT-based RAMs (LUTRAMs) are employed for polynomial coefficient storage. LUTRAM_0 and LUTRAM_2 each have a capacity of 128×12 bits, whereas LUTRAM_1 and LUTRAM_3 each have a capacity of 192×12 bits. The additional 64 addresses in LUTRAM_1 and LUTRAM_3 are reserved for storing intermediate data during computation.

Each LUTRAM supports one write port and up to two read ports, with the dual-read-port configuration requiring the duplication of LUT resources. The outputs of the two PEs are connected to the write-data ports of the LUTRAMs, allowing the intermediate results generated by the PEs to be stored directly in memory. The memory read and write patterns for the NTT, INTT, and PWM operations are described in the following subsections.

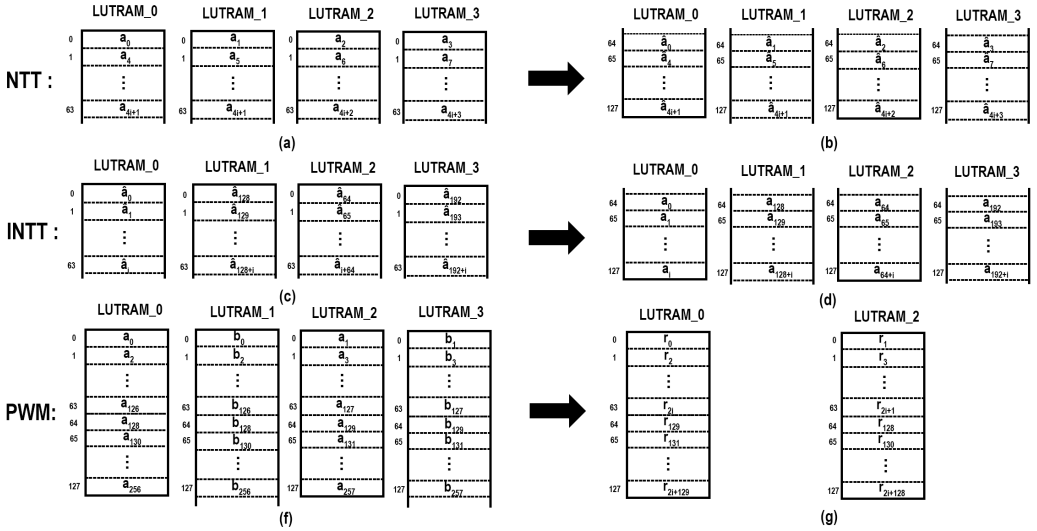

Figure 6: LUTRAM coefficient storage patterns for the proposed dual-PE architecture: (a) initial NTT coefficient placement, (b) final NTT output placement, (c) initial INTT coefficient placement, (d) final INTT output placement, (e) initial PWM coefficient placement, and (f) final PWM output placement.

4.4.1. NTT and INTT Dataflow

The memory read and write patterns for NTT and INTT operations are relatively straightforward, as both employ a ping-pong memory organization. The initial coefficient placement and final output locations for these operations are illustrated in Fig. 6. Specifically, Figs. 6(a) and (c) show the initial coefficient placement for NTT and INTT, respectively, while Figs. 6(b) and (d) illustrate the corresponding memory locations from which the transformed coefficients are read upon completion.

4.4.2. PWM Dataflow

In contrast to NTT and INTT, the PWM operation requires a more complex memory-access pattern, which is organized into five computational steps. Let A and B denote the two input polynomials. The initial coefficient placement for PWM is illustrated in Fig. 6(e). For each coefficient-pair index i, the computation proceeds as follows.

In Step 1, the two PEs compute

$$X_1 = A[2i + 128]B[2i + 128], \tag{15}$$

$$Y_1 = A[2i]B[2i + 128]. \tag{16}$$

In Step 2, the corresponding products are computed for the next coefficient pair:

$$X_2 = A[2i + 129]B[2i + 129], \tag{17}$$

$$Y_2 = A[2i + 1]B[2i + 129]. \tag{18}$$

During Steps 1 and 2, only the second output of each PE is utilized, as these computations require only modular multiplication.

In Step 3, X_1 and X_2 are multiplied by the corresponding constants γ_{2i} and $-\gamma_{2i}$, respectively, yielding

$$Z_1 = \gamma_{2i} X_1, \tag{19}$$

$$Z_2 = -\gamma_{2i} X_2. \tag{20}$$

Here, $\gamma_{2i+1} = -\gamma_{2i} \pmod q$, as defined by the ML-KEM base-case multiplication. Only the second output of each PE is utilized in this step.

In Step 4, the first half of the PWM outputs is computed as

$$O_1 = Y_1 + A[2i + 128]B[2i], \tag{21}$$

$$O_2 = Y_2 + A[2i + 129]B[2i + 1]. \tag{22}$$

Finally, in Step 5, the remaining outputs are computed as

$$O_3 = Z_1 + A[2i]B[2i], \tag{23}$$

$$O_4 = Z_2 + A[2i + 1]B[2i + 1]. \tag{24}$$

Steps 4 and 5 utilize the first output of each PE, which performs the required modular addition. By appropriately scheduling memory accesses and reusing the computational resources of both PEs, the proposed dual-PE architecture performs the complete PWM operation without requiring additional arithmetic units. The final placement of the PWM output coefficients is illustrated in Fig. 6(f).

Compared with the previously presented single-PE architecture, the proposed dual-PE architecture reduces the total memory capacity required for polynomial multiplication. The single-PE architecture employs a total memory capacity of 768×12 bits, including the memory allocated for intermediate computations, whereas the proposed dual-PE architecture requires only 640×12 bits. In addition to eliminating BRAM usage, the proposed memory organization enables parallel coefficient access for both PEs while preserving the regular memory-access patterns required by the constant-geometry NTT and INTT algorithms.

Table 1: Comparison of FPGA Resource Utilization and Performance

Work	LUT	FF	DSP	BRAM	CC(NTT/INTT/PWM)	Frequency	PWM	ATP
This work (a)	1763	2763	0	0	483/481/355	436	✓	8.81
This work (b)	427	468	1	1.5	987/988/706	210	✓	26.56
[26]	1154	1031	2	0	456/456/265	300	✓	9.38
[27]	1170	1164	4	2	235/235/138	303	✓	9.60
[17]	1005	599	2	0	448/448/256	227	✓	10.38
[27]	2283	2095	8	4	140/140/88	282	✓	12.07
[27]	4619	4166	16	8	84/84/56	273	✓	15
[21]	650	130	0	1.5	924/924/643	250	✓	16.14
[16]	1052	1058	0	13.5	234/235/169	334	✓	21.68
[16]	590	671	0	7.5	458/458/328	342	✓	23.18
[3]	2543	792	4	9	232/233/864	182	✓	37.41
[3]	948	352	1	2.5	904/904/3359	190	✓	41.91
[16]	407	411	0	7.5	906/906/649	350	✓	42.79
[28]	360	145	3	2	940/1203/1289	115	✓	66.05

$\mathrm{ATP} = \frac{\mathrm{Slice} \times \mathrm{NTT}_{CC}}{100 \times \mathrm{Frequency}}, \quad \mathrm{Slice} = 0.25\,\mathrm{LUT} + 0.125\,\mathrm{FF} + 100\,\mathrm{DSP} + 200\,\mathrm{BRAM}$

5. Results

The synthesis results of the proposed architectures are presented in Table 1, where This Work (a) and This Work (b) denote the proposed dual-PE and single-PE architectures, respectively. Both architectures target the Xilinx Artix-7 FPGA and were synthesized using Vivado 2023.2 with default synthesis settings. The operating frequencies were estimated through post-synthesis timing analysis. The functional correctness of both architectures was verified by comparing their output coefficients with reference results obtained using SageMath.

Both proposed architectures employ an out-of-place constant-geometry approach, which has also been adopted in [21]. The architecture TW(b) is similar to the architecture presented in this work. The main advantage of the constant-geometry approach is its simplified and regular read/write pattern for both NTT and INTT operations. However, the out-of-place implementation requires a ping-pong memory organization, resulting in higher memory usage. In the context of polynomial multiplication (PWM), this additional memory can be beneficial because the operation involves two polynomials rather than processing a single polynomial. In particular, the PWM operation in Kyber is not simply a coefficient-wise multiplication between coefficients with the same index. Therefore, the additional memory required by the constant-geometry architecture can be utilized to support the data-storage

requirements of the PWM operation.

Additionally, This Work (b) achieves one of the lowest LUT utilizations among the compared implementations, along with [21], [16], and [28]. The design reported in [21] requires slightly more LUTs than This Work (b), while being DSP-free and using the same amount of BRAM. The implementation in [16] achieves lower LUT utilization and is also DSP-free; however, it requires significantly more BRAM resources. It also operates at a higher frequency than This Work (b). Similarly, [28] reports the lowest LUT utilization among the compared designs, but requires more DSP and BRAM resources and operates at a lower frequency than the proposed implementation.

In contrast, This Work (a) is both DSP- and BRAM-free and employs two PEs instead of one. It uses four LUTRAM. in place of BRAMs, with the output of each butterfly (BF) directly connected to the write port of the corresponding LUTRAM. Among the compared implementations, [26] achieves an ATP value closest to that of the proposed designs. This implementation is also BRAM-free and employs three FIFOs for coefficient storage.

Although both proposed architectures employ the same modular reduction approach, they differ in pipeline depth and the number of PEs. This Work (a) employs a deeper pipeline to achieve a higher operating frequency, resulting in greater flip-flop (FF) utilization than This Work (b). Furthermore, the use of two PEs enables parallel butterfly computations, reducing the computational cycle count. Consequently, This Work (a) achieves a lower ATP than This Work (b) and all other implementations included in the comparison table.

6. Conclusion

This paper presented two FPGA-based architectures for Kyber polynomial arithmetic, supporting NTT, INTT, and PWM. The single-PE architecture targets low hardware area, whereas the dual-PE architecture targets higher throughput. Both designs incorporate a low-complexity modular multiplication and reduction method for $q = 3329$, a unified butterfly unit, and constant-geometry data-access mechanisms with dedicated address-generation algorithms. The dual-PE architecture operates without dedicated DSP blocks or BRAMs. FPGA evaluation shows that it achieves the lowest area–time product and the highest operating frequency among the evaluated designs, demonstrating a competitive balance between hardware cost and

performance. These results highlight the potential of the proposed architectures for efficient hardware implementation of Kyber polynomial arithmetic.

Acknowledgment

This research was financially supported by the Iran National Science Foundation (INSF) under project No. 40403698. The authors would like to thank INFS for this support.

References

[1] National Institute of Standards and Technology, Module-Lattice-Based Key-Encapsulation Mechanism Standard, Tech. Rep. FIPS 203, National Institute of Standards and Technology (Aug. 2024). doi:10.6028/NIST.FIPS.203.

[2] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, D. Stehlé, CRYSTALS-Kyber: Algorithm Specifications and Supporting Documentation (Version 3.02), Tech. rep., NIST Post-Quantum Cryptography Standardization Project, round-3 submission (Aug. 2021).
URL `https://pq-crystals.org/kyber/data/kyber-specification-round3-20210804.pdf`

[3] F. Yaman, A. C. Mert, E. Öztürk, E. Savaş, A hardware accelerator for polynomial multiplication operation of CRYSTALS-KYBER PQC scheme, Cryptology ePrint Archive, Paper 2021/485 (2021).

[4] K. Javeed, D. Gregg, Efficient number theoretic transform architecture for CRYSTALS-kyber, IEEE Trans. Circuits Syst. II Express Briefs 72 (1) (2025) 263–267.

[5] M. O. Alshomrany, M. Al-Hashimi, F. Alsolami, M. Saleh, O. Abulnaja, T. F. Al-Somani, A. Altuwaijri, Lightweight FPGA-based number-theoretic transform for ML-KEM in post-quantum cryptography, IEEE Access 13 (2025) 203620–203632.

[6] B. Taghavi, R. Azarderakhsh, M. Mozaffari Kermani, ParallelNTT: Maximizing performance of forward and inverse NTT on FPGA for ML-DSA and ML-KEM, in: Proceedings of the Great Lakes Symposium on VLSI 2025, ACM, New York, NY, USA, 2025, pp. 372–378.

[7] J. Bertels, I. Verbauwhede, A high throughput kyber NTT, Cryptology ePrint Archive, Paper 2025/1763 (2025).
URL https://eprint.iacr.org/2025/1763

[8] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, D. Stehlé, CRYSTALS-Kyber: Algorithm specifications and supporting documentation (version 3.02), Round-3 submission to the NIST Post-Quantum Cryptography Standardization Project, version 3.02, dated August 4, 2021 (Aug. 2021).
URL https://pq-crystals.org/kyber/data/kyber-specification-round3-20210804.pdf

[9] W. Guo, S. Li, Highly-efficient hardware architecture for CRYSTALS-kyber with a novel conflict-free memory access pattern, IEEE Trans. Circuits Syst. I Regul. Pap. 70 (11) (2023) 4505–4515.

[10] D. E. S. Kundi, J. M. B. Mera, P.-Y. Strub, M. Hutter, High-performance NTT hardware accelerator to support ML-KEM and ML-DSA, in: Proceedings of the 2024 Workshop on Attacks and Solutions in Hardware Security, ACM, New York, NY, USA, 2024, pp. 100–105.

[11] B. Taghavi, R. Azarderakhsh, M. M. Kermani, KNightCore: An ultra-lightweight NTT-based polynomial multiplier for ML-KEM on resource-constrained platforms, in: Proceedings of the 2025 1st ACM CCS Workshop on Quantum-Resistant Cryptography and Security, ACM, New York, NY, USA, 2025, pp. 73–77.

[12] K. Javeed, D. Gregg, AT-NTT: Area-time efficient polynomial multiplication architecture for CRYSTALS-Kyber, Comput. Electr. Eng. 133 (111069) (2026) 111069.

[13] J. Bertels, Q. Norga, I. Verbauwhede, A better kyber butterfly for FPGAs, Cryptology ePrint Archive, Paper 2024/1367 (2024).
URL https://eprint.iacr.org/2024/1367

[14] M. Bisheh-Niasar, R. Azarderakhsh, M. Mozaffari-Kermani, Instruction-set accelerated implementation of CRYSTALS-kyber, IEEE Trans. Circuits Syst. I Regul. Pap. 68 (11) (2021) 4648–4659.

[15] L. Li, G. Qin, Y. Yu, W. Wang, Compact instruction set extensions for kyber, IEEE Trans. Comput.-aided Des. Integr. Circuits Syst. 43 (3) (2024) 756–760.

[16] A. Alhassani, M. Benaissa, High-speed polynomials multiplication HW accelerator for CRYSTALS-kyber, IEEE Trans. Circuits Syst. I Regul. Pap. 71 (12) (2024) 6105–6113.

[17] T.-H. Nguyen, D.-T. Dam, P.-P. Duong, B. Kieu-Do-Nguyen, C.-K. Pham, T.-T. Hoang, Efficient hardware implementation of the lightweight CRYSTALS-kyber, IEEE Trans. Circuits Syst. I Regul. Pap. 72 (2) (2025) 610–622.

[18] Z. Ni, A. Khalid, W. Liu, M. O'Neill, Towards a lightweight CRYSTALS-kyber in FPGAs: An ultra-lightweight BRAM-free NTT core, in: 2023 IEEE International Symposium on Circuits and Systems (ISCAS), IEEE, 2023.

[19] U. Banerjee, T. S. Ukyab, A. P. Chandrakasan, Sapphire: A configurable crypto-processor for post-quantum lattice-based protocols (2019).

[20] B. Taghavi, R. Azarderakhsh, M. Mozaffari Kermani, ParallelNTT: Maximizing performance of forward and inverse NTT on FPGA for ML-DSA and ML-KEM, in: Proceedings of the Great Lakes Symposium on VLSI 2025, ACM, New York, NY, USA, 2025, pp. 372–378.

[21] B. Taghavi, R. Azarderakhsh, M. M. Kermani, KNightCore: An ultra-lightweight NTT-based polynomial multiplier for ML-KEM on resource-constrained platforms, in: Proceedings of the 2025 1st ACM CCS Workshop on Quantum-Resistant Cryptography and Security, ACM, New York, NY, USA, 2025, pp. 73–77.

[22] H. Chen, H.-W. Chen, S.-H. Huang, P.-Y. Chen, A high-performance hardware design for polynomial multiplication in the CRYSTALS-kyber algorithm, in: 2025 IEEE Symposium on Low-Power and High-Speed Chips and Systems (COOL CHIPS), IEEE, 2025, pp. 1–6.

[23] G. M. D. Nist, Module-lattice-based key-encapsulation mechanism standard, Tech. rep., Gaithersburg, MD (2024).

[24] H. Naderi Varandi, R. Salarifard, A lightweight general modular multiplier for kyber pqc, The ISC International Journal of Information Security (Online First) (Mar. 2025). doi:10.22042/isecure.2025.217886.
URL https://doi.org/10.22042/isecure.2025.217886

[25] Q. D. Truong, H. Lee, Efficient polynomial arithmetic and hash modules for ML-DSA and ML-KEM standards, in: 2024 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS), IEEE, 2024, pp. 776–780.

[26] Z. Ni, A. Khalid, W. Liu, M. O'Neill, Towards a lightweight CRYSTALS-kyber in FPGAs: An ultra-lightweight BRAM-free NTT core, in: 2023 IEEE International Symposium on Circuits and Systems (ISCAS), IEEE, 2023.

[27] M. Li, J. Tian, X. Hu, Z. Wang, Reconfigurable and high-efficiency polynomial multiplication accelerator for CRYSTALS-kyber, IEEE Trans. Comput.-aided Des. Integr. Circuits Syst. 42 (8) (2023) 2540–2551.

[28] M. Bisheh-Niasar, R. Azarderakhsh, M. Mozaffari-Kermani, Instruction-set accelerated implementation of CRYSTALS-kyber, IEEE Trans. Circuits Syst. I Regul. Pap. 68 (11) (2021) 4648–4659.