A Hardware Accelerator for Polynomial Multiplication Operation of CRYSTALS-KYBER PQC Scheme

Ferhat Yaman, Ahmet Can Mert, Erdinç Öztürk, Erkay Savaş
Faculty of Engineering and Natural Sciences, Sabanci University, Istanbul, Turkey
{ferhatyaman, ahmetcanmert, erdinco, erkays}@sabanciuniv.edu

***Abstract*—Polynomial multiplication is one of the most time-consuming operations utilized in lattice-based post-quantum cryptography (PQC) schemes. CRYSTALS-KYBER is a lattice-based key encapsulation mechanism (KEM) and it was recently announced as one of the four finalists at round three in NIST's PQC Standardization. Therefore, efficient implementations of polynomial multiplication operation are crucial for high-performance CRYSTALS-KYBER applications. In this paper, we propose three different hardware architectures (lightweight, balanced, high-performance) that implement the NTT, Inverse NTT (INTT) and polynomial multiplication operations for the CRYSTALS-KYBER scheme. The proposed architectures include a unified butterfly structure for optimizing polynomial multiplication and can be utilized for accelerating the key generation, encryption and decryption operations of CRYSTALS-KYBER. Our high-performance hardware with 16 butterfly units shows up to 112×, 132× and 109× improved performance for NTT, INTT and polynomial multiplication, respectively, compared to the high-speed software implementations on Cortex-M4.**

***Index Terms*—CRYSTALS-KYBER, PQC, NTT, Polynomial Multiplication, Hardware**

I. Introduction

Lattice-based cryptography has already gained great interest and it forms the mathematical basis for many different applications such as post-quantum key-encapsulation mechanisms (KEMs), post-quantum signature protocols and homomorphic encryption [1], [2]. National Institute of Standards and Technology (NIST) has started a post-quantum cryptography standardization process in 2016 and many lattice-based post-quantum schemes are proposed since then. NIST recently announced the finalists at the round three of the standardization process and the lattice-based KEM CRYSTALS-KYBER (Kyber) is one of the four finalists.

Lattice-based cryptosystems work with polynomial rings and perform costly polynomial arithmetic; multiplication of two large-degree polynomials, in particular. Schoolbook polynomial multiplication method is inefficient for implementing polynomial multiplication operations and it has $\mathbf{O}(n^2)$ complexity. Number theoretic transform (NTT) reduces $\mathbf{O}(n^2)$ complexity to quasi-linear complexity and, therefore, it is utilized in many lattice-based cryptosystems suffering from high complexity of polynomial arithmetic [1], [2], [3], [4], [5]. There are many works in the literature targeting efficient implementations of main arithmetic blocks of the post-quantum cryptosystems for software [6], [7], [8] and hardware [9], [10], [11], [12], [13], [14] platforms.

Key generation, encryption and decryption operations of Kyber scheme also use polynomial multiplication operation and NTT-based polynomial multiplication are utilized for efficient implementation of these operations. The team of Kyber adopted the technique proposed by Seiler *et al.* [15] and reduced the parameter q of Kyber from 7681 to 3329. This changed the definitions of NTT, Inverse NTT (INTT) and coefficient-wise multiplication operations. To the best of our knowledge, there are three hardware [13], [14], [10] and one software [7] implementations proposed for the NTT/INTT and polynomial multiplication operations of the Kyber with the new parameters and operation definitions. Our proposed hardware architecture outperforms the works in [7], [10] and [13].

In this work, we propose three different hardware architectures (lightweight, balanced, high-performance) performing NTT/INTT and polynomial multiplication operations for the new parameter set of Kyber[1]. The proposed architectures utilize a unified butterfly structure, which can be used for both NTT and INTT operations. The lightweight, balanced and high-performance hardware architectures use one, four and sixteen butterfly units, respectively. They can be used to accelerate key generation, encryption and decryption operations of Kyber.

The rest of the paper is organized as follows. Section II introduces preliminaries. Section III presents the hardware architectures with optimizations. Section IV presents the implementation results and compares the results with prior work, and Section V concludes the paper.

II. Preliminaries

In this section, we present the notation we follow throughout the paper, a brief definition of Kyber scheme and its arithmetic operations.

A. Notation

Let the ring $\mathbb{Z}_q$ represent the integers $\{0, 1, \ldots, q-1\}$. Let the polynomial ring $\mathbf{R}_q = \mathbb{Z}_q[x]/\phi(x)$ represent the

F. Yaman, E. Öztürk and E. Savaş are supported by TUBITAK under Grant Number 118E725.

[1]Code is available at https://github.com/acmert/kyber-polmul-hw

polynomials reduced with $\phi(x)$ with coefficients in $\mathbb{Z}_q$. For example, when $\phi(x)$ is form of x^n+1, $\mathbf{R}_q = \mathbb{Z}_q[x]/(x^n+1)$ represents the polynomials with degree of at most $n-1$.

Throughout the paper, we represent a polynomial, a column vector of polynomials and a matrix of polynomials with regular lowercase (e.g. v), bolded lowercase (e.g. $\mathbf{v}$) and bolded uppercase letters (e.g. $\mathbf{V}$), respectively. Let $\mathbf{v}^T$ and $\mathbf{V}^T$ represent the transpose of a vector of polynomials $\mathbf{v}$ and a matrix of polynomials $\mathbf{V}$, respectively. Coefficients of an n element vector a (or polynomial $a(x)$) are represented with a_i where i represents the position of the coefficient starting from 0. Polynomials in NTT domain are represented with a line over their names. For example, $\overline{a}(x)$ represents NTT domain representation of polynomial $a(x)$. Let $a \leftarrow \mathbf{R}_q$ and $a \overset{\$}{\leftarrow} \mathbf{R}_q$ represent that a is sampled uniformly from $\mathbf{R}_q$ and from centered binomial distribution, respectively. Let $\cdot$, $\odot$ and $\circ$ represent integer, coefficient-wise and matrix multiplications, respectively. For the rest of the paper, q and n represent the coefficient modulus and the degree of the polynomial ring, which are 3329 and 256 for Kyber, respectively.

B. NTT-based Polynomial Multiplication

NTT is the discrete Fourier transform defined over the ring $\mathbb{Z}_q$. An n-point(pt) NTT operation transforms an n element vector $a = [a_0, a_1, \ldots, a_{n-1}]$ (or $(n-1)$ degree polynomial $a(x) = \sum_{i=0}^{n-1} a_i x^i$) to another n element vector $\overline{a} = [\overline{a}_0, \overline{a}_1, \ldots, \overline{a}_{n-1}]$ (or $(n-1)$ degree polynomial $\overline{a}(x) = \sum_{i=0}^{n-1} \overline{a}_i x^i$) using

$$\bar{a}_i = \sum_{j=0}^{n-1} a_j \omega^{ij} \pmod q \text{ for } i = 0, 1, \ldots, n-1.$$

The NTT operation uses a constant called twiddle factor, $\omega \in \mathbb{Z}_q$, which is n-th root of unity. The twiddle factor satisfies the conditions $\omega^n \equiv 1 \pmod q$ and $\omega^i \neq 1 \pmod q$ $\forall i < n$, where $q \equiv 1 \pmod n$. Similarly, INTT operation uses the same formula as NTT operation with $\omega^{-1} \pmod q$ and it also requires the resulting coefficients to be multiplied with n^{-1} $(\bmod\ q)$ in $\mathbb{Z}_q$. There are mainly two well-known approaches for NTT operation: *decimation in time* and *decimation in frequency*. The former and latter NTT operations utilize Cooley-Tukey (CT) and Gentleman-Sande (GS) butterfly structures, respectively [16], [17].

NTT/INTT operations convert schoolbook polynomial multiplication operation into coefficient-wise multiplication (CWM) and reduce the complexity of polynomial multiplication operation. When $\phi(x)$ is not in any special form, multiplication of polynomials $a(x)$ and $b(x)$ in $\mathbf{R}_q$ requires doubling the sizes of inputs with zero-padding and a separate polynomial reduction operation by $\phi(x)$ as shown in Eqn. 1 where $\mathtt{NTT}_{2n}$ and $\mathtt{INTT}_{2n}$ represent $2n$-pt NTT and $2n$-pt INTT operations, respectively.

$$c = \mathtt{INTT}_{2n}(\mathtt{NTT}_{2n}(a) \odot \mathtt{NTT}_{2n}(b)) \bmod \phi(x) \qquad (1)$$

When $\phi(x)$ is form of x^n+1, a technique called *negative wrapped convolution* is utilized which eliminates the

Algorithm 1 Forward In-place NTT Algorithm

```
Input: $a(x) \in \mathbb{Z}_q[x]/(x^n+1)$ in standard-order
Input: $\omega_n \in \mathbb{Z}_q$ (primitive $n$-th root of unity)
Input: $n = 2^l$, $q$ (s.t. $q \equiv 1 \mod n$)
Output: $\overline{a}(x) \in \mathbb{Z}_q[x]/(x^n+1)$ in bit-reversed order
 1: $k \leftarrow 1$
 2: for $i$ from 1 by 1 to $l-1$ do
 3:     $m \leftarrow 2^{l-i}$
 4:     for $s$ from 0 by $m$ to $n$ do
 5:         for $j$ from $s$ by 1 to $s+m$ do
 6:             $A, B, W \leftarrow a[j], a[m+j], \omega^{br_{l-1}(k)} \bmod q$
 7:             $T \leftarrow (W \cdot B) \bmod q$
 8:             $E, O \leftarrow (A+T) \bmod q, (A-T) \bmod q$
 9:             $a[j], a[j+m] \leftarrow E, O$
10:         end for
11:         $k \leftarrow k+1$
12:     end for
13: end for
```

need for doubling the input sizes and separated polynomial reduction for polynomial multiplication operations in $\mathbf{R_q}$ [18]. However, this requires the coefficients of input and output polynomials to be multiplied with $[\Psi^0, \Psi^1, \ldots, \Psi^{(n-1)}]$ and $[\Psi^0, \Psi^{-1}, \ldots, \Psi^{-(n-1)}]$, respectively. These operations are called pre-processing and post-processing, respectively. The constant Ψ is $2n$-th root of unity satisfying the conditions $\Psi^{2n} \equiv 1 \pmod q$ and $\Psi^i \neq 1 \pmod q$ $\forall i < 2n$, where $q \equiv 1 \pmod{2n}$.

In [15], Seiler *et al.* proposes a variant of NTT operation which enables efficient polynomial multiplication in $\mathbf{R}_q^n$ with satisfying only $q \equiv 1 \pmod n$ and without requiring pre-processing and post-processing operations. This technique is adopted by Kyber and their parameter q is reduced from 7681 to 3329. The NTT, INTT and CWM operations of Kyber are also changed accordingly [2]. This new variant of NTT operation generates 128 degree-2 polynomials, different from original NTT operation. Similarly, new INTT operation takes 128 degree-2 polynomials as input. Since the outputs of NTT operation are 128 degree-2 polynomials, CWM operation is performed as the multiplications of two degree-2 polynomials in $\mathbb{Z}_q[x]/(x^2 - \omega^i)$ where i changes according to index of coefficients. Algorithms for NTT, INTT and CWM operations of Kyber are shown in Algorithm 1, 2 and 3, respectively, where $br_{l-1}(\cdot)$ represents bit-reversal operation with bit size of $l-1$.

This new variant of NTT/INTT operations are represented as $\mathcal{NTT}/\mathcal{INTT}$ while original NTT/INTT operations are represented with $\mathtt{NTT/INTT}$. The polynomial multiplication operation for Kyber with new NTT definition is shown in Eqn. 2.

$$c = \mathcal{INTT}_n(\mathtt{CWM}(\mathcal{NTT}_n(a), \mathcal{NTT}_n(b))) \qquad (2)$$

C. CRYSTALS-KYBER

Kyber is a KEM transformed from public-key encryption scheme and it is proposed for NIST's post-quantum standard-

Algorithm 2 Inverse In-place NTT Algorithm

```
Input: $\overline{a}(x) \in \mathbb{Z}_q[x]/(x^n+1)$ in bit-reversed order
Input: $\omega_n^{-1} \in \mathbb{Z}_q$ (inverse of primitive $n$-th root of unity)
Input: $n = 2^l$, $q$ (s.t. $q \equiv 1 \mod n$)
Output: $a(x) \in \mathbb{Z}_q[x]/(x^n+1)$ in standard-order
1: $k \leftarrow 0$
2: for $i$ from $l-1$ by $-1$ to 1 do
3:     $m \leftarrow 2^{l-i}$
4:     for $s$ from 0 by $m$ to $2^l$ do
5:         for $j$ from $s$ by 1 to $s+m$ do
6:             $A, B, W \leftarrow \overline{a}[j], \overline{a}[j+m], \omega^{br_{l-1}(k)+1} \bmod q$
7:             $E, O \leftarrow (A+B) \bmod q, (A-B) \cdot W \bmod q$
8:             $\overline{a}[j], \overline{a}[j+m] \leftarrow \texttt{DIVby2}(E), \texttt{DIVby2}(O)$
9:         end for
10:        $k \leftarrow k+1$
11:    end for
12: end for
```

Algorithm 3 Coefficient-wise Multiplication Algorithm

```
Input: $\overline{a}(x), \overline{b}(x) \in \mathbb{Z}_q[x]/(x^n+1)$ in bit-reversed order
Input: $\omega \in \mathbb{Z}_q$ (primitive $n$-th root of unity)
Output: $\overline{c}(x) \in \mathbb{Z}_q[x]/(x^n+1)$ in bit-reversed order
1: for $i$ from 0 by 1 to $2^{l-1}$ do
2:     $W \leftarrow \omega^{br_{l-1}(i)+1} \bmod q$
3:     $a_0, a_1 \leftarrow \overline{a}[2i], \overline{a}[2i+1]$
4:     $b_0, b_1 \leftarrow \overline{b}[2i], \overline{b}[2i+1]$
5:     $\overline{c}[2i] \leftarrow (a_0 \cdot b_1 + a_1 \cdot b_0) \bmod q$
6:     $\overline{c}[2i+1] \leftarrow (a_1 \cdot b_1 \cdot W + a_0 \cdot b_0) \bmod q$
7: end for
```

ization process [2]. Kyber algorithm works with polynomial ring $\mathbf{R}_q$ where $\phi(x)$, q and n are x^n+1, 3329 and 256, respectively. It specifies its key generation, encryption and decryption operations as follow:

- **Key Generation:** For $\overline{\mathbf{A}} \leftarrow \mathbf{R}_q^{k\times k}$ and $\mathbf{s}, \mathbf{e} \xleftarrow{\$} \mathbf{R}_q^k$,
$$(pk, sk) = (\overline{\mathbf{A}} \circ \mathcal{NTT}(s) + \mathcal{NTT}(e), \mathcal{NTT}(s))$$
- **Encryption:** For $\overline{\mathbf{A}} \leftarrow \mathbf{R}_q^{k\times k}$, $\mathbf{r}, \mathbf{e_1} \xleftarrow{\$} \mathbf{R}_q^k$, $e_2 \xleftarrow{\$} \mathbf{R}_q$, pk and $m \in \mathbf{R}_q$,
$$ct = (\mathbf{u}, v) = (\mathcal{INTT}(\overline{\mathbf{A}}^T \circ \mathcal{NTT}(\mathbf{r})) + \mathbf{e_1}, \mathcal{INTT}(pk^T \circ \mathcal{NTT}(\mathbf{r})) + e_2 + m)$$
- **Decryption:** For $sk = \overline{\mathbf{s}}$ and $ct = (\mathbf{u}, v)$,
$$m = v - \mathcal{INTT}(sk^T \circ \mathcal{NTT}(\mathbf{u}))$$

Kyber adjusts its security level by changing the parameter k which can take any of $\{2, 3, 4\}$. Key generation operation requires $2k$ NTT operations and k^2 CWM operations. Encryption operation requires k NTT, k^2+k CWM and $k+1$ INTT operations. Decryption operation requires k NTT, k CWM and 1 INTT operations.

III. The Proposed Design

In this section, the proposed hardware architectures and their main arithmetic blocks are explained in a bottom-up fashion, starting from the implementation of modular reduction unit. Then, we present our unified butterfly unit and finally overall design is presented.

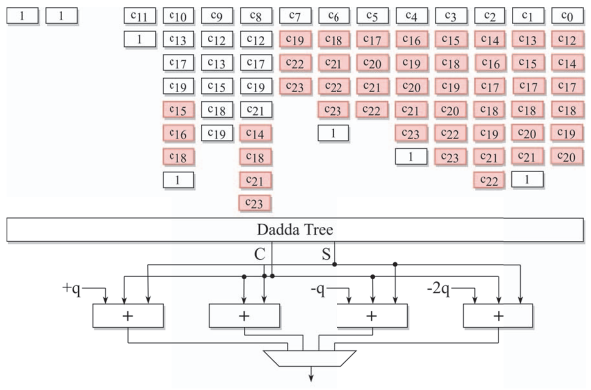


Fig. 1. Modular Reduction Unit

1) Modular Reduction Unit

In this section, we present a constant time modular reduction hardware for $q = 3329 = 2^{12} - 2^9 - 2^8 + 1$. Modular reduction unit takes a 24-bit integer c as input from a DSP block performing 12-bit×12-bit unsigned integer multiplication with 1 clock cycle latency. The proposed modular reduction hardware utilizes the property $2^{12} \equiv 2^9 + 2^8 - 1 \pmod{3329}$ recursively in a similar approach with Zhang *et al.* [19] as shown in Eqn. 3-7 where $c[x:y]$ represents the bits of c from y^{th} bit to x^{th} bit.

$$d = 2^{12}c[23:12] + c[11:0] \tag{3}$$

$$d = 2^9 c[23:12] + 2^8 c[23:12] - c[23:12] + c[11:0] \tag{4}$$

$$d = 2^{12}c[23:15] + 2^{12}c[23:16] + 2^9 c[14:12] + 2^8 c[15:12] - c[23:12] + c[11:0] \tag{5}$$

$$d = (2^9 + 2^8 - 1)(c[23:15] + c[23:16]) + 2^9 c[14:12] + 2^8 c[15:12] - c[23:12] + c[11:0] \tag{6}$$

$$d = 2^9 c[23:15] + 2^8 c[23:15] - c[23:15] + 2^9 c[23:16] + 2^8 c[23:16] - c[23:16] + 2^9 c[14:12] + 2^8 c[15:12] - c[23:12] + c[11:0] \tag{7}$$

We apply this approach recursively until no bits left at position greater than 12. Throughout this process, we eliminate redundant operations by combining identical bits at the same position. For example, there are two $2^8 c[15]$ in Eqn. 7 where they can be combined as single $2^9 c[15]$. We also convert subtraction operation into addition by negating the subtracted integer, adding 1 to the final result and extending the sign bit to 15th bit. As shown in Fig. 1, this recursive process generates a tree of bits where white and red boxes represent bit and negated bit of the input c, respectively.

In order to reduce the tree, we utilize Dadda's method [20] which produces an incomplete result with two integers, C and S, using 50 full adders (FAs) and 13 half adders (HAs). In order to generate the final result, the resulting integers need to be added. We observe that the final result at the end of addition operation is between 9271 and -3264 which is not in the desired range $[0, q)$. Therefore, $(C + S + q)$, $(C + S)$,

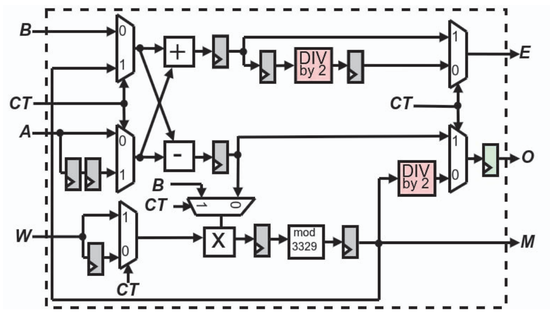
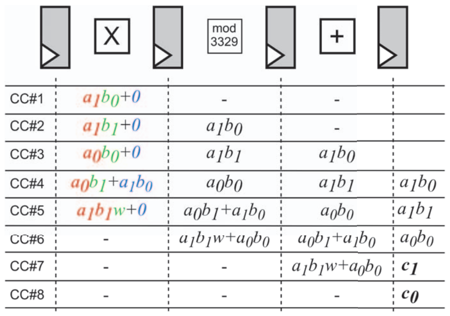


Fig. 2. Unified Butterfly Unit



CC#1	a_1b_0+0	-	-	
CC#2	a_1b_1+0	a_1b_0	-	
CC#3	a_0b_0+0	a_1b_1	a_1b_0	
CC#4	$a_0b_1+a_1b_0$	a_0b_0	a_1b_1	a_1b_0
CC#5	a_1b_1w+0	$a_0b_1+a_1b_0$	a_0b_0	a_1b_1
CC#6	-	$a_1b_1w+a_0b_0$	$a_0b_1+a_1b_0$	a_0b_0
CC#7	-	-	$a_1b_1w+a_0b_0$	c_1
CC#8	-	-	-	c_0

Fig. 3. Scheduling of CWM Operation for Kyber

$(C+S-q)$ and $(C+S-2q)$ are calculated separately after the Dadda tree and the result in the range $[0, q)$ is selected as the final result. The proposed modular reduction unit for $q = 3329$ has 1 clock cycle latency. There are other modular reduction methods such as Barrett and Montgomery [9]. Both Barrett and Montgomery reductions require 2 multiplication operations with additional addition and subtraction operations. Montgomery modular reduction also requires its operands to be converted back and forth in Montgomery space. On the other hand, our proposed implementation utilizes only addition and subtraction operations.

2) Unified Butterfly Unit

NTT operation requires CT butterfly which performs $A + B \cdot \omega \pmod q$ and $A - B \cdot \omega \pmod q$ while INTT operation utilizes GS butterfly structure which performs $A+B \pmod q$ and $(A-B) \cdot \omega \pmod q$. Since the proposed design aims using different butterfly structures for NTT and INTT operations, we propose an unified butterfly unit shown in Fig. 2.

The proposed butterfly unit uses no extra modular multiplier, adder or subtractor than a dedicated CT or GS butterfly unit. The proposed butterfly unit has one modular multiplier, one modular adder and one modular subtractor. It also has pipeline registers for synchronizing output coefficients and multiplexers for reconfigurability. The proposed butterfly unit takes A, B and W as inputs, performs butterfly operation and generates two coefficients E and O as outputs corresponding to the Steps 6–9 of the Algorithm 1 and the Steps 6–8 of the Algorithm 2, respectively. It also takes control signal CT as input which is used as selection signal for multiplexers in the butterfly unit. When CT signal is set as 0 and 1, the butterfly unit is configured to work as GS and CT butterfly, respectively. The butterfly unit has 3 clock cycles latency for both CT and GS configurations.

In [19], Zhang *et al.* proposes a technique to eliminate the multiplication of resulting coefficients with $n^{-1} \pmod q$ after the INTT operation. For an odd prime q, $\frac{x}{2} \pmod q$ can be performed as shown in Eqn. 8 where $\gg$ represents right shift.

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

Utilizing this property, one can divide the output of GS butterfly by two and generate $\frac{A+B}{2} \pmod q$ and $\frac{(A-B)\omega}{2} \pmod q$ coefficients instead of $A+B \pmod q$ and $(A-B)\omega \pmod q$ as shown in the Step 8 of Algorithm 2. In this work, we adopt this technique and insert two `DIVby2` units into our butterfly units (shown as red boxes in Fig. 2) which perform the operation in Eqn. 8. Therefore, we eliminate extra 256 multiplication operations in $\mathbb{Z}_q$ after INTT operation at the expense of extra hardware in the butterfly unit. This technique reduces the number of modular multiplication operation in INTT by 22%.

The proposed butterfly unit can also be configured to perform CWM operations defined in Algorithm 3. A polynomial multiplication in $\mathbb{Z}_q[x]/(x^2 - \omega^i)$ requires 5 multiplication and 2 addition operations as shown in the Steps of 5–6 of Algorithm 3. This operation can be realized using a series of multiply-accumulate which can be performed using CT butterfly configuration and reading E output. Fig. 3 illustrates the multiplication of $(a_0 + a_1x)$ and $(b_0 + b_1x)$ in $\mathbb{Z}_q[x]/(x^2 - \omega)$ where red, green and blue letters represent the input coefficients to B, W and A inputs of butterfly unit, respectively. Multiplication results a_1b_0, a_1b_1 and a_0b_0 are forwarded to the input of butterfly unit before being stored back to the memory. It takes 5 clock cycles for a CWM operation after filling the pipeline.

Unified butterfly is also utilized in previous works [10], [14]. Both of the proposed units use 2 multipliers, whereas our design uses only single modular multiplier unit. Also, our butterfly unit realizes the multiplication with $\frac{1}{2} \pmod q$ operation after GS butterfly using adder and therefore, eliminates the multiplications with n^{-1} at the end of INTT operation.

3) Overall Design

Fig. 4 shows the high-level block diagram of the proposed hardware architecture with one butterfly unit. There are four dual-port BRAMs for each butterfly unit where two BRAMs are used to store the first input polynomial and output polynomial, and two BRAMs are used to store the second input polynomial. There is also one BROM for each butterfly unit to store pre-computed and loaded powers of twiddle factor. Prior to any operation, input polynomials are loaded into the BRAMs with input multiplexers. Then, the proposed hardware starts its operation according to start signals and the resulting polynomial is read using the output multiplexers after the operation.

The proposed architecture implements NTT and INTT schemes shown in Algorithm 1 and 2, respectively. Both operations consist of 7 stages and 128 butterfly operations should be performed in each stage. An n-pt NTT operation can be performed as separate two $(n/2)$-pt NTT operations after

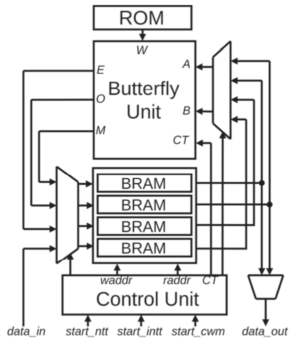


Fig. 4. Overall Design with 1 Butterfly Unit

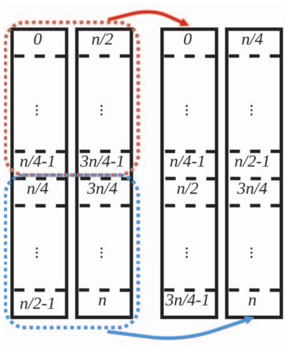


Fig. 5. Memory Access Pattern

the first stage and this approach recursively can be applied to the smaller NTT operations. In this work, we utilize this property and propose an efficient memory access pattern as shown in Fig. 5 where numbers inside the boxes represent indices of stored coefficients. In an NTT stage, coefficients stored in the first half of the memory blocks should be read and written into the first memory block due to changing access pattern of NTT operation in each stage. Similarly, coefficients in the second half of the memory blocks should be read and written into the second memory block as shown in Fig. 5.

We adopt a changing memory read pattern for efficient memory management. Since coefficients in consecutive memory addresses should be written into the same memory for the next stage, they should not be read consecutively. Instead, coefficients from the first and second half of the memory blocks should be read in an alternating way. For example, in the first stage of NTT operation, after reading the coefficients in address 0, coefficients in address $(n/4)$ should be read instead of address 1 for the next butterfly operation. Since the coefficients read in a clock cycle should be written into the same memory block after butterfly operation, we placed an extra register at the output of O output of butterfly unit as shown with green in Fig. 4. Therefore, two coefficients can be written in the same memory block in two consecutive clock cycles. Since the proposed architecture is pipelined, this extra register does not cause any stall. Similarly, INTT operation uses the same memory access pattern.

Our proposed balanced and high-performance designs use 4 and 16 butterfly units respectively. Number of memory blocks increases with number of butterfly units to not stall the pipeline. Coefficients are distributed across memory blocks. While number of memory blocks are increasing, used address space decreases accordingly.

The proposed designs can take two polynomials as inputs and they can perform NTT and INTT operations for both input polynomials. Also, the designs can perform CWM operation between input polynomials. The proposed architectures with 1, 4 and 16 butterfly units finish NTT operation in 904, 232, 69, INTT operation in 904, 233, 71 and CWM operation in 647, 167, 47 clock cycles, respectively, excluding the time for loading input polynomials into BRAMs.

IV. Results

In this section, we present our implementation results and their comparison with the works in the literature.

A. Prior Works

There are full software implementations of Kyber with old parameter sets in the literature. Previously, Seiler *et al.* [6] optimized NTT, INTT and polynomial multiplication operations utilizing AVX2 instructions on Skylake and Haswell architecture processors. Later, they proposed a faster design in [8] using new modular reduction technique. Another implementation of Kyber scheme is published by Botros *et al.* [7]. They proposed a new NTT module for new Kyber parameters with optimizations, such as merging NTT layers and instruction alignment of polynomials on ARM Cortex-M4 processors. There are also hardware accelerators for Kyber proposed in the literature with new Kyber parameters. In [13], the authors proposed a full hardware implementation of Kyber with new parameters utilizing resource reusing. Recent work [14] implements PQC specific schemes on a vector co-processor with RISC-V SCR1 processor targeting ASIC platform. This work supports Kyber with new parameters. The study in [10] implements an NTT accelerator with RISC-V architecture which can be applied to different PQC schemes. There are also hardware accelerators proposed for previous Kyber parameters [9], [11], [12]. Some of these works do not provide results for operations separately, so they are not included or are shown partially on Table I.

B. Implementation Results and Comparison

We developed three hardware architectures with one, four and sixteen butterfly units (lightweight, balanced and high-performance, respectively) proposed in this work into Verilog modules. Then, they are synthesized, placed and routed for different FPGA families. The proposed hardware architectures are first implemented for Spartan-6 FPGA (xc6slx75fgg676-3) using Xilinx ISE 14.7 with default synthesis options. Then, they are implemented for Artix-7 FPGA (xc7a200tffg1156-3) using Xilinx Vivado 2018.1 with default synthesis options. Implementation results of our architectures and the works in the literature are shown in Table I. The proposed modular reduction and butterfly units are also synthesized, placed and routed as separate units for Spartan-6 and Artix-7 FPGAs. Modular reduction unit uses 236 LUTs with 154 MHz and 195 LUTs with 212 MHz for Spartan-6 and Artix-7, respectively. Butterfly unit uses 377 LUTs, 242 DFFs, 1 DSPs with 129 MHz and 312 LUTs, 207 DFFs, 1 DSPs with 192 MHz for Spartan-6 and Artix-7, respectively.

There are not many prior works in the literature using new Kyber parameter set. However, not all works gives performance figures for separate operations. Therefore, it is hard to compare our designs for each operation and parameter set with other works. Our high-performance hardware architecture outperforms prior works for polynomial multiplication operation in terms of latency. The proposed high-performance hardware shows up to 112×, 132× and 109× better performance for

TABLE I
IMPLEMENTATION RESULTS AND THEIR COMPARISON TO PRIOR WORK

Work	Platform	n	q / $\lceil \log_2(q) \rceil$	LUT / REG / DSP / BRAM	Clock (MHz)	Latency (CC)		
						NTT	INTT	PMUL
[6] [a]	Intel Corei7-6600U Skylake	256	7681 / 13	– / – / – / –	–	419	394	1278
	Intel Core i7-4770K Haswell	256	7681 / 13	– / – / – / –	–	460	440	1432
[7] [a,b,c]	ARM Cortex-M4	256	7681 / 13	– / – / – / –	–	9452	10373	32576
		256	3329 / 12	– / – / – / –	–	7725	9347	27873
[10] [a,b]	Zynq-7000	256	– / 16	2908 / 170 / 9 / –	–	1935	1930	–
[11] [a]	Virtex-6	256	7681 / 13	4549 / 3624 / 1 / 12	262	2096	–	–
[12] [a]	Virtex-6	256	7681 / 13	1349 / 860 / 1 / 2	313	1691	–	–
		512	12289 / 14	1536 / 953 / 1 / 3	278	3443		
[13] [b,c]	Artix-7, Virtex-7	256	3329 / 12	– / – / – / –	225	1834	–	–
[14] [a,b]	28nm CMOS	256	3329 / 12	512K / – / – / –	–	41	–	–
		256	7681 / 13			45		
TW-1 BTF [b,c]	Spartan-6	256	3329 / 12	985 / 444 / 1 / 5	138	904	904	3359
	Artix-7			948 / 352 / 1 / 2.5	190			
TW-4 BTFs [b,c]	Spartan-6			2498 / 1046 / 4 / 18	127	232	233	864
	Artix-7			2543 / 792 / 4 / 9	182			
TW-16 BTFs [b,c]	Spartan-6			9898 / 3688 / 16 / 70	115	69	71	256
	Artix-7			9508 / 2684 / 16 / 35	172			

TW:This Work. **PMUL**:INTT(CWM(NTT(A), NTT(B))). [a]:Works with multiple n and q. [b]:Supports new Kyber parameters. [c]:Optimized for constant q.

NTT, INTT and polynomial multiplication, respectively, compared to the high-speed software implementation on Cortex-M4 [7]. Our high-performance design is also 6× faster for NTT operation compared to the software implementation of Kyber PQC scheme with old parameter sets on Intel processors with Skylake and Haswell architectures [6]. For hardware implementations, our design shows better latency performance for NTT operation than the prior works in the literature except for the work in [14] proposing an ASIC design with large 521K gate area. Our high-performance design accelerates NTT operation on FPGA platform up to 28× compared to the work in [10] which uses a 16-bits q. Our balanced design is even faster in terms of latency. However, Fritzmann *et al.* [10] reuses processor resources via RISC-V extended instruction set architecture. We also provide utilization results of our works for low cost Spartan-6 FPGAs. The works [6] and [8] are using Skylake CPUs which are not suitable for embedded systems. That makes our design affordable and practical in use of Kyber PQC scheme in embedded systems.

V. CONCLUSION

In this paper, we present three hardware architectures (lightweight, balanced, high-performance) performing NTT, INTT and polynomial multiplication operations for Kyber scheme. These operations are extensively utilized in key generation, encryption and decryption operations of Kyber and our proposed polynomial multiplier hardware can be utilized as a hardware accelerator for these operations, hence for Kyber. Compared to the high-speed software implementations on Cortex-M4 [7], the proposed high-performance hardware shows up to 112×, 132× and 109× better performance for NTT, INTT and polynomial multiplication, respectively.

REFERENCES

[1] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, “Crystals-dilithium: A lattice-based digital signature scheme,” *IACR Trans. on CHES*, pp. 238–268, 2018.

[2] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, “CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM,” in *IEEE Euro S&P*, 2018, pp. 353–367.

[3] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe, “Post-quantum key exchange—a new hope,” in *25th USENIX*, 2016, pp. 327–343.

[4] E. Alkim, P. S. Barreto, N. Bindel, J. Krämer, P. Longa, and J. E. Ricardini, “The lattice-based digital signature scheme qTESLA,” in *ACNS*. Springer, 2020, pp. 441–460.

[5] P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang, “Falcon: Fast-Fourier lattice-based compact signatures over NTRU,” *Submission to the NIST’s post-quantum cryptography standardization process*, 2018.

[6] G. Seiler, “Faster AVX2 optimized NTT multiplication for Ring-LWE lattice cryptography,” Crypto. ePrint Arch., Report 2018/039, 2018.

[7] L. Botros, M. J. Kannwischer, and P. Schwabe, “Memory-Efficient High-Speed Implementation of Kyber on Cortex-M4,” *Progress in Cryptology – AFRICACRYPT 2019 Lecture Notes in CS*, p. 209–228, 2019.

[8] V. Lyubashevsky and G. Seiler, “NTTRU: Truly Fast NTRU Using NTT,” Cryptology ePrint Archive, Report 2019/040, 2019.

[9] A. C. Mert, E. Karabulut, E. Öztürk, E. Savaş, M. Becchi, and A. Aysu, “A Flexible and Scalable NTT Hardware : Applications from Homomorphically Encrypted Deep Learning to Post-Quantum Cryptography,” in *2020 DATE*, pp. 346–351.

[10] T. Fritzmann, G. Sigl, and J. Sepúlveda, “RISQ-V: Tightly Coupled RISC-V Accelerators for Post-Quantum Cryptography,” *IACR Trans. on CHES*, vol. 2020, no. 4, pp. 239–280, Aug. 2020.

[11] T. Pöppelmann and T. Güneysu, “Towards practical lattice-based public-key encryption on reconfigurable hardware,” in *International Conference on SAC*. Springer, 2013, pp. 68–85.

[12] S. S. Roy, F. Vercauteren, N. Mentens, D. D. Chen, and I. Verbauwhede, “Compact Ring-LWE Cryptoprocessor,” in *CHES*, 2014, pp. 371–391.

[13] Y. Huang, M. Huang, Z. Lei, and J. Wu, “A Pure Hardware Implementation of CRYSTALS-KYBER PQC Algorithm through Resource Reuse,” *IEICE Electronics Express*, vol. advpub, 2020.

[14] G. Xin, J. Han, T. Yin, Y. Zhou, J. Yang, X. Cheng, and X. Zeng, “VPQC: A Domain-Specific Vector Processor for Post-Quantum Cryptography Based on RISC-V Architecture,” *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 67, no. 8, pp. 2672–2684, 2020.

[15] V. Lyubashevsky and G. Seiler, “NTTRU: Truly Fast NTRU Using NTT,” *IACR Trans. on CHES*, vol. 2019, no. 3, pp. 180–201, May 2019.

[16] E. Chu and A. George, *Inside the FFT black box: serial and parallel fast Fourier transform algorithms*. CRC press, 1999.

[17] P. Longa and M. Naehrig, “Speeding up the Number Theoretic Transform for Faster Ideal Lattice-Based Cryptography,” in *Cryptology and Network Security*, Milan, Italy, Nov. 2016, pp. 124–139.

[18] T. Pöppelmann and T. Güneysu, “Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware,” in *Progress in Cryptology – LATINCRYPT 2012*, 2012, pp. 139–158.

[19] N. Zhang, B. Yang, C. Chen, S. Yin, S. Wei, and L. Liu, “Highly Efficient Architecture of NewHope-NIST on FPGA using Low-Complexity NTT/INTT,” *IACR Trans. on CHES*, vol. 2020, no. 2, pp. 49–72, 2020.

[20] L. Dadda, “Some Schemes for Parallel Multipliers,” in *Alta Frequenza*, vol. 34, 1965, pp. 349–356.