High-Speed Polynomials Multiplication HW accelerator for CRYSTALS-Kyber

Abdullah Alhassani, *Student Member, IEEE*, and Mohammed Benaissa, *Senior Member, IEEE*

Abstract— **NIST has selected CRYSTALS-Kyber as the primary Key Encapsulation Mechanism (KEM) algorithm for the standardization process of post-quantum cryptography. This paper proposes a high-speed hardware accelerator targeting the polynomial multiplication of Kyber. The NTT-based algorithm is employed in Kyber to perform polynomial multiplication, where modular multiplication is the most time-consuming operation in the computation of the NTT. This paper proposes a new Residue Number System (RNS) methodology to perform the modular multiplication in Kyber based on fast look-up tables with a novel sub-moduli RNS decomposition of the operation into smaller tables. A high-speed polynomial multiplier FPGA accelerator is developed based on the proposed RNS modular multiplier for both single and double butterfly modes. The resulting designs were implemented on Xilinx Artix-7 FPGA, and post-place and route hardware results obtained confirmed the significant improvements over state-of-art.**

Index Terms— **CRYSTALS-Kyber, Post-Quantum Cryptography (PQC), Number Theoretic Transform (NTT), Lattice-Based Cryptography (LBC), FPGA, Hardware, RNS, High-Speed**

I. INTRODUCTION

THE recent development of quantum computing poses a severe threat to the current public key cryptography primitives such as RSA and ECC. These primitives rely on mathematically hard problems to secure communication between two parties, and with the future availability of a powerful quantum computer running Shor's algorithm, breaking these schemes will be accomplished in polynomial time[1]. For this reason, in 2016[2], the National Institute of Standards and Technology (NIST) started a competition for Post-Quantum Cryptography (PQC) algorithms, which resist quantum and classical attacks, and went on three rounds to select the best candidates. NIST selected four schemes for the standard in July 2022[3], and the competition will continue for a fourth round to select from another four candidates.

CRYSTALS-Kyber[4] was chosen by NIST for standardization as the primary key encapsulation mechanism (KEM), which is a lattice-based cryptography scheme where its security is based on solving the Module Learning With Errors (MLWE) hard problem[5]. The scheme consists of three algorithms: key generation, encryption, and decryption, where polynomial multiplication dominates the computations in these algorithms.

Hardware accelerators play a crucial role in cryptography as they offer high-speed operations compared to software implementation and aim for practical deployment of post-quantum cryptography schemes for real-life applications. Kyber adopted the NTT-based algorithm to perform polynomial multiplication instead of the classical time domain techniques.

Several NTT architectures have been reported in the literature often trading-off hardware complexity and throughput. For example, a highly pipelined hardware architecture based on the systolic array technique is adopted to accelerate the Coefficient Wise Multiplication (CWM) operation at the expense of increased hardware resources [6]. Similarly, memory-based iterative NTT designs based on a small hardware module for performing the NTT operations are deployed to reduce hardware usage [7, 8].

The intrinsic operation in the hardware structure of NTT is the butterfly unit, which involves modular multiplication, modular addition, and modular subtraction. While modular addition and subtraction can be easily implemented, modular multiplication is the most complex operation in the butterfly and, therefore, received attention in most of the previous works.

The modular multiplication is computed in two steps: multiplication of two numbers followed by a reduction to keep the result in the same ring. The state-of-the-art works associated with the hardware implementation of Kyber polynomial multiplication involve both Montgomery reduction[9] and Barret reduction[10]. The authors in [11] used Montgomery to implement the modular reduction. A variant of Montgomery reduction that includes KRED and KRED-2X functions was presented in [12] and further improved in [13] to optimize the reduction further for Kyber. However, most other works focussed on Barrett reduction and its variants. In [14, 15], Barret reduction is utilized directly for the modular reduction, whereas [16-18] derived improved versions from the original Barrett.

Most reported hardware implementations for Kyber have been targeted at FPGA technology, where DSP slices were favoured for the implementation of the modular multiplication during the NTT/INTT computations and the CWM operation. FPGA is a platform that allows hardware acceleration as well as flexibility. It also allows hardware performance evaluation.

In this work, we first consider modular multiplication and propose a new Residue Number System (RNS) methodology

Abdullah Alhassani and Mohammed Benaissa are with the Department of Electronic and Electrical Engineering, University of Sheffield, Sheffield S1 3JD, UK (e-mail: ahmalhassani1@sheffield.ac.uk; m.benaissa@sheffield.ac.uk).

based on look-up tables to compute the $a.b \bmod q$ operation that is optimized for Kyber. A ROM array architecture is proposed based on judicious isomorphic mapping, optimal sub-moduli decomposition, and efficient memory addressing. The architecture exploits the use of BRAMs on FPGAs to accelerate computations. The use of BRAMs to support computations can free up the FPGA logic blocks and DSP resources for further computational tasks thus enabling trade-offs to be made between computation logic and memory resources.

A single butterfly unit (SBU) and a dual butterfly unit (DBU) are proposed using the proposed modular multiplier in single and dual mode, respectively, to build an NTT-based polynomial multiplier for Kyber that is capable of computing the whole operation without using DSP slices.

The resulting hardware accelerator performs the complete polynomial multiplication, including computing NTT, INTT and CWM. The implementation results on the NIST-recommended FPGA Xilinx Artix-7 show that the proposed design can compute the entire polynomial multiplication in 9.6 μs with one SBU, 5 μs with one DBU, and in 2.6 μs with two DBUs hence outperforming similar previous works by 46%, 17%, and 36% respectively.

Research in Side Channel and Fault injection Analysis countermeasures in NTT designs has received increasing interest in recent years due to their importance in securing lattice-based PQC implementations such as the error detection schemes introduced in [19, 20]. The use of look-up tables in this work can offer the advantage of constant time implementation and, hence, better resistance to timing attacks [21].

The rest of the paper is organized as follows: Section II provides background on the Kyber protocol and the NTT-based polynomial multiplication algorithm. Section III details our contributions in this paper, including our design methodology and the proposed designs. The implementation results and comparisons with the existing work are presented in section IV. Finally, the conclusion is given in section V.

II. Background

A. Notation

Let $\mathbb{Z}_q$ denotes the ring of integers $\{0, 1, \dots, q-1\}$ with the modulus q, and $\mathbb{R}_q = \mathbb{Z}_q[x]/(x^n+1)$ represents the quotient ring of polynomials where the polynomials are reduced by cyclotomic polynomial x^n+1, and the coefficients are integers in $\mathbb{Z}_q$. The degree of a polynomial $f \in \mathbb{R}_q$ is at most $n-1$.

A polynomial in the normal domain is written with regular lower-case letters as a, and its transformation to the NTT domain is defined as $\hat{a} \leftarrow NTT(a)$ where $\hat{a}$ represents the polynomial in the NTT domain. The inverse of the NTT transform is defined as $a \leftarrow INTT(\hat{a})$.

A k-dimensional polynomial vector is written with bold lower-case letters, e.g., $\boldsymbol{s}$ and a $k \times k$ dimensional polynomial matrix is written with bold upper-case letters, e.g., $\boldsymbol{A}$. The transpose of a vector $\mathbf{a}$ (or matrix $\mathbf{A}$) is represented as $\boldsymbol{a}^T$ (or $\boldsymbol{A}^T$). Multiplication in the NTT domain, i.e., CWM denoted as * where $\cdot$ indicates integer multiplication and matrix multiplication is represented as $\circ$.

B. CRYSTALS-Kyber (Kyber)

Kyber[4] is the primary KEM selected by NIST for the PQC standard. The scheme first encrypts a 32-bytes message using the conventional method to build an indistinguishability under the Chosen-Plaintext Attack (IND-CPA) secure public-key encryption scheme. Next, a modified Fujisaki-Okamoto (FO) transform[22] is used to construct an indistinguishability under adaptive Chosen Ciphertext Attack (IND CCA2) secure KEM.

Elements in Kyber are represented as polynomials in $\mathbb{R}_q = \mathbb{Z}_q[x]/(x^n+1)$. The degree of the polynomials $n = 256$ and $q = 3329$. Kyber provides three NIST security levels: 1, 3, and 5 as Kyber-512, Kyber-768, and Kyber-1024, respectively. This can be adjusted according to the module lattice dimension k, which can take the values of 2, 3, and 4 for the three security levels in order. The Kyber IND-CPA consists of three algorithms: key generation, encryption, and decryption. The simplified version of these algorithms is presented as follows, and for further details, please refer to the protocol specification in[4]:

- Keygen(): at Allice's side, firstly, a public matrix $\widehat{\boldsymbol{A}} \in \mathbb{R}_q^{k\times k}$ is sampled from a uniform distribution in the NTT domain using a random seed ρ. Then, a secret $\boldsymbol{s}$ and an error $\boldsymbol{e} \in \mathbb{R}_q^k$ are sampled from a cantered binomial distribution (CBD) and computing $\hat{\boldsymbol{s}} = NTT(\boldsymbol{s})$ and $\hat{\boldsymbol{e}} = NTT(\boldsymbol{e})$. Finally, the public key is generated as $pk = (\rho, \hat{\boldsymbol{t}})$ where $\hat{\boldsymbol{t}} = \widehat{\boldsymbol{A}} \circ \hat{\boldsymbol{s}} + \hat{\boldsymbol{e}}$, and the secret key as $sk = \hat{\boldsymbol{s}}$.
- Encryption(pk, m): at Bob's side again, the public matrix $\widehat{\boldsymbol{A}}$ is sampled from uniform distribution using the same seed ρ. Then, $\boldsymbol{r}, \boldsymbol{e_1} \in \mathbb{R}_q^k$ and $e_2 \in \mathbb{R}_q$ are sampled from CBD and computing $\hat{\boldsymbol{r}} = NTT(\boldsymbol{r})$. The message m is encoded, and then the ciphertext is constructed as $ct = (\boldsymbol{u}, v)$ where $\boldsymbol{u} = INTT(\widehat{\boldsymbol{A}}^T \circ \hat{\boldsymbol{r}}) + \boldsymbol{e_1}$ and $v = INTT(\hat{\boldsymbol{t}}^T \circ \hat{\boldsymbol{r}}) + e_2 + m$.
- Decryption(ct, sk): Allice recovers the message from the received ciphertext as $m = v - INTT(\hat{\boldsymbol{s}}^T \circ NTT(\boldsymbol{u}))$.

C. Polynomials multiplication based on NTT

The bottleneck operation in Kyber is polynomial multiplication, where the NTT-based algorithm is used in the scheme instead of the traditional time domain approach. Multiplication using the NTT can perform the operation in $O(n(\log n))$ time complexity. The NTT transform is a generalization of the Discrete Fourier Transform and is defined over the ring $\mathbb{Z}_q$. Let a polynomial $a(x) = a_0 + a_1x + a_2x^2 \dots.. a_{n-1}x^{n-1}$, its forward NTT transformation into $\hat{a}(x)$ can be written as [23]:

$$\hat{a}(x) = \sum_{j=0}^{n-1} a_j\, \omega_n^{ij} \bmod q, i \in [0, n-1] \tag{1}$$

Where ω, i.e., the twiddle factor constant represents the n-th root of unity satisfying the two conditions: $\omega^n \equiv 1(mod\ q)$ and $\omega^n \neq 1(mod\ q)\ \forall i < n$. The modulus q also must be a prime number that satisfies the condition $q \equiv 1\ (mod\ n)$. Converting a polynomial back to the normal domain is accomplished using INTT transform, where $\omega^{-1}\ (mod\ q)$ utilized as twiddle factors instead of ω, and scaling the result with n^{-1}.

Algorithm 1: Forward NTT based on CT butterfly

```
Input   $a(x) \in \mathbb{R}_q$ in normal order, $\omega \in \mathbb{Z}_q$ and $n = 2^l$
Output  $\hat{a}(x) \in \mathbb{R}_q$ in bit-reversed order
 1:  $k = 1$
 2:  for $(i = 1;\ i < l - 1;\ i = i + 1)$ do
 3:      $m = 2^{l-i}$
 4:      for $(s = 0;\ s = n;\ s = s + m)$ do
 5:          for $(j = s;\ s = s + m;\ j = j + 1)$ do
 6:              $W = \omega^{br_7(k)}\ mod\ q$
 7:              $X = a[j], Y = a[j + m]$
 8:              $T = (W \cdot Y) mod\ q$
 9:              $a[j] = (X + T)\ mod\ q$
10:              $a[j + m] = (X - T)\ mod\ q$
11:          end for
12:          $k = k + 1$
13:      end for
14:  end for
```

Since polynomials multiplication in Kyber is performed over the ring $\mathbb{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$, the negative wrapped convolution can be utilized to eliminate the need for doubling the inputs and the extra polynomial reduction by $x^n + 1$. If this case is considered, inputs need pre-processing by scaling the coefficients with powers of the $2n$-th root of unity. Similarly, the outputs require post-processing operation by scaling the coefficients with powers of the inverse $2n$-th root of unity. However, in [24], a new method is presented to perform polynomial multiplication using the NTT without the pre-processing and post-processing operations. The new method only restricts the modulus to have the n-th root of unity $q \equiv 1\ (mod\ n)$ without requiring the $2n$-th root of unity to exist. This variant is called incomplete-NTT and produces $^{n}/_{2}$ polynomials of degree 1 and the definitions of NTT and INTT have been modified accordingly as presented in algorithms 1 and 2, respectively, where $br_7(k)$ is the bit reversal function of an unsigned 7-bit integer k. Furthermore, the Cooley-Tukey (CT) [25] butterfly structure for NTT and Gentleman-Sande (GS) [26] butterfly for the INTT are proposed to avoid bit-reversal operations [18].

In the incomplete-NTT, Coefficients Wise Multiplication (CWM) is performed differently. The operation is performed by multiplying the two 128 degree-1 polynomials in the ring $\mathbb{Z}_q[x]/(x^2 - \omega^{2br_7(i)+1})$ for $i = 0, \dots, 127$. For example, for two polynomials $\hat{a}$ and $\hat{b}$, their CWM is computed as:

$$\begin{aligned}\hat{c} &= CWM(\hat{a}, \hat{b}) \\ &= \hat{a} * \hat{b} \\ &= (\hat{a}_{2i+1}X + \hat{a}_{2i}) \cdot (\hat{b}_{2i+1}X + \hat{b}_{2i})\ Mod\ (x^2 - \omega^{2br_7(i)+1}) \\ &= (\hat{a}_{2i+1} \cdot \hat{b}_{2i} + \hat{a}_{2i} \cdot \hat{b}_{2i+1})X + \\ &\quad \hat{a}_{2i} \cdot b_{2i} + \hat{a}_{2i+1} \cdot \hat{b}_{2i+1} \cdot \omega^{2br_7(i)+1} \end{aligned} \tag{2}$$

This requires five modular multiplications and two modular additions for each pair of coefficients, and for the 128 pairs, it costs 640 modular multiplications and 256 modular additions.

Algorithm 2: Inverse NTT based on GS butterfly

```
Input   $\hat{a}(x) \in \mathbb{R}_q$ in bit-reversed order, $\omega^{-1} \in \mathbb{Z}_q$ and $n = 2^l$
Output  $a(x) \in \mathbb{R}_q$ normal in order
 1:  $k = 0$
 2:  for $(i = l - 1;\ i < 1;\ i = i - 1)$ do
 3:      $m = 2^{l-i}$
 4:      for $(s = 0;\ s = n;\ s = s + 2m)$ do
 5:          for $(j = s;\ s = s + m;\ j = j + 1)$ do
 6:              $W = \omega^{br_7(k)+1}\ mod\ q$
 7:              $X = \hat{a}[j], Y = \hat{a}[j + m]$
 8:              $\hat{a}[j] = (X + Y) \div 2\ mod\ q$
 9:              $\hat{a}[j + m] = (X - Y) \cdot W \div 2\ mod\ q$
10:          end for
11:          $k = k + 1$
12:      end for
13:  end for
```

III. THE PROPOSED DESIGN

A. Modular multiplier

The proposed multiplier adopts an RNS methodology based on a look-up tables approach. This requires the design of an efficient ROM array architecture based on judicious isomorphic mapping, sub-moduli decomposition, and memory addressing, as explained below.

The operation of multiplication modulo q can be computed using look-up tables, and for Kyber, the modulus is 3329, which can be represented as a 12-bit unsigned number. However, direct implementation is not efficient as it requires storing 2^{24} of 12-bit records in the look-up tables, which consumes large memory. The isomorphism between a multiplicative group g having elements $g_n = \{1, 2, \dots, q - 1\}$with the multiplication modulo q, and the additive group k having elements $k_n = \{0, 1, \dots, q - 2\}$ with the addition, modulo $q - 1$ can be used to improve modular multiplication using the mapping $g_n = \propto^{kn}$

where $\propto$ is a primitive root of q. In this case, the multiplication is replaced by addition and can be computed as follows[27]:

$$\left|g_n \cdot g_j\right|_q = \alpha^{|k_n+k_j|_{q-1}} \tag{3}$$

Where $|a|_q$ represents the least positive residue of a modulo q.

The modular multiplication can be performed by finding the index k_i for the two multiplied numbers. Then, performing the addition of the two indexes modulo $q-1$ and, finally, inversing the index operation to find the correct result. The merit of replacing multiplication with addition is that addition can be computed in a modulus other than the prime modulo of Kyber, i.e., 3329, with only restricting the new modulus to be at least twice as the original one.

Since the only restriction on the new modulus is its minimum size, a composite modulus with an adder tree can be used to improve the implementation efficiency. In this work, an RNS submodular ROM array adder is proposed where the modulus is decomposed into three relatively prime moduli and the addition is performed in these three submodules. Finally, the result is reconstructed using a look-up table, and it involves submodular reconstruction, modulus overflow correction and inverse index look-up.

Now we explain how to generate look-up tables ROM entries.

1- Submodular index tables:

The primitive root $\propto = 3$ is selected considering the modulus of Kyber $q = 3329$, and $\{7, 31\ and\ 32\}$ as our three submodular system satisfying the following condition: $2q < 7 \cdot 31 \cdot 32$. Firstly, the following table is generated from the mapping: $g = |3^k|_{3329}$ for $k = 0, 1, \dots, q-2$

k	0	1	2	3	…	200	201	..	3326	3327
g	1	3	9	27	…	1242	397	..	370	1110

Then, the contents and the addresses of the table are interchanged and then taking the modulo of the three sub-moduli to generate the index table for each sub-moduli as in the following table:

g	1	2	3	…	370	371	…	3327	3328
k	0	1134	1	…	3326	2306	…	2798	1664
$\|k\|_7$	0	0	0	…	1	3	…	5	5
$\|k\|_{31}$	0	18	1	..	9	12	…	8	21
$\|k\|_{32}$	0	14	1	…	30	2	…	14	0

2- Submodular addition tables:

For each of the sub-moduli, a table is generated that contains the sub-modulo residue of the addition of the two input residues. The addresses of these tables are determined by concatenating the two to be added sub-modulo residues.

3- Reconstruction table

This table is represented in a three-dimensional way to recover the correct result of the modular multiplication given the use of three sub-moduli. The addresses of this table are formed by concatenating the output from the previous three submodular addition tables; q_1 determines the page address and q_2 and q_3 both give the column and row addresses, respectively. The contents of this table are calculated in the following three steps:

i- Submodular reconstruction:

For our sub-moduli system q_1, q_2 and q_3, the corresponding residues are r_1, r_2 and r_3, and using the Chinese Remainder Theorem (CRT), a number can be constructed back as follows:

$$r = \left|\sum_{j=1}^{3} \hat{q}_j \cdot \left|\frac{r_j}{\hat{q}_j}\right|_{q_j}\right|_{q_1 \cdot q_2 \cdot q_3} \tag{4}$$

where $\hat{q}_j = \frac{q_1 \cdot q_2 \cdot q_3}{q_j}$

This can be written as:

$$r = \left| r_1 \cdot \hat{q}_1 \cdot \left|\frac{1}{\hat{q}_1}\right|_{q_1} + r_2 \cdot \hat{q}_2 \cdot \left|\frac{1}{\hat{q}_2}\right|_{q_2} + r_3 \cdot \hat{q}_3 \cdot \left|\frac{1}{\hat{q}_3}\right|_{q_3} \right|_{q_1 \cdot q_2 \cdot q_3} \tag{5}$$

This can be simplified further as follows:

$$r = \left| \begin{array}{c} r_1 \cdot q_2 \cdot q_3 \cdot \left|\frac{1}{q_2 \cdot q_3}\right|_{q_1} + \\ r_2 \cdot q_1 \cdot q_3 \cdot \left|\frac{1}{q_1 \cdot q_3}\right|_{q_2} + \\ r_3 \cdot q_1 \cdot q_2 \cdot \left|\frac{1}{q_1 \cdot q_2}\right|_{q_3} \end{array} \right|_{q_1 \cdot q_2 \cdot q_3} \tag{6}$$

Given our chosen sub-moduli system and computing the following modular inverse:

$$\begin{aligned} \left|\frac{1}{q_2 \cdot q_3}\right|_{q_1} &= \left|\frac{1}{992}\right|_7 = 3 \\ \left|\frac{1}{q_1 \cdot q_3}\right|_{q_2} &= \left|\frac{1}{224}\right|_{31} = 9 \\ \left|\frac{1}{q_1 \cdot q_2}\right|_{q_3} &= \left|\frac{1}{217}\right|_{32} = 9 \end{aligned} \tag{7}$$

The equation now is written as:

$$r = |r_1 \cdot 2976 + r_2 \cdot 2016 + r_3 \cdot 1953|_{6944} \tag{8}$$

ii- Modulus overflow correction:

The sub-moduli q_1, q_2 and q_3 have no overflow, and only the modulus $q-1$ needs overflow correction as follows:

$$r_i = |r|_{q-1} \tag{9}$$

iii- Inverse index look-up:
The following mapping is used to inverse the operation and recover the modular multiplication result:

$$x_i = |\propto^{r_i}|_q \tag{10}$$

Following these three steps, the whole reconstruction table contents are generated.

Algorithm 3 summarises the proposed modular multiplication procedure.

B. Modular multiplier interconnections and architecture

The proposed architecture of the modular multiplier is shown in Fig. 1. It shows the interconnections of the ROMs with some of the table's content. In theory, multiplication by zero is not allowed; therefore, this particular case is coded manually. The sub-moduli $q_2 = 31$ route is chosen to detect the multiplication by zero by storing 31 in index zero as this number is not a valid result of sub-modulo 31. Then, the addition table corresponding to the row and column of 31 will also contain 31, and this will lead to the result of zero output from the reconstruction table as column address 31 contains all zeros.

Algorithm 3: The proposed Modular multiplication

```
Input    Two integers $x$ and $y$
         Modulus $q$ , Primitive root of the modulus $\propto$
         Sub-moduli set $q_1, q_2, q_3$
Output   $z = |x \cdot y|q$
   1:    Find index $k$ for $x$ and $y$ using $g = |\propto^k|_q$
   2:    Compute submodular residues for $k(x), k(y)$:
   3:        $x_1 = |k(x)|_{q_1}, x_2 = |k(x)|_{q_2}, x_3 = |k(x)|_{q_3}$
   4:        $y_1 = |k(y)|_{q_1}, y_2 = |k(y)|_{q_2}, y_3 = |k(y)|_{q_3}$
   5:    Compute residues additions:
   6:        $r_1 = |x_1 + y_1|q_1$
   7:        $r_2 = |x_2 + y_2|q_2$
   8:        $r_3 = |x_3 + y_3|q_3$
   9:    Compute CRT: $r = CRT(r_1, r_2, r_3)$
  10:    Correct modulus overflow: $r_i = |r|_{q-1}$
  11:    Inverse index look-up: $z = |\propto^{r_i}|_q$
```

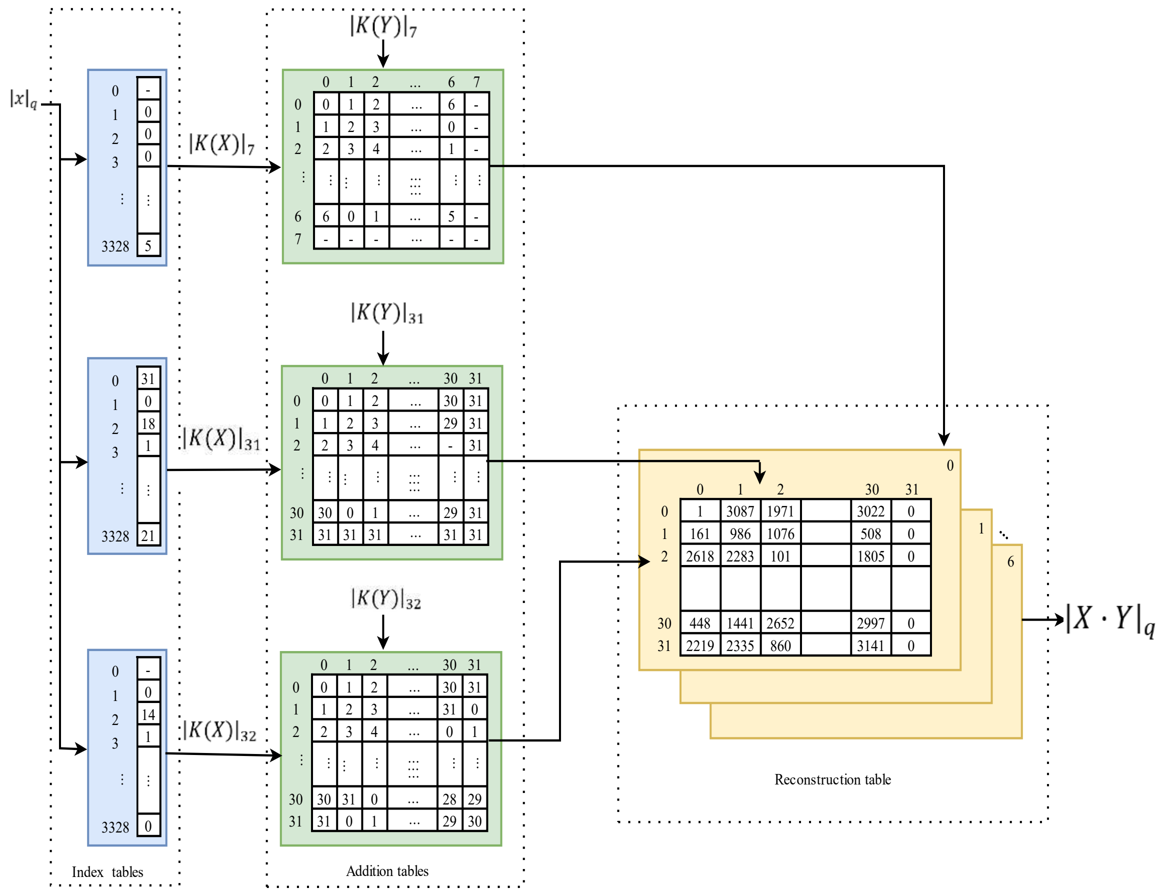

Fig. 1: Multiplier tables interconnections

The sub-moduli systems have been carefully selected as this will lead to an efficient implementation. Sub-moduli 7 is represented with 3-bit, and both sub-moduli 31 and 32 are represented with 5-bit. There is an unused address pattern in the addition tables of these sub-moduli and the reconstruction table; however, we believe this is the minimal for Kyber modulus.

Implementing the multiplier on FPGA requires only storing the tables in Read Only Memory (ROM). In this work, we chose to implement the tables in the FPGA BRAMs configured as ROM. Multiplying two numbers requires 3 ROMs for each input to store the index tables. The size of the first ROM is 3-bit × 3329, while the other two are 5-bit × 3329. These three ROMs are concatenated together to form a 13-bit × 3329 ROM for an efficient FPGA BRAM utilization, which is equivalent to 1.5 36K BRAM. Then, addition tables are stored in 3 ROMs where the first one is 3-bit × 64, and the other two are 5-bit × 1024. These small tables are equivalent to half of a 36k BRAM. Finally, a 12-bit × 7168 ROM is required to store the reconstruction table and is taking 2.5 36k BRAM. In total, a 6 36k BRAMs are needed to perform the modular multiplication of two numbers. A dual-port ROM is available in most FPGAs, which can be utilized to implement two modular multiplications with the same resources, and in this work, we call it a dual modular multiplier (DMM). The operation is completed in three clock cycles for both single and dual modes.

C. Dual butterfly

The proposed dual butterfly unit is shown in Fig. 2. It employs our dual modular multiplier to perform two butterfly operations simultaneously with the same modular multiplier resources. The butterfly is designed to be configurable to perform NTT, INTT or CWM butterfly operations. As presented in algorithm 1, the NTT utilizes CT butterfly, while GS butterfly architecture is used for the INTT operations, as shown in algorithm 2. INTT requires multiplying the coefficients by n^{-1}, which results in an extra 256 modular multiplication for each polynomial. However, a method presented in[28] eliminates this extra multiplication by dividing by two the outputs of the GS butterfly operation at each INTT stage. This work follows this approach and creates a special hardware unit for modular division by 2 for each output. The butterfly also performs the CWM as in (2), which needs five modular multiplications and two modular additions. The butterfly comprises of one dual modular multiplier, two modular additions, two modular subtractions, and four modular divisions by two.

The single butterfly unit is similar to the dual one except that it processes one pair of the input coefficients at a time instead of two pairs and uses the proposed multiplier in single mode. Both butterfly structures require 6 clock cycles to execute the butterfly operations. The delay registers are placed to synchronize the operations and allow pipeline execution. Modular addition is implemented by addition followed by reduction and is implemented in two clock cycles to minimize the critical delay path—similarly, modular subtraction is implemented. Modular division by 2, i.e., $(a/2 \ mod \ q)$ is implemented by only shifting the input right by one if the input is even. In case the input is odd, then the shifting right of the input by one followed by addition with the precomputed constant $q + 1/2$.

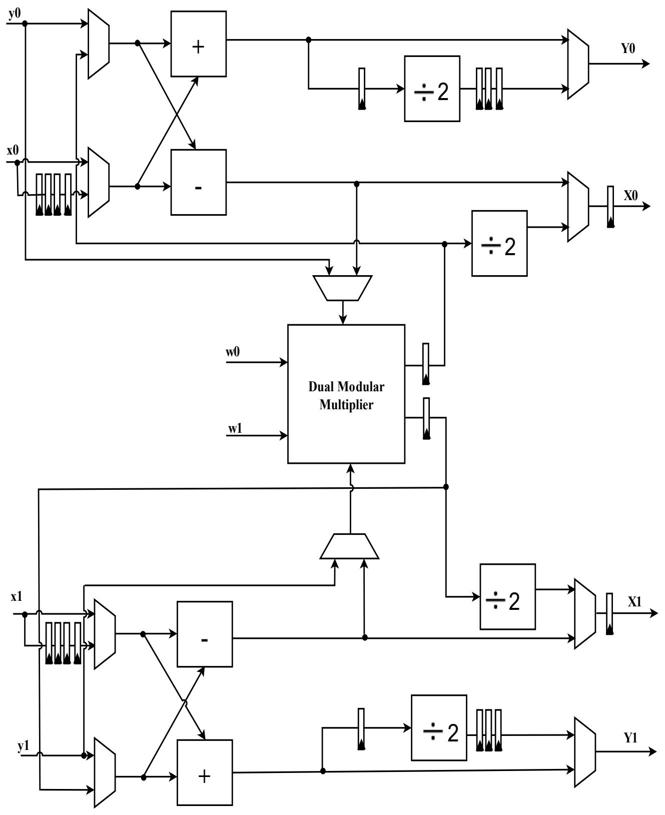

Fig. 2: Dual Butterfly Unit (DBU)

D. Top-level design

The top-level design of the proposed accelerator with one dual butterfly unit is shown in Fig. 3. The engine consists of 4 main components: BRAMs, ROM, DBU, and control unit. The twiddle factors are precomputed and stored in the ROM. The control logic generates the appropriate read-and-write addresses and initiates the desired state machine for NTT, INTT or CWM operations.

Memory access is a challenging aspect of designing NTT accelerators. A flexible design presented in[16] which requires 4 BRAMs for each butterfly, resulting in a total of 8 BRAMs used to store the two multiplied polynomials. In this work an efficient memory organization is proposed as shown in Fig. 4 where only a single memory bank is used for both operands but with opposite orders to allow efficient access during the CWM process. This lowers the BRAM usage to by half in comparison with the work in [16]. Second, for multiple PE configurations as in the one DBU and two DBU cases, the coefficients are packed together to form a 24-bit and 48-bit data paths, respectively, and stored in one memory bank. As a result, 2 BRAMs are enough to store the two polynomials for the three versions of the accelerator which is equivalent to one 36k BRAM.

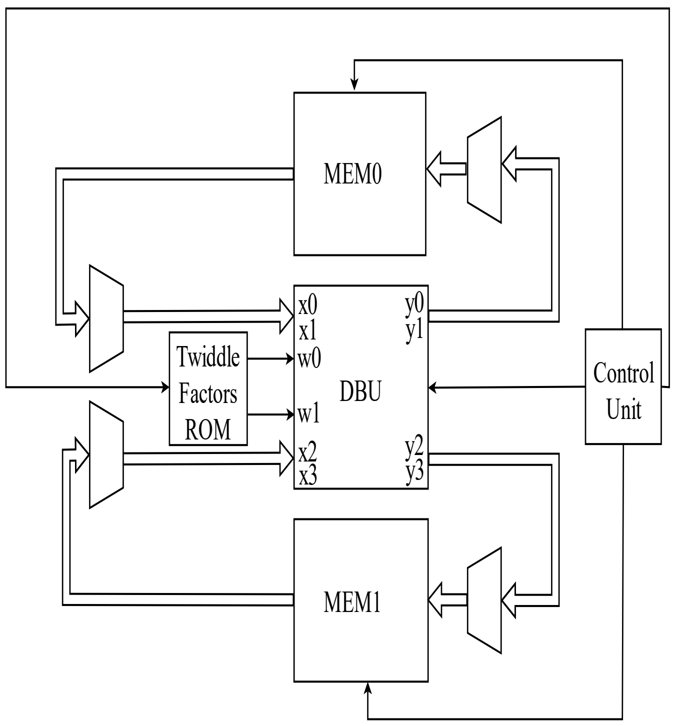

Fig. 3: Top-level design with one Dual Butterfly Unit (DBU)

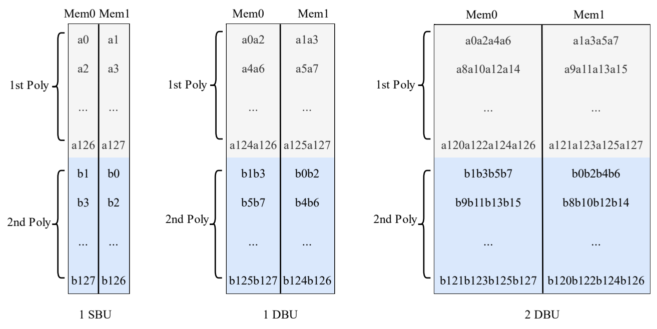

Fig. 4: Memory Organization

The CWM operations schedule is shown in Fig. 5. The schedule starts with a_1b_1 multiplication since it is the longest path to calculate the term $a_1b_1\omega$ and allows an efficient use of the modular multiplier. The first result of the CWM is available after 9 clock cycles, and then it takes 5 clock cycles after filling the pipeline. The proposed accelerator first computes the NTT for both input polynomials, then executes coefficient-wise multiplication and finally, inverses the NTT to obtain the multiplication result. Since the NTT in Kyber is incomplete, it requires 7 stages to perform both the NTT and INTT. This requires 128 operations in each stage, and with one DBU design, the NTT and INTT can be computed in 458 clock cycles. The CWM is performed in 328 clock cycles with the same butterfly configuration.

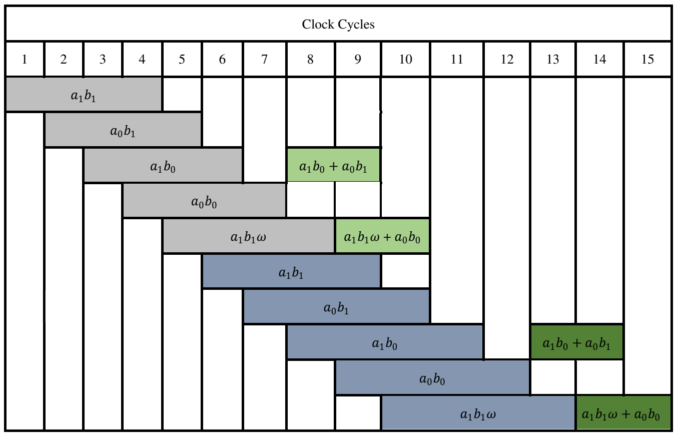

Fig. 5: The CWM operations schedule

IV. Results and comparison

In this section, we present the results of the FPGA implementation of the proposed design. The design is written in VHDL language, and we used Xilinx Artix-7 XC7A200 FPGA to allow a fair comparison with the existing works. We used the Xilinx Vivado 2022 tool to synthesize and implement the design, and we recorded the result after place and route for a more accurate result. To explore the performance of our design, we implemented three versions of our NTT-based polynomial multiplier: SBU design, one DBU-based design, and two DBU-based. For a fair comparison, we compare each version with its counterpart in the literature, as shown in Table I.

The proposed modular multiplier shortens the critical data path to compute the modular multiplication into FPGA routing between the device BRAMs, as the entire modular multiplication is now performed in look-up tables, reducing the time complexity significantly. Therefore, the accelerator performance of our three versions outperforms the related existing works. The proposed multiplier is evaluated with complete polynomial multiplication, which comprises two NTT operations, one CWM and one INTT. The SBU design computes the polynomial multiplication in 9.6 μs, outperforming the work in[16] by 46%, while our design with one DBU achieves 17% improvement compared to the work in[29] to compute the multiplication in 5 μs. Finally, with a 36% improvement compared to the design in[30], the 2 DBU design computes the NTT in 2.6 μs.

To investigate the design efficiency, the Area-Time Product with relation to LUT (ATP-LUT) and with relation to the Equivalent Number of Slices (ATP-ENS) are compared against previous works. The ATP-LUT shows a significant improvement compared to previous works with the SBU and two DBU cases. Our design with SBU is 71 % better than the work in[31], while the two DBU gives a 61% improvement in comparison with the work in[30]. With one DBU design, our proposed design contributes to nearly 37% improvement in comparison with the work in[29]. Although the proposed methodology consumes higher BRAM usage, the ATP-ENS points to the efficiency of the proposed methodology with SBU and two DBU designs showing a slight improvement against

TABLE I

FPGA IMPLEMENTATION RESULTS WITH COMPARISON TO RELATED WORK

Work	BU[†]	AREA LUT	FF	Slices	DSP	BRAM	Freq. (MHz)	Latency (CC) NTT/INTT/CWM	PMUL Time[‡]	ATP-LUT[§]	ATP-ENS[¶]
[16]	1	948	352	281	1	2.5	190	904/904/647	**17.7**	16.8	**15.8**
[31]		360	145	187	3	2	115	940/1203/1289	38	**13.7**	35.7
This Work - 1 SBU		**407**	**411**	**153**	**0**	**7.5**	**350**	**906/906/649**	**9.6 (46%)**	**3.9 (71%)**	**15.6 (1%)**
[29]	2	784	441	259	2	3	200	313/313/256	**6.0**	**4.7**	6.5
[31]		737	290	371	6	4	115	474/602/1289	24.7	18.2	46.3
[17]		1579	1058	527	2	3	161	448/448/256	9.9	15.7	13.5
[18]		880	999	345	2	1.5	229	448/448/256	7.0	6.1	**6.1**
This Work - 1 DBU		**590**	**671**	**210**	**0**	**7.5**	**342**	**458/458/328**	**5.0 (17%)**	**2.9 (37%)**	**8.4 (-28%)**
[16]	4	2543	792	735	4	9	182	232/233/167	4.7	12.1	14.1
[32]		1549	788	635	4	16	159	457/457/140	6.6[1]	10.3	28.2
[30]		1740	643	575	4	5	200	225/225/140	**4.1**	**7.1**	**8.3**
This Work - 2 DBU		**1052**	**1058**	**395**	**0**	**13.5**	**334**	**234/235/169**	**2.6 (36%)**	**2.7 (61%)**	**7.9 (4%)**

[†] The number of butterflies in the design
[‡] The time in **μs** of complete polynomial multiplication i.e., INTT(NTT(X) * NTT(Y)).
[§] Area-Time Product (ATP-LUT) = Total Time of computing polynomial multiplication in ms × number of LUT.
[¶] Area-Time Product (ATP-ENS) = Total Time of computing polynomial multiplication in ms × Equivalent Number of Slices (ENS). ENS = Slices + DSP × 120 + BRAM × 196
[1] 1 NTT operation is considered in total time instead of 2 because 2 polynomials are computed in parallel.

the previous work in[16] and[30] by 1% and 4%, respectively and the one DBU design has ATP-ENS lower than the work in[18] by 28%.

Fig. 6 compares the three proposed accelerators in terms of time required to compute the polynomial multiplication (delay), ATP-LUT and ATP-ENS. As expected, more resources (from one SBU towards 2 DBUs) result in a shorter delay, but interestingly, the ATP-LUT and ATP-ENS improve gradually and then remain almost steady from one DBU to 2 DBUs designs. This is because, in the SBU design, the modular multiplier is used with single mode configuration, whereas in the other two designs, the multiplier is used efficiently in dual mode.

Our design does not consume any DSP slices as we implemented the entire modular multiplication in ROM look-up tables using BRAMs instead of the device-distributed memory. This enables reduced LUT resource usage on the FPGA while still achieving high clock frequencies of 350 MHz, 342 MHz, and 330 MHz for the three design versions, respectively, the highest in the literature to date to our best knowledge.

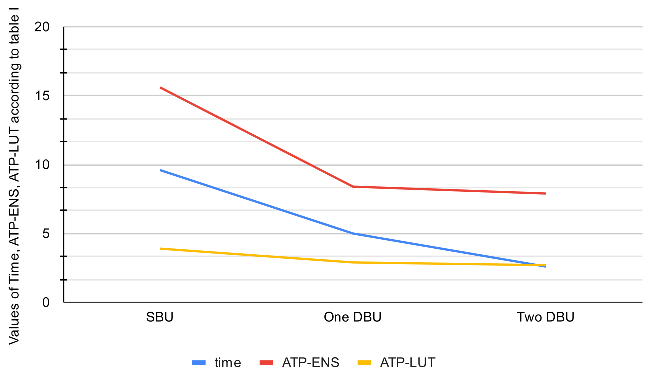

Fig. 6: Performance comparison of the three proposed accelerator

V. Conclusion

This paper presented a high-speed hardware accelerator for the polynomial multiplication of CRYSTALS-Kyber based on NTT. A new look-up tables approach is proposed with corresponding hardware architecture to compute the modular multiplication efficiently. A new sub-moduli system is developed to decompose the operation in the Kyber modulus into smaller sub-moduli for efficient implementation. On top of this modular multiplier, single and dual butterfly NTT units are built to evaluate the performance of the proposed design. The designs were implemented, placed, and routed on Xilinx Artix-7 FPGA. The accelerator with SBU computes the entire polynomial multiplication in 9.6 μs faster than the previous works by 46%.

The methodology presented enables trade-off to be made between computing logic and memory resources. Although the presented designs were targeted to FPGA fabrics, it is expected that similar conclusions to be drawn in terms of results for ASIC implementation as demonstrated with the presented Area-Time Product (ATP) metric. Furthermore, the ATP results could be used as indicative of energy performance, however, further investigation would be required for power/energy optimization.

Finally, further research is required into understanding issues around compatibility of current hardware technologies for post-quantum cryptography with potential future quantum computing hardware platforms to ensure that the advances made maintain their efficiencies when deployed in practice in future. There are already ongoing efforts into the challenges involved in migrating to PQC for cybersecurity in the quantum era as exemplified by the ongoing work under the National Cybersecurity Centre of Excellence (NCCoE)'s Migration to PQC project [33].

REFERENCES

[1] P. W. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," *SIAM J. Comput.,* vol. 26, no. 5, pp. 1484–1509, 1997.

[2] L. Chen *et al.*, "Report on post-quantum cryptography," The National Institute of Standards and Technology, 2016. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/ir/2016/NIST.IR.8105.pdf

[3] G. Alagic *et al.*, "Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process," The National Institute of Standards and Technology, July 2022. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/ir/2022/NIST.IR.8413-upd1.pdf

[4] R. Avanzi *et al.*, "CRYSTALS-Kyber: Algorithm Specifications And Supporting Documentation," 2021. [Online]. Available: https://pq-crystals.org/kyber/data/kyber-specification-round3-20210131.pdf.

[5] A. Langlois and D. Stehle, "Worst-case to average-case reductions for module lattices," *Designs, Codes and Cryptography,* vol. 75, no. 3, pp. 565-599, 2015.

[6] W. Tan, Y. Lao, and K. K. Parhi, "KyberMat: Efficient Accelerator for Matrix-Vector Polynomial Multiplication in CRYSTALS-Kyber Scheme via NTT and Polyphase Decomposition," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2023, pp. 1-9.

[7] M. Li, J. Tian, X. Hu, and Z. Wang, "Reconfigurable and High-Efficiency Polynomial Multiplication Accelerator for CRYSTALS-Kyber," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 42, pp. 2540-2551, 2023.

[8] J. Mu *et al.*, "Scalable and Conflict-Free NTT Hardware Accelerator Design: Methodology, Proof, and Implementation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 42, no. 5, pp. 1504-1517, 2023.

[9] P. L. Montgomery, "Modular Multiplication Without Trial Division," *Mathematics of computation,* vol. 44, no. 170, pp. 519-521, 1985.

[10] P. Barrett, "Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor," presented at the Proceedings on Advances in cryptology---CRYPTO '86, Santa Barbara, California, USA, 1986.

[11] Y. Huang, M. Huang, Z. Lei, and J. Wu, "A pure hardware implementation of CRYSTALS-KYBER PQC algorithm through resource reuse," *IEICE Electron. Express,* vol. 17, 2020.

[12] P. Longa and M. Naehrig, "Speeding up the Number Theoretic Transform for Faster Ideal Lattice-Based Cryptography," in *International Conference on Cryptology and Network Security*, 2016: Springer, pp. 124-139.

[13] M. Bisheh-Niasar, R. Azarderakhsh, and M. Mozaffari-Kermani, "High-Speed NTT-based Polynomial Multiplication Accelerator for Post-Quantum Cryptography," in *2021 IEEE 28th Symposium on Computer Arithmetic (ARITH)*, June 2021, pp. 94-101.

[14] Z. Chen, Y. Ma, T. Chen, J. Lin, and J. Jing, "High-performance area-efficient polynomial ring processor for CRYSTALS-Kyber on FPGAs," *Integration,* vol. 78, pp. 25-35, 2021.

[15] L. Ma, X. Wu, and G. Bai, "Parallel polynomial multiplication optimized scheme for CRYSTALS-KYBER Post-Quantum Cryptosystem based on FPGA," in *2021 International Conference on Communications, Information System and Computer Engineering (CISCE)*, 2021: IEEE, pp. 361-365.

[16] F. Yaman, A. C. Mert, E. Ozturk, and E. Savas, "A Hardware Accelerator for Polynomial Multiplication Operation of CRYSTALS-KYBER PQC Scheme," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, France, 2021: IEEE, pp. 1020-1025.

[17] X. Yufei and L. Shuguo, "A Compact Hardware Implementation of CCA-Secure Key Exchange Mechanism CRYSTALS-KYBER on FPGA," *IACR Transactions on Cryptographic Hardware and Embedded Systems,* vol. 2021, no. 2, pp. 328–356, 2021.

[18] V. B. Dang, K. Mohajerani, and K. Gaj, "High-Speed Hardware Architectures and FPGA Benchmarking of CRYSTALS-Kyber, NTRU, and Saber," *IEEE transactions on computers,* vol. 72, no. 2, pp. 306-320, 2023.

[19] A. Sarker, M. Mozaffari-Kermani, and R. Azarderakhsh, "Hardware Constructions for Error Detection of Number-Theoretic Transform Utilized in Secure Cryptographic Architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 27, no. 3, pp. 738-741, 2019.

[20] A. Sarker, A. C. Canto, M. M. Kermani, and R. Azarderakhsh, "Error Detection Architectures for Hardware/Software Co-Design Approaches of Number-Theoretic Transform," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 42, no. 7, pp. 2418-2422, 2023.

[21] T. Pöppelmann and T. Güneysu, "Towards Practical Lattice-Based Public-Key Encryption on Reconfigurable Hardware," in *20th International Conference on Selected Areas in Cryptography, SAC 2013*, 2013, pp. 68-85.

[22] E. Fujisaki and T. Okamoto, "Secure Integration of Asymmetric and Symmetric Encryption Schemes," in *19th Annual International Cryptology Conference*, 1999, vol. 1666: Springer, pp. 537-554.

[23] H. J. Nussbaumer, *Fast Fourier transform and convolution algorithms*, 2 ed. Springer, 1982.

[24] V. Lyubashevsky and G. Seiler, "NTTRU: Truly Fast NTRU Using NTT," *IACR Transactions on Cryptographic Hardware and Embedded Systems,* no. 3, pp. 180-201, 2019.

[25] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of computation,* vol. 19, no. 90, pp. 249-259, 1965.

[26] W. M. Gentleman and G. Sande, "Fast fourier transforms—For fun and profit," in *AFIPS Conference Proceedings - 1966 Fall Joint Computer Conference, AFIPS 1966*, 1966, pp. 563-578.

[27] G. A. Jullien, "Implementation of Multiplication, Modulo a Prime Number, with Applications to Number Theoretic Transforms," *IEEE Transactions on Computers,* vol. C-29, no. 10, pp. 899-905, 1980.

[28] 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 transactions on cryptographic hardware and embedded systems,* pp. 49-72, 2020.

[29] W. Guo and S. Li, "Split-Radix Based Compact Hardware Architecture for CRYSTALS-Kyber," *IEEE Transactions on Computers,* vol. 73, no. 1, pp. 97-108, 2024.

[30] W. Guo and S. Li, "Highly-Efficient Hardware Architecture for CRYSTALS-Kyber With a Novel Conflict-Free Memory Access Pattern," *IEEE Transactions on Circuits and Systems I: Regular Papers,* vol. 70, no. 11, pp. 4505-4515, 2023.

[31] M. Bisheh-Niasar, R. Azarderakhsh, and M. Mozaffari-Kermani, "Instruction-Set Accelerated Implementation of CRYSTALS-Kyber," *IEEE transactions on circuits and systems. I, Regular papers,* vol. 68, no. 11, pp. 4648-4659, 2021.

[32] W. Guo, S. Li, and L. Kong, "An Efficient Implementation of KYBER," *IEEE Transactions on Circuits and Systems II: Express Briefs,* vol. 69, no. 3, pp. 1562-1566, 2022.

[33] C. Paquin, J. Goodman, J. Gray, and V. Krummel, "PANEL: NIST SP 1800-38C, Quantum Readiness: Testing Draft Standards for Interoperability and Performance," presented at the 5th NIST PQC Standardization Conference, Rockville, USA, April 2024. [Online]. Available: https://csrc.nist.gov/csrc/media/Presentations/2024/panel-nccoe-interoperability-and-performance/images-media/panel-nccoe-interoperability-pqc2024.pdf.

Abdullah Alhassani received his bachelor's degree in computer engineering from Umm Al-Qura University, Saudi Arabia, in 2008 and his master's degree in computer engineering from RMIT University, Australia, in 2011. He is currently pursuing his Ph.D. with the Department of Electronic and Electrical Engineering at University of Sheffield, UK. His research interests include post-quantum cryptography and the secure design and efficient implementation of cryptographic algorithms.

Mohammed Benaissa is currently a Professor of information engineering in the Department of Electronic and Electrical Engineering, University of Sheffield, Sheffield, UK. His research interests include finite number systems, signal processing and electronic system design. He has published over 200 papers on contributions in these areas and their application to security, communications and healthcare.