An efficient hardware implementation of number theoretic transform for CRYSTALS-Kyber post-quantum cryptography

Trang Hoang, Tu Dinh Anh Duong, Thinh Quang Do
Faculty of Electrical-Electronics, Ho Chi Minh City University of Technology, Vietnam National University Ho Chi Minh City, Ho Chi Minh City, Vietnam

Article Info

Article history:

Received Mar 17, 2024
Revised Oct 25, 2024
Accepted Nov 11, 2024

Keywords:

CRYSTALS-Kyber
Hardware implementation
Number theoretic transform
Polynomial multiplication
Post-quantum cryptography

ABSTRACT

CRYSTALS-Kyber was chosen to be the standardized key encapsulation mechanisms (KEMs) out of the finalists in the third round of the National Institute of Standards and Technology (NIST) post-quantum cryptography (PQC) standardization program. Since the number theoretic transform (NTT) was used to reduce the computational complexity of polynomial multiplication, it has always been a crucial arithmetic component in CRYSTALS-Kyber design. In this paper, a simple and efficient architecture for NTT is presented where we easily archived the functionality of polynomial multiplication with efficient computation time. Only 857 Look-Up Tables and 744 flip-flops were utilized in our NTT design, which consisted of two processing elements (PEs) and two butterfly cores within each PE.

Corresponding Author:

Trang Hoang
Faculty of Electrical-Electronics, Ho Chi Minh City University of Technology
Vietnam National University Ho Chi Minh City
268 Ly Thuong Kiet, District 10, Ho Chi Minh City, Vietnam
Email: hoangtrang@hcmut.edu.vn

1. INTRODUCTION

Since the application of Shor’s algorithm, classical public-key cryptography protocols have become increasingly vulnerable to quantum computer attacks [1]. CRYSTALS-Kyber is one of the finalists in the third round of post-quantum cryptography (PQC) algorithm evaluation by the National Institute of Standards and Technology (NIST), a competition to determine various algorithms to withstand attacks from quantum computers. Specifically, the algorithm is a lattice-based cryptosystem based on the module learning-with-errors problem (MLE). In July 2022, CRYSTALS-Kyber was selected as the standard Public-Key Encryption/Key Encapsulation Mechanism [2].

CRYSTALS-Kyber and other lattice-based cryptosystems extensively rely on polynomial multiplication. It serves as the primary operation, but its implementation might be computationally intensive, causing a bottleneck [3]. To overcome this issue, an algorithm based on number theoretic transform (NTT) for polynomial multiplication has been used. This algorithm is a typical method for calculating polynomial multiplication with less complex operations, thereby decreasing the computational burden of lattice-based cryptosystems. By applying the NTT-based polynomial multiplication method, the performance of lattice-based cryptosystems can be significantly enhanced, making them more applicable and effective in a range of contexts. Optimizing the complexity of hardware accelerators is one of the most critical factors in selecting an optimal hardware design approach for schemes. For the purpose of validating the efficacy of a specified design approach, benchmarks should be executed on a variety of implementation platforms. Despite the fact that software-implemented designs provide intuitive and user-friendly interfaces, their performance is typically inferior to that of other platforms [4]. A hybrid design of software and hardware was used to improve the performance while retaining the flexibility of the design with software only. Although the designs implemented on both platforms would be flexible and design-time efficient, they did not provide the same level of performance as the pure hardware design. In contrast, hardware implementation is characterized by the complexity of its algorithms and the lack of basic units written in hardware description languages, posing a challenge to designers who are tasked with creating the hardware for complex algorithms using basic logic gates. In return, pure hardware implementations would offer designers the best performance.

The hardware design encompasses a diverse array of devices with varying specifications, design constraints, and implementation settings. These factors make it difficult to fairly compare the hardware implementations of various designs. Application-specific integrated circuit (ASIC) and field-programmable gate array (FPGA) are the two primary architectures used for hardware implementation, and they are frequently employed in research analysis and development. The NIST recommends the use of Xilinx Artix-7 FPGA for hardware implementation in order to compare the design with greater precision [5]. ASIC results, on the other hand, are highly dependent on the technology library and system configurations used during the implementation process. This makes it difficult to compare designs to others, as the implementation details of different projects can vary significantly. Despite these obstacles, ASICs continue to be an important and widely employed tool in the field of hardware design, especially for large-scale and complex systems that require a high degree of precision and dependability [6]. For example, Banerjee *et al.* [6] presented a lattice cryptography processor with configurable parameters that had an NTT block accelerated by a single-port RAM-based memory architecture on an ASIC that aimed to optimize resources, but their design was inefficient, especially in terms of frequency and latency. Moreover, a low-power and resource-efficient NTT ASIC design was presented in [7]. Although the area and power results were good, the design did not offer a good trade-off between timing and resource efficiency. Song *et al.* [8] also proposed one of the quickest and most energy-efficient NTTs on ASIC, but with opaque resources, which could be interpreted as a trade-off between resources, timing, and power.

The comparisons on the FPGA side did not look good either, whereas designers used different FPGA families and there were usually conflicts between three aspects: resources, timing, and power. Fritzmann *et al.* [9], RISQ-V, a tightly coupled RISC-V accelerator for both ASIC and Xilinx Zynq-7000 FPGA was proposed. Karabulut and Aysu [10] also created a resource-optimized RISC-V NTT core on a Xilinx Virtex-7 FPGA, albeit with high timing latency. In [4] and [11], Xilinx Artix-7 was used as the implementation platform for the NTT design; however, the NTT design introduced in [4] was more timing-efficient than that in [11] due to the double number of butterfly cores, resulting in a higher frequency and lower latency. The frequency results for [9] and [10] were null. On the other hand, the designers omitted the resource results in [4] and [11], making it difficult to compare the design approach.

Mentioning FPGA resources, the number of look-up tables (LUTs) and flip flops (FFs) used in [6] is enormous due to the large number of multipliers, whereas designer could use the digital signal processing (DSP) module in the FPGA for integer multiplier to reduce the number of LUTs and DFFs that might be used. This may also balance the FPGA resources to conserve them for other logic components within the CRYSTALS-Kyber hardware. The NTT hardware described in [4] utilized only one block RAM (BRAM) for storing polynomials, resulting in a design with a high latency.

In summary, at the moment, most studies on NTT hardware implementation have their own weaknesses that could be addressed for better performance. All the aforementioned factors led us to design a Xilinx Artix-7 FPGA and ASIC NTT with balanced frequency, latency, and power efficiency. Our design also utilized the FPGA resource by incorporating the DSP and BRAM unit, which prevented the excessive use of LUT and FF and the risk of FPGA resource over-utilization when integrating the NTT design into the complete CRYSTALS-Kyber hardware design. Specifically, the use of DSP in our FPGA resulted in a 33% reduction in the number of LUTs and DFF used. Moreover, we used three BRAMs per butterfly core to store polynomials and twiddle factors, allowing our butterfly cores to operate in parallel and reducing latency by 64.5% relative to [4].

Contribution of this work:

Polynomial multiplication arithmetic is one of the most important components of the CRYSTALS Kyber hardware implementation since it has a significant impact on the latency and efficiency of the design [12]–[17]. By utilizing the NTT core to apply NTT-based polynomial multiplication, we may lower the operation complexity and boost throughput while consuming fewer resources. This paper presents a resource-efficient NTT hardware design with a balance of frequency, latency, and power. In comparison with other works that handle NTT in hardware implementation, we would like to prove our efficiency upon outstanding working frequency, improved timing operation, and less resource cost (LUT and FF) leading to power optimization.

- Pure hardware implementation with balanced benefits: We proposed a pure hardware capable of handling NTT and INTT (Inverse NTT) operations for the CRYSTALS-Kyber PQC algorithm with a balance of resources and performance. By optimizing the resources on the FPGA for the processing units and managing the memory access efficiently, we have been able to reduce the overall size and memory usage without slowing down the performance.
- Standard benchmark for future verification: By implementing on both FPGA (Xilinx Artix-7) and ASIC (TSMC 65nm technology), we offer our design as a standard benchmark for other researchers to clearly evaluate the NTT and INTT parts in their CRYSTALS-Kyber hardware designs. This aims to analyze the advantages and disadvantages of each design rationale for CRYSTALS-Kyber hardware, which helps designers soon determine and develop the most suitable hardware design approach of their own.
- Outstanding performance in comparison to previous studies: Our results on Xilinx Artix-7 made use of the DSPs and BRAMs of the FPGA so that the number of LUTs and FFs logic was small in comparison to [9], [6], and the computation time was reduced compared to [4], [11]. By utilizing efficient memory access, our ASIC results outperformed other NTT hardware designs in [6], [7].

The remainder of this paper is organized as follows:

- Section 2 explains the mathematical foundations of the NTT and INTT algorithms in CRYSTALS-Kyber, which contains a number of algorithms in pseudocode forms and their descriptions in detail.
- In Section 3, we describe our strategy for hardware design and the overall module architecture, where major components are depicted, including the polynomial multiplications, butterfly units, memory address controls and the overall design containing them.
- The findings and comparisons with other designs are analyzed in Section 4. Most of the results were obtained from the synthesis and implementation processes since the design was based on hardware. Parallel implementation on FPGA and ASIC was executed for mutual discussions as well as comparison to other work for efficiency.
- The conclusion of this work is presented in the final section, where the study is summarized, and some implications can be stated.

2. BACKGROUND

2.1. NTT-based polynomial multiplication

Efficient polynomial multiplication, especially for large degrees, is fundamental in encryption and lattice-based cryptography but can be time-consuming though. In Kyber, polynomials are defined in the ring $\mathbb{Z}_q[x]/(x^n+1)$. The operation takes polynomials $A(x)=\sum_{i=0}^{n-1}(a_i x^i)$ and $B(x)=\sum_{i=0}^{n-1}(b_i x^i)$ as inputs and return the output polynomial $C(x)=\sum_{i=0}^{n-1}(c_i x^i)$ as multiplication output. As usual, the technique for multiplying two polynomials has a $o(n^2)$ complexity, which leads to slow processing time when dealing with large-degree polynomials. NTT is used to speed up this operation since NTT performs it with $o(n.logn)$ complexity.

The resultant polynomial $C(x)$ can be further reduced with negative wrapped convolution technique shown within Algorithm 1, whereas, the reduction polynomial, $\phi(x)=(x^n+1)$ and $q\equiv 1\ (mod\ 2n)$. This technique hence directly reduces the degree of the resulting polynomial $C(x)$ to degree $n-1$, which is accomplished by multiplying the coefficients of the input and output polynomials by the power of $\Psi \epsilon Z_q$ and $\Psi^{-1}\epsilon Z_q$, respectively, where Ψ is a primitive $2n$-th root of unity in Z_q satisfying $\Psi^{2n}\equiv 1(mod\ q)$ and $\forall i<2n, \Psi^i \neq 1\ (mod\ q)$, when $q\equiv 1\ (mod\ 2n)$ [18].

Algorithm 1. NTT-based Polynomial Multiplication with negative wrapped convolution (NWT) [16]

```
    Input: $A(x), B(x) \in Z_q[x]/(x^n+1)$
    Input: Primitive $2n$-th root of unity $\Psi \in Z_q$
    Output: $C(x) = A(x) \times B(x), C(x) \in Z_q[x]/(x^n+1)$
1 $\hat{A} = (a_0, a_1, \ldots, a_{n-1}) \odot (1, \Psi^1, \Psi^2, \ldots, \Psi^{n-1})$
2 $\hat{B} = (b_0, b_1, \ldots, b_{n-1}) \odot (1, \Psi^1, \Psi^2, \ldots, \Psi^{n-1})$
3 $\bar{A} = NTT(\hat{A})$
4 $\bar{B} = NTT(\hat{B})$
5 $\bar{C} = \bar{A} \odot \bar{B}$
6 $\hat{C} = INTT(\bar{C})$
7 $C(x) = (\hat{c}_0, \hat{c}_1, \ldots, \hat{c}_{n-1}) \odot (1, \Psi^{-1}, \Psi^{-2}, \ldots, \Psi^{-(n-1)})$
8 return $C(x)$
```

2.2. Number theoretic transformation

An $(n-1)$ degree polynomial $A(x)=\sum_{i=0}^{n-1} a_i x^i$ is transformed into NTT domain as $\bar{A}(x)=\sum_{i=0}^{n-1} \mathrm{A}_i x^i$ by using a n-pt forward NTT operation. The coefficient of $\bar{A}(x)$ in NTT domain is $\mathrm{A}_i =$

$\sum_{j=0}^{n-1} a_i\omega^{ij}$ over $\mathbb{Z}_q$ for $i = 0,1,\ldots,n-1$. After pointwise multiplying $\bar{A}(x)$ and $\bar{B}(x)$ in NTT domain, an n-point INNT is used to transform the result back to the polynomial domain with $a_i = n^{-1}\sum_{j=0}^{n-1} \mathrm{A}_i\omega^{-ij}$ in $\mathbb{Z}_q$. Moreover, NTT and INTT regularly use the twiddle factor, $\omega \in \mathbb{Z}_q$ and its modular inverse, $\omega^{-1} \in \mathbb{Z}_q$ as input. It is a primitive n-th root of unity in $\mathbb{Z}_q$ and the conditions $\omega^n \equiv 1 \pmod q$ and $\forall i < n, \omega^i \neq 1 \pmod q$, where $q \equiv 1\ (mod\ n)$.

In this study, the iterative NTT (Algorithm 2) and INNT (Algorithm 3), which utilize the Gentleman-Sande butterfly phenomenon, were applied. The NTT and INTT algorithms transform the polynomial from normal order to bit-reversed order and vice versa. The bit-reversal operation on $(l-1)$-bit integer k, where $l = log\ log\ n$ is executed by the $br(k, l-1)$ operation in Algorithm 3 [18]. In addition, the coefficients of the output polynomial must always be multiplied by $(1/n)\ mod\ q$ in the INTT operation since the bit reversion, as shown in steps 19-21 of Algorithm 3. Algorithm 2 can also be further modified to implement the INTT operation by substituting ω with ω^{-1} and having the output polynomial coefficients divided by n in Z_q.

Algorithm 2. Iterative NTT Algorithm [14]

```
   Input: $A(x) \in Z_q[x]/(x^n+1)$ in normal order
   Input: $\omega \in Z_q, n = 2^l$
   Output: $\bar{A}(x) = NNT(A) \in Z_q[x]/(x^n+1)$ in bit-reversed order
1  for $i$ from 1 by 1 to $l$ do
2  |    $m = 2^{l-i}$
3  |    for $j$ from 0 by 1 to $2^{i-1}-1$ do
4  |    |    for $k$ from 0 by 1 to $m-1$ do
5  |    |    |    $i_e \leftarrow 2 \cdot j \cdot m + k$
6  |    |    |    $i_o \leftarrow 2 \cdot j \cdot m + k + m$
7  |    |    |    $i_w \leftarrow 2^{i-1} \cdot k$
8  |    |    |    $U \leftarrow A[i_e]$
9  |    |    |    $V \leftarrow A[i_o]$
10 |    |    |    $W \leftarrow \omega^{i\omega}\ mod\ q$
11 |    |    |    $E \leftarrow (U+V)\ mod\ q$
12 |    |    |    $O \leftarrow (U-V) \cdot W\ mod\ q$
13 |    |    |    $A[i_e] \leftarrow E$
14 |    |    |    $A[i_o] \leftarrow O$
15 |    |    end for
16 |    end for
17 end for
18 return $A$
```

Algorithm 3. Iterative INTT Algorithm [14]

```
   Input: $\bar{A}(x) \in Z_q[x]/(x^n+1)$ in bit-reversed order
   Input: $\omega^{-1} \in Z_q, n = 2^l$
   Output: $A(x) = INNT(A) \in Z_q[x]/(x^n+1)$ in normal order
1  $m = 1$
2  $v = n$
3  while $v > 1$ do
4  |    for $i$ from 0 by 1 to $m-1$ do
5  |    |    $k = 0$
6  |    |    for $j$ from $i$ by $2 \cdot m$ to $(n-2)$ do
7  |    |    |    $U \leftarrow A[j]$
8  |    |    |    $V \leftarrow A[j+m]$
9  |    |    |    $E \leftarrow (U+V)\ mod\ q$
10 |    |    |    $O \leftarrow (U-V) \cdot \omega^{-br(k,l-1)}\ mod\ q$
11 |    |    |    $A[j] \leftarrow E$
12 |    |    |    $A[j+m] \leftarrow O$
13 |    |    |    $k = k+1$
14 |    |    end for
15 |    end for
16 |    $m = 2 \cdot m$
17 |    $v = v/2$
18 end while
19 for $i$ from 0 by 1 to $n-1$ do
20 |    $A[i] \leftarrow A[i] \cdot (1/n)\ (mod\ q)$
21 end for
22 return $A$
```

3. DESIGN RATIONALE

In this section, our hardware design approach and the overall architecture of the main modules are given. As described in Section 3.1, the multiplier using Montgomery's modular algorithm at the word level performs the modular multiplication in our hardware. The primary arithmetic operation in NTT and INTT algorithms is the butterfly operation; We describe our processing elements hardware with the butterfly unit in Section 3.2. In Section 3.3, our efficient memory and its address generator are given, which enhances the performance of our design. Section 3.4 describes the overall structure. The proposed architecture can be implemented using either FPGA or ASIC technology.

3.1. NTT-based polynomial multiplication

Modular multiplier is one of the most essential components in NTT system. The aforementioned component involves two different blocks: a DSP-based integer multiplication unit and a word-level Montgomery modular reduction unit. Both blocks are independent of the other. The Montgomery [18] and Barrett algorithms [6] are the most common ones used in modular reduction, and they are designed to achieve efficiency. For this design, we decided to build a Montgomery reduction unit, since the algorithm uses fewer components, and processes through fewer stages in comparison to Barrett's; therefore, it can be hardware implemented into smaller designs in terms of area, as well as can work in larger frequencies. The unit hence can be used for the word-level Montgomery modular reduction operation for modulus-satisfying $q \equiv 1 \ (mod\ 2n)$.

As shown in Figure 1, the DSP unit of the FPGA is utilized for the integer multiplier unit to cut down on the number of LUTs and FFs. Moreover, the integer multiplication unit that has been proposed used pipelining technique to ensure that the product of the multiplication is synchronized with the system. The output of the multiplier, however, needs to have its bit length brought down to match that of the modulus, which we call the reduction operation. The Montgomery modular word-level algorithm of that operation is provided in Algorithm 4. Any NTT prime q, possessing the $q \equiv 1 \ (mod\ 2n)$ property when the negative wrapped convolution method is applied, can be written as $q = q_H \cdot 2^w + 1$; and by harnessing this property, a Montgomery reduction operation can be performed at the word level with the word size $w = (2n)$. This property allows the reduction operation to be handled in multiple stages, rather than running it all at once. To perform the modular reduction operation on a K-bit modulus, $L = K/w$ iterations are also required. The Montgomery modular reduction constant, which was previously written as $\mu = -q^{-1}\ mod\ 2^w$), is now written as $-1\ mod\ 2^w$. This change makes it possible to use a simple two's complement operation instead of a multiplication operation $A \cdot B \cdot \mu\ (mod\ 2^w)$ in the Montgomery scheme, as demonstrated in Step 6 of Algorithm 4. In each NTT component of CRYSTALS-Kyber, a fixed modulus $q = 3329$ is used [19], while the parameters $n = 128$ and $K = 12$, q can be written as $q_H \cdot 2^8 + 1$, with word size $w = 8$. So, we need $L = 12/8 = 2$ iterations for the algorithm to work. R^{-1} can be calculated as $R^{-1} = {q_H}^2 = 13^2 = 169$ in CRYSTALS-Kyber.

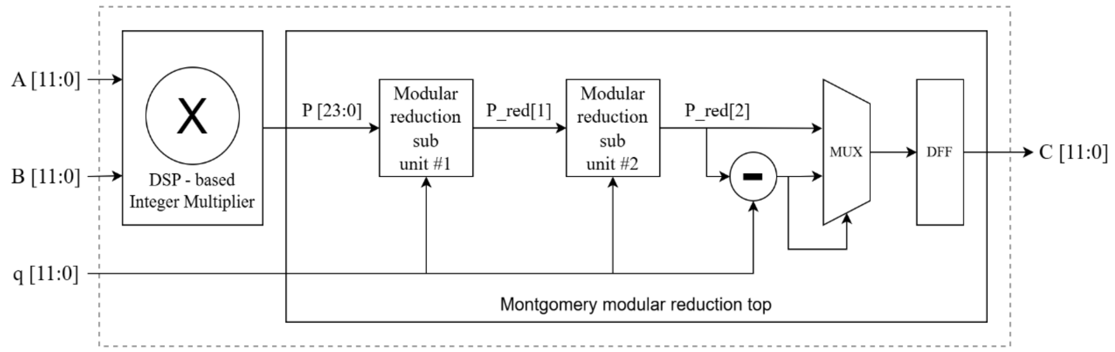

A [11:0]
B [11:0]
q [11:0]
X
DSP - based Integer Multiplier
P [23:0]
Modular reduction sub unit #1
P_red[1]
Modular reduction sub unit #2
P_red[2]
MUX
DFF
C [11:0]
Montgomery modular reduction top

Figure 1. 12-bit montgomery modular multiplier

Algorithm 4. Word-Level Montgomery Reduction Algorithm for NTT-friendly modulus [20] [21]

```
    Input: $C = A \cdot B$ (a $2K$-bit positive integer)
    Input: $q$ (a $K$-bit modulus), $q = q_H \cdot 2^w + 1$
    Input: $\boldsymbol{w} = (2n)$ (word size)
    Output: $\boldsymbol{Res} = \boldsymbol{C} \cdot R^{-1}\ (mod\ q)$ where $R = 2^{w \cdot L}$
```

```
1  $L \leftarrow K/w$
2  $T \leftarrow C$
3  **for** $i$ from 0 to $L$ **do**
4  |    $T1_H \leftarrow T >> w$
5  |    $T1_L \leftarrow T\ (mod\ 2^w)$
6  |    $T2 \leftarrow two's\ complement\ of\ T1_L$
7  |    $C_{in} \leftarrow T2[w-1]\ \vee\ T1_L[w-1]$
8  |    $T \leftarrow T1_H + (q_H \cdot T2[w-1:0]) + C_{in}$
9  **end for**
10 $T4 \leftarrow T - q$
11 **if** $(T4 < 0)$ **then** $Res = T$ **else** $Res = T4$
12 return $Res$
```

As can be seen in Step 8 of Algorithm 4, the word-level Montgomery modular reduction algorithm makes use of a number of multiply and accumulate (MAC) operation units, which is responsible for performing the X ·Y +Z+Cin operation. Figure 2 depicts the hardware architecture that was developed for the word-level Montgomery modular reduction algorithm. This architecture includes two Modulo Reduction sub-blocks. The first Modulo Reduction sub-block performs a reduction that takes the 24-bit input data P and transforms it into the 16-bit intermediate data P_red[1]. After that, the data is reduced by the second Modulo Reduction sub-block for a second time in order to obtain the 14-bit data P_red[2]. By utilizing a subtractor and a multiplexer, the modular value of P_red[2] is obtained, which is also the output of the Montgomery modular reduction algorithm at the word level.

To be more specific, the Modulo Reduction Sub's proposed hardware is shown in Figure 2, which is the hardware implementation for Step 4 to Step 8 of Algorithm 4. The m-bit input data T1 is divided into two parts: T2L = T1[7:0] (8 last bits) and T2H_t = T1[m:8] (the rest) due to the word size w=8 of the Montgomery reduction algorithm. One adder is also used to calculate the mult t, carry t, and T2H t, therefore the reduction result C t can be obtained.

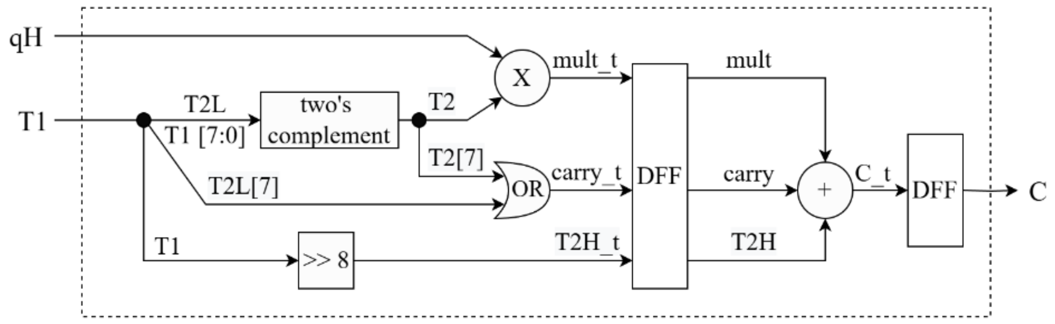


Figure 2. Modular reduction sub-unit hardware

3.2. Butterfly unit

After we had completed an effective implementation for the modular arithmetic, the construction of the hardware for the butterfly units was concentrated. These butterfly units make use of modular operations and are located within the PEs (processing elements). The butterfly operation is conducted by the PEs, each of which receives one twiddle factor and two coefficients as inputs. Each PE then generates two resulting coefficients, which are referred to as the odd (O) and even (E) coefficients, as outputs. As can be seen in Figure 3, the proposed PE module to implement the butterfly operation consists of one modular adder, one modular subtractor, and one modular multiplier. In each PE, three dual-port BRAMs are used for necessary data storage. One of them is called the twiddle factor BRAM (TW BRAM), while the other two are called the input and intermediate coefficient BRAM (both are called DATA BRAM).

The even coefficient output of the PE is the output of the modular adder, while the odd coefficient output of the PE is the output of the modular subtractor and multiplier, as shown in steps 11-12 of Algorithm 2. To maintain synchronization between the output of the odd and even coefficients, additional flip-flops were inserted at the modular adder hardware output. For an n-point NTT operation, the maximum number of processing units that can be included in the design is equal to $n/2$, and the number of processing units must be a power of 2. In the design that we have proposed, we make use of 2 PEs, which indicates that the NTT operation is carried out through two butterfly units. The whole hardware design for a typical PE, which contains a butterfly unit, is demonstrated in Figure 3.

By assigning the value 0 to the input signal in0, the first multiplicand to the input signal in1, and the second multiplicand to the input signal in mult, it is possible to utilize the butterfly hardware to handle a modular multiplication operation, as illustrated in Figure 3. Here, we referred to the 2-input butterfly unit as an NTT2 unit. To compute the modular addition and subtraction of the two data sets, the NTT2 unit processes two sets of 12-bit input data called in0 and in1. The modular sum is synchronized after it is passed through a shift register, and then the value is used as the even index output of the NTT2 device. For the modular multiplier unit to produce the 12-bit output data MODout, its inputs consist of the result of the modular subtraction, the modulo q, and the input data MULin, and one DFF is used to synchronize this signal to obtain the odd index output of the NTT2.

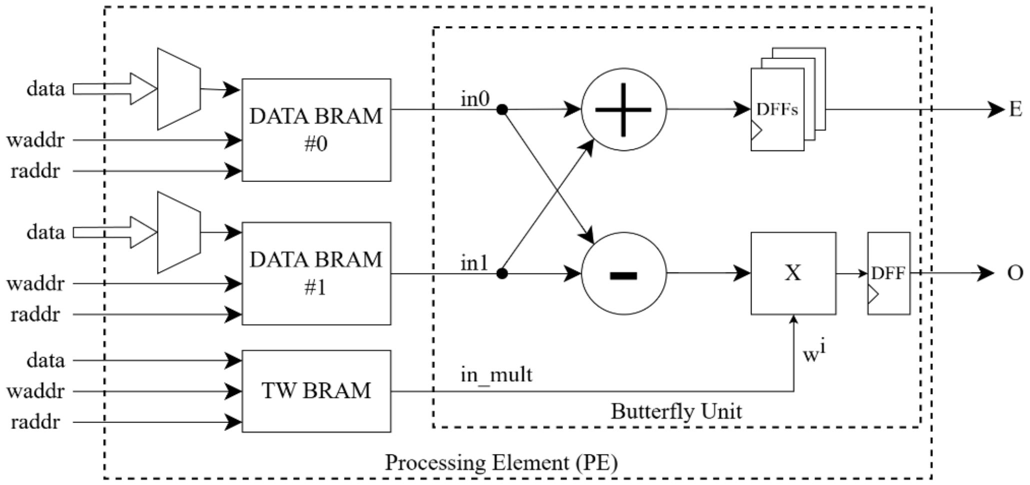


Figure 3. Processing Element and Butterfly Unit hardware

3.3. Memory access and address generator

In CRYSTALS-Kyber's NTT & INTT hardware, accessing the memory needs to be implemented well and orderly to avoid bottleneck problems [22]–[25]. The Iterative NTT, which is illustrated in Algorithm 2, consists of n stages, and there are $n/2$ butterfly operations used in each stage. Additionally, the read address pattern for the input coefficients varies from stage to stage. The calculation of the index occurs between Steps 5 and 6 of Algorithm 2 in the NTT processing flow. To have control over the BRAMs presenting in each PE, an address generator is required. This unit grants the NTT block the ability to read the input coefficients for the process of the current NTT stage, and it also grants the NTT block the ability to store the output coefficients in the appropriate index order for the subsequent NTT stages. The state diagram for the address generator can be seen in Figure 4, which is a finite-state machine, suitable for hardware implementation.

There are three states in the address generator: IDLE, NTT, and WAIT state. In NTT state, read addresses for the input coefficients and the corresponding twiddle factor, ω^i, are generated for the PEs to perform NTT processing. The write addresses are also generated to store the NTT output coefficients in the BRAMs, these coefficients are then used as inputs for the next NTT stage. There are 7 stages in a 128-pt NTT, so the NTT state (STATE 1) is iterated 7 times to get the final NTT result, as shown in Step 1 of Algorithm 2. The states between these NTT states are called WAIT states. These states start after completing the generation of the read addresses and end when all the output coefficients are stored in the BRAMs with the generated write addresses. The next NTT stage process can start only if the current WAIT state finishes. After 7 WAIT states, which correspond to 7 respective NTT states, the system can transit from the final WAIT state back to IDLE state for the next NTT process.

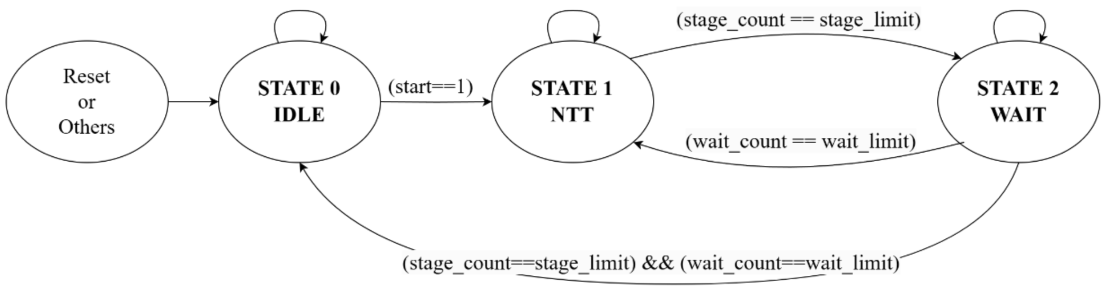

Reset
or
Others
STATE 0
IDLE
(start==1)
STATE 1
NTT
(stage_count == stage_limit)
(wait_count == wait_limit)
STATE 2
WAIT
(stage_count==stage_limit) && (wait_count==wait_limit)

Figure 4. Address Generator state diagram

3.4. Overall design

Due to the property that an n-pt NTT operation can be implemented by two $(n/2)$-pt NTT operation, we can implement CRYSTALS-Kyber's 256-pt NTT operation by using two separated classic 128-pt NTT, of which algorithm is shown in Algorithm 2 (Figure 5). These two NTTs will gather their data allocated in the BRAMS, where data in and out of the NTT block are stored. An address generator is also required to instruct the BRAMS to output the correct information for each NTT respectively.

Before starting the NTT/INTT operation, the hardware stores twiddle factors (for NTT operation), modular inverse twiddle factors (for INNT operation), and the input coefficients into the BRAMs of each PE. The twiddle factor and its modular inverse values are stored in 2 BRAM, while the input coefficients are stored in 4 BRAMS: 2 BRAMs for the first PE and 2 BRAMs for the second PE. After the NTT2 operation, the output coefficients at the current stage are then stored back to the same 4 BRAMs to be used as input coefficients of the next stage. This data storing process is handled at the top level using the generated read and write addresses from the address generator as mentioned above. After finishing 7 NTT stages, the DOUT BLOCK unit would set the done signal to high to indicate the completion of NTT/INTT operation, while also passing the output coefficients data through the dout signal. Figure 5 shows our proposed NTT overall design structure.

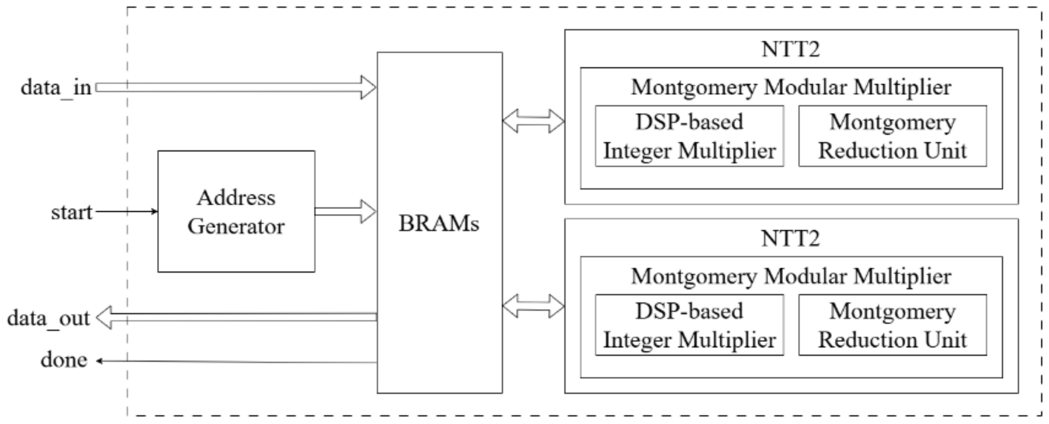

data_in
start
Address Generator
BRAMs
data_out
done
NTT2
Montgomery Modular Multiplier
DSP-based Integer Multiplier
Montgomery Reduction Unit
NTT2
Montgomery Modular Multiplier
DSP-based Integer Multiplier
Montgomery Reduction Unit

Figure 5. NTT top overall architecture

4. EXPERIMENTAL RESULTS AND COMPARISON

4.1. Experimental scenario

Our hardware design was written in Verilog Hardware Description Language, the most common one for hardware implementation in the meantime. On the other hand, our design was synthesized and implemented using Xilinx Vivado tools, on the Xilinx Artix-7 FPGA of name xc7a12tcpg238-3, specifically. In parallel, the design was also synthesized and post-synthesis verified using Synopsys Design Compiler and Synopsys Formality tools with the TSMC 65nm library, as a method for comparison. The NTT software for the reference model was written in Python and based on the NIST submissions' reference C source code of the CRYSTALS Kyber developing team.

4.2. Experimental scenario

Our proposed design was synthesized with the CRYSTALS-Kyber parameters of q=3329 and n=256. There were 2 PEs used in our design and each of those had 2 butterfly units. The synthesis result using Synopsys Design Compiler tools with TSMC 65nm technology library is shown in Table 1.

One of the most crucial and precious factors of the synthesized design was its maximum frequency to work properly. The value of our design was approximately 497 MHz, which outweighs most of the cryptosystems at the current time. The value was measured by changing the frequency value in the sdc (Synopsys Design Constraints) file and re-run the synthesis process until we got the largest value possible to fulfill the setup and hold constraints. Meanwhile, on the area side, the gate count value of our design on the TSMC 65nm library was around 472K with most of the used area for non-combinational logic (64.8%), while the combinational logic accounted for 35.2% of the total area.

From another perspective, Table 2 reports the synthesis and implementation results using Xilinx Vivado tools. which resulted in an area-friendly NTT design with a relatively high frequency of 102 MHz. The value was moderately lower than the value of 497 MHz mentioned above, which can be explained by the fact that the hardware options for FPGA are much more hindered in comparison to that for ASIC. In contrast, the number of LUTs used in the implementation design was much less in comparison to the synthesis design, since the better physical optimization and full implementation work of the FPGA. Additionally, the design only used 4.65% of the total amount of usable register and there was no latch generated in our design. This can be seen as an optimist result in the area and timing efficiency

Table 1. Synthesis Result on ASIC

Library		TSMC65nm
Frequency (MHz)		497
Area (μm^2)	Total	906113.7472
	Combinational	319100.1598
	Buf/Inv	6389.75997
	Noncombinational	587013.5874

Table 2. Synthesis and Implementation results on FPGA

Design		Synthesis	Implementation
Frequency (MHz)		102	102
Resources	LUT	872	857
	REG	744	744
	DSP	6	6
	BRAM	3	3

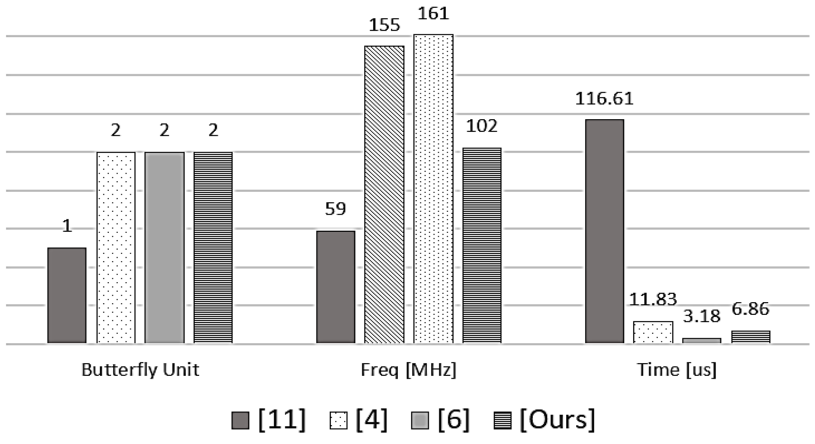
4.3. Comparison to prior work

Table 3 and Figure 6 compare the results in our work and previous works. The relationship between the number of butterfly units used in the design and implementation timing results are shown in Table 3 and Figure 6(a). The more butterfly units there were, the more work was shared for each butterfly unit, which hence decreased the NTT/INTT processing time and increased the maximum working frequency of the implementation design. Using two of these units, it can be seen that our works far surpassed [11], which only had one, in terms of operating frequency as well as latency. Our hardware design's resource efficiency can be affirmed by the comparative analysis of resource utilization among the designs featured in Table 3 and Figure 6(b). This achievement was a result of the strategic optimization of FPGA DSP for calculating operations and the reduction of necessary BRAM blocks for memory ones. Regarding the timing and performance, the maximum frequency of our work was 102 MHz, which is relatively medium compared to other work (59, 155, and 161 MHz). However, by generating the read and write address efficiently accessing the memory, the latency of our design was much smaller than other studies in [11], [4], and [6], with the respective figures being 6.86, 116.61, 11.83, and 3.18 microseconds.

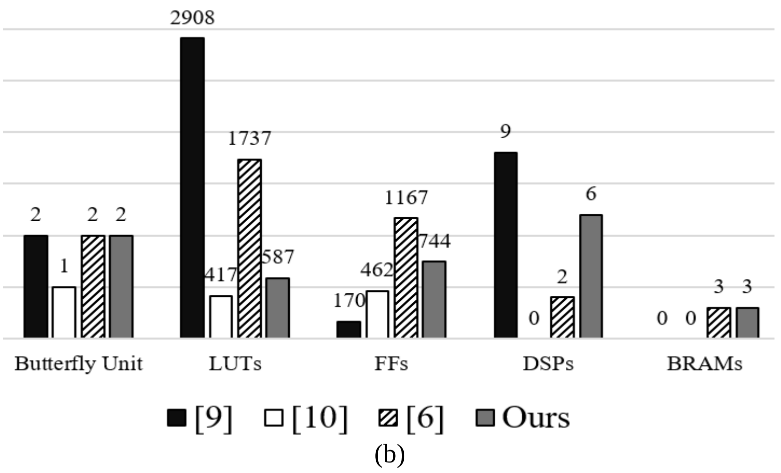
Our ASIC design, as shown in Table 4, by utilizing coherent design schemes and pipelining, achieved a frequency of 497 MHz, and far surpassed the NTT hardware designs in [6] and [7]. However, this optimization came at the cost of a high gate count of 472K since the trade-off between area and timing properties. Our design's architecture though resulted in a relatively small latency of just 686 and an insignificant processing time of 1.38 microseconds for the NTT operation. This latency and operation time are notably lower than other works such as [6] and [7], primarily due to our well-planned memory read and write scheme that employed the address generator block. Despite our design's relatively small latency for the NTT operation, that same value was larger than that of [8] (0.5 microseconds), which was achieved by utilizing a significant number of resources (area) in the synthesized design. Nevertheless, our design struck an overall balance between area and timing when compared to other works by having an outstanding performance on ASIC, without using up a large number of resources, rendering it a brighter solution for designs that orient performance on ASIC platform.

Table 3. Comparison of implementation results for NTT design (q=3329) on FPGA

Work	[9]	[10]	[11]	[4]	[6]	Ours
Platform	Zynq 7000	Virtex 7	Artix 7	Artix 7	Artix 7	Artix 7
Butterfly	2	1	1	2	2	2
NTT/INTT latency [CCs]	1935/1930	43756/-	6868/6367	1834/-	512/576	686/842
Freq [MHz]	-	-	59	155	161	102
Time [us]	-	-	116.61	11.83	3.18	6.86
LUTs	2908	417	-	-	1737	587
FFs	170	462	-	-	1167	744
DSPs	9	0	-	-	2	6
BRAMs	0	0	-	-	3	3



(a)



(b)

Figure 6. Comparison of NTT implementation on FPGA regarding (a) timings and (b) resources

Table 4. Comparison of synthesis results for NTT design (q=3329) ASIC flow

Work	Platform	n	q	NTT latency (CCs)	Freq (MHz)	Time (us)	Gate count
[6]	40nm CMOS	256	13	1289	72	17	106K
[8]	40nm CMOS	256	13	160	300	0.5	-
[7]	UMC 65 nm	256	13	2056	25	82	14K
Ours	TSMC 65 nm	256	12	686	497	1.38	472K

5. CONCLUSION

In this paper, we designed a fast and efficient architecture for NTT-based polynomial architecture settings that are suitable for the CRYSTALS-Kyber key encapsulation module (KEM). To the best of our knowledge, our proposed design was able to perform the NTT/INTT operations with one of the lowest latencies in the literature, while obtaining optimistic results on other aspects of design like area and resources required. This also implies the effectiveness of hardware implementation for complex cryptography algorithms, especially ones for the post-quantum era, allowing us to establish secure communication in the future. Overall, our NTTcore can perform CRYSTALS-Kyber's NTT/INTT operations at 102 MHz on an Atrix-7 FPGA, with a latency of 3.43 µs for the NTT operation and 4.21 µs for the INTT operation, while the respective figures for ASIC platform with the TSMC 65nm technology being 497 MHz for maximum working frequency, 1.38 µs for latency, and 472K gates for gate count. The study hence implied that CRYSTALS-Kyber can be efficiently implemented in hardware and might be produced in industrial. Future research might be based on this study to improve further the NTT structure, or to fully construct the CRYSTALS-Kyber hardware structure, as well as continue optimizing any cryptography design in terms of hardware efficiency.

ACKNOWLEDGEMENTS

We acknowledge Ho Chi Minh City University of Technology (HCMUT), VNU-HCM for supporting this study.

REFERENCE

[1] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, 1994, pp. 124–134, doi: 10.1109/SFCS.1994.365700.

[2] "Selected Algorithms 2022-post-quantum cryptography," *NIST Computer Security Resource Center*. Accessed: Apr. 17, 2023. [Online]. Available: https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022

[3] A. C. Mert, E. Karabulut, E. Ozturk, E. Savas, M. Becchi, and A. Aysu, "A Flexible and Scalable NTT hardware: applications from homomorphically encrypted deep learning to post-quantum cryptography," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Mar. 2020, pp. 346–351, doi: 10.23919/DATE48585.2020.9116470.

[4] 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. 17, no. 17, pp. 1–6, 2020, doi: 10.1587/ELEX.17.20200234.

[5] M. Bisheh-Niasar, R. Azarderakhsh, and M. Mozaffari-Kermani, "A monolithic hardware implementation of kyber: comparing apples to apples in PQC candidates," in *Progress in Cryptology–LATINCRYPT 2021: 7th International Conference on Cryptology and Information Security in Latin America, Bogotá, Colombia*, Colombia: Springer International Publishing, 2021, pp. 108–126, doi: 10.1007/978-3-030-88238-9_6.

[6] U. Banerjee, T. S. Ukyab, and A. P. Chandrakasan, "Sapphire: A configurable crypto-processor for post-quantum lattice-based protocols," *arXiv preprint:1910.07557*, 2019.

[7] T. Fritzmann and J. Sepulveda, "Efficient and flexible low-power NTT for lattice-based cryptography," in *Proceedings of the 2019 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2019*, 2019, pp. 141–150, doi: 10.1109/HST.2019.8741027.

[8] S. Song, W. Tang, T. Chen, and Z. Zhang, "LEIA: A 2.05mm 2 140mW lattice encryption instruction accelerator in 40nm CMOS," in *2018 IEEE Custom Integrated Circuits Conference (CICC)*, Apr. 2018, pp. 1–4, doi: 10.1109/CICC.2018.8357070.

[9] T. Fritzmann, G. Sigl, and J. Sepúlveda, "RISQ-V: tightly coupled RISC-V accelerators for post-quantum cryptography," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2020, no. 4, pp. 239–280, 2020, doi: 10.46586/tches.v2020.i4.239-280.

[10] E. Karabulut and A. Aysu, "RANTT: a risc-v architecture extension for the number theoretic transform," in *Proceedings - 30th International Conference on Field-Programmable Logic and Applications, FPL 2020*, 2020, pp. 26–32, doi: 10.1109/FPL50879.2020.00016.

[11] E. Alkim, H. Evkan, N. Lahr, R. Niederhagen, and R. Petri, "Isa extensions for finite field arithmetic accelerating kyber and newhope on risc-v," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2020, no. 3, pp. 219–242, 2020, doi: 10.13154/tches.v2020.i3.219-242.

[12] 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)*, Jun. 2021, pp. 94–101, doi: 10.1109/ARITH51176.2021.00028.

[13] G. Song, K. Jang, S. Eum, M. Sim, and H. Seo, "NTT and inverse NTT quantum circuits in CRYSTALS-Kyber for post-quantum security evaluation," *Applied Sciences*, vol. 13, no. 18, p. 10373, Sep. 2023, doi: 10.3390/app131810373.

[14] Y. Itabashi, R. Ueno, and N. Homma, "Efficient modular polynomial multiplier for NTT accelerator of crystals-kyber," in *2022 25th Euromicro Conference on Digital System Design (DSD)*, Aug. 2022, pp. 528–533, doi: 10.1109/DSD57027.2022.00076.

[15] H. Nguyen and L. Tran, "Design of polynomial NTT and INTT accelerator for post-quantum cryptography CRYSTALS-Kyber," *Arabian Journal for Science and Engineering*, vol. 48, no. 2, pp. 1527–1536, Feb. 2023, doi: 10.1007/s13369-022-06928-w.

[16] T. Pöppelmann and T. Güneysu, "Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2012, vol. 7533 LNCS, pp. 139–158, doi: 10.1007/978-3-642-33481-8_8.

[17] P. Longa and M. Naehrig, "Speeding up the number theoretic transform for faster ideal lattice-based cryptography," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2016, vol. 10052 LNCS, pp. 124–139, doi: 10.1007/978-3-319-48965-0_8.

[18] A. C. Mert, E. Ozturk, and E. Savas, “Design and implementation of a fast and scalable ntt-based polynomial multiplier architecture,” in *Proceedings - Euromicro Conference on Digital System Design, DSD 2019*, 2019, pp. 253–260, doi: 10.1109/DSD.2019.00045.

[19] T. L. Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz and D. S. Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, *Algorithm specifications and supporting documentation (version 3.01)*. CRYSTALS-Kyber, 2021.

[20] A. C. Mert, E. Ozturk, and E. Savas, “Design and implementation of encryption/decryption architectures for BFV homomorphic encryption scheme,” *IEEE Transactions on Very Large-Scale Integration (VLSI) Systems*, vol. 28, no. 2, pp. 353–362, 2020, doi: 10.1109/TVLSI.2019.2943127.

[21] P. L. Montgomery, “Modular multiplication without trial division,” *Mathematics of Computation*, vol. 44, no. 170, pp. 519–521, 1985, doi: 10.2307/2007970.

[22] Z. Chen, Y. Ma, T. Chen, J. Lin, and J. Jing, “Towards efficient kyber on FPGAs: A processor for vector of polynomials,” in *Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC*, 2020, vol. 2020-January, pp. 247–252, doi: 10.1109/ASP-DAC47756.2020.9045459.

[23] Z. Ni, A. Khalid, D.-S. Kundi, M. O’Neill, and W. Liu, “HPKA: a high-performance CRYSTALS-kyber accelerator exploring efficient pipelining,” *IEEE Transactions on Computers*, vol. 72, no. 12, pp. 3340–3353, Dec. 2023, doi: 10.1109/TC.2023.3296899.

[24] 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, no. 8, pp. 2540–2551, Aug. 2023, doi: 10.1109/TCAD.2022.3230359.

[25] S. Khan *et al.*, “Efficient, error-resistant NTT architectures for CRYSTALS-Kyber FPGA accelerators,” in *2023 IFIP/IEEE 31st International Conference on Very Large-Scale Integration (VLSI-SoC)*, Oct. 2023, pp. 1–6, doi: 10.1109/VLSI-SoC57769.2023.10321885.

BIOGRAPHIES OF AUTHOR

Trang Hoang was born in Nha Trang city, Vietnam. He received the Bachelor of Engineering, and Master of Science degree in Electronics-Telecommunication E-ngineering from HCMUT in 2002 and 2004, respectively. He received the Ph.D. degree in Micro-electronics MEMS from CEA-LETI and University Joseph Fourier, France, in 2009. From 2009–2010, he did the postdoctorate research in Orange Lab-France Telecom. Since 2010, he is lecturer at Faculty of Electricals–Electronics En gineering, HCMUT. His field of research interest is in the domain of ASIC-FPGA implementation, IC architecture, micro-fabrication, wireless communication, quantumn computing, optimization in analog IC design. He can be contacted at email: hoangtrang@hcmut.edu.vn.

Tu Dinh Anh Duong received the B.S. degree in Electronics and Telecommunications Engineering from Ho Chi Minh City University of Technology, VNU-HCM, Vietnam (2022). Tu does research in hardware implementation of applied cryptography. His current research focuses on designing efficient hardware for post-quantum cryptography schemes, especially CRYSTALS-Kyber. He can be contacted at email: tu.duonghk@gmail.com.

Thinh Quang Do was born in Lam Dong province, Vietnam. He received a Bachelor of Engineering, and a Master of Engineering degree in Electronics Engineering in 2019 and 2023, respectively. During 2021-2023, he worked as a researcher and teaching assistant at Ho Chi Minh City University of Technology, Vietnam National University. His field of research includes IC design, machine learning, encryption/decryption, quantum algorithms, and computer arithmetic. He can be contacted at email: dqtblldvntd@gmail.com.