

Design of polynomial NTT and INTT accelerator for Post-Quantum Cryptography CRYSTALS-Kyber

Hung Nguyen¹, Linh Tran¹

¹ Department of Electronics, Ho Chi Minh University of Technology, VNU-HCM, Vietnam

Article Info

Article history:

Keywords:

Cryptography
CRYSTALS-Kyber
FPGA
Hardware design
Post-quantum cryptography

ABSTRACT

NIST post-quantum cryptography standardization round 3 announced CRYSTALS-Kyber as one of the finalists. As a lattice-based cryptography scheme, CRYSTALS-Kyber relies heavily on polynomial multiplication efficiency. This paper presents a high-speed and pipelined hardware polynomial multiplier design based on number theoretic transform (NTT). Our work centers around designing and optimizing the NTT accelerator architecture for accelerating hardware implementation of CRYSTALS-Kyber. The work includes modifying the modular arithmetic modules and butterfly units with an efficient low complexity algorithm. As a result, our design achieved 237 MHz Fmax synthesized on Intel FPGA Cyclone V with Quartus. Resources utilization through combinational logic path rebalances allowed us to efficiently pipelining between hardware modules.

This is an open access article under the [CC BY-SA](#) license.



Corresponding Author:

Linh Tran

Department of Electronics, Ho Chi Minh University of Technology, VNU-HCM
268 Ly Thuong Kiet St., Ward 14, District 10, Ho Chi Minh City, Vietnam

Email: linhtran@hcmut.edu.vn

1. INTRODUCTION

The National Institute of Standards and Technology (NIST) announced the Post-Quantum Cryptography Standardization process in 2016 [1]. The goal of the standardization is to establish a new cryptographic system for both classical computers and quantum computers in the future. Research and development progress in Post-Quantum Cryptography has improved significantly through recent years. In 2020, third-round candidates were announced, with 4 out of 5 finalists for Public-key Encryption and Key-establishment Algorithms are lattice-based cryptography schemes. Lattice-based cryptography, together with the Learning with Errors problem (LWE), is proven to be safe to use under worst-case scenarios and offers a good trade-off between efficiency and security. CRYSTALS-Kyber is one of the five finalists in round 3.

CRYSTALS-Kyber is a key-encapsulation mechanism (KEM) based on the Module-LWE problem [2]. It also has a digital signature sibling, called CRYSTALS-Dilithium [3]. Kyber requires serious computational effort, primarily as the multiplication of polynomials over a constant-size polynomial ring. This scheme over "module lattices" offers a good trade-off between efficiency and security. However, the key generation, encryption, and decryption process could account for a large proportion of microprocessors' computing power and clock cycles. Kyber utilizes a high-speed multiplication technique called Number-Theoretic Transform (NTT) and chooses parameters to support this technique. In order to efficiently implement Kyber, optimizing NTT and Inverse NTT (INTT) is crucial.

Many researchers implement lattice-based cryptographical schemes on software as a proof of concept and utility. However, the public implementations are more likely to be processed on co-hardware and software or full hardware solutions. Many PQC's candidates and projects have their research design on an FPGA, ASIC, or co-processor for ARM, RISC V. We believe FPGA's reconfigurable hardware is a reliable platform to research cryptography algorithms implementation.

In previous researches, one of the earliest full hardware implementations of PQC [3] is for Round5. The design includes a hardware design of Keccak, AES-GCM, and the Round5 algorithm. The simplicity from Round5 modular advantages is used efficiently on hardware. In [4], a similar full hardware design of CRYSTALS-Kyber is presented, which speed-up the performance 129 times compared to Cortex-M4 processor implementation [5]. In [6], Jati et al. show the first side-channel attack-protected design of hardware CRYSTALS-Kyber with impressive performance. Zhao et al. [7] analyze the software code to optimize their design, yielding positive results. Other implementations of Kyber vary from hardware and software processor hybrids [8] [9] [10] to pure hardware [5] [6] [11] [12]. Recent researches on CRYSTALS-Kyber hardware implementation mostly optimize its polynomial multiplication process, which is our research's purpose. [13] present a novel modular technique called K2-RED for their NTT structure, which improves modular performance. A different modular technique is used in [14], [15] modified with pre-computed constants. Zhang et al. [16] present a ping-pong memory access scheme to access the RAM efficiently. Poppelmann et al. [17] introduce a master-slave structure for multiple polynomial multipliers. Our proposed design improves in speed and resource efficiency comparing to [5], [14], [15], [17].

In this paper, we present a hardware polynomial NTT and INTT accelerator for the CRYSTALS-Kyber PQC scheme. Our proposed method includes a modified NTT/Invert NTT butterfly unit for lowering the complexity of NTT; a tweaked version of Kyber modular reduction called Exact-KRED; an NTT architecture utilizing the modified 2x2 butterfly unit configuration. Our design is simulated with Intel ModelSim and synthesized with Intel Quartus for FPGA Intel Cyclone V 5CSXFC6D6F31C6.

We organize the rest of the paper as follows. Section 2 includes the preliminaries and our proposed design. In the preliminaries, we provide notations for CRYSTALS-Kyber and Number-Theoretic Transform (NTT). For NTT, we also explain previous works and present our algorithms with modifications. For our proposed design, we present the modular unit, the butterfly unit, the hardware architecture of the NTT accelerator, and the structure NTT algorithm. Section 3 is the result and discussion in which we compare our design result to other similar researches. We conclude the paper in section V.

2. RESEARCH METHOD

2.1. Preliminaries

In this section, we briefly describe the CRYSTALS-Kyber scheme and related mathematical information. After that, we present the NTT/INTT algorithm used for the architecture, with our adjustment and modification.

2.1.1 CRYSTALS-Kyber scheme

Here is our brief description of CRYSTALS-Kyber and the reason behind our research with NTT for Kyber cryptography computation. CRYSTALS-Kyber is a cryptography scheme based on the module-lattice learning-with-errors problem (MLWE [18]). The detail of Kyber can be found in its updated specification [19]. Kyber has two parts: a chosen-plaintext attack (CPA) secure public-key encryption (PKE) or CPAPKE, a chosen-ciphertext attack (CCA) secure key encapsulation mechanism (KEM) or CCAKEM. The CPAPKE is included inside CCAKEM as a mandatory step for ciphertext and key generation. While CPAPKE has three different procedures (key generation, encrypt and decrypt), all three steps require heavy polynomial multiplication. A very efficient way to deal with polynomial multiplications is Number Theoretic Transform (NTT). For that reason, NTT is included in the definition of Kyber, and the parameters of Kyber versions are tweaked to be calculated efficiently with NTT. Table 1 shows the parameters of each Kyber version.

Table 1. Parameters for each Kyber version

Version	n	k	q	η_1	η_2	(d_u, d_v)	δ
Kyber512	256	2	3329	3	2	(10,4)	2^{-139}
Kyber768	256	3	3329	2	2	(10,4)	2^{-164}
Kyber1024	256	4	3329	2	2	(11,5)	2^{-174}

Kyber's prime q was chosen for efficient NTT calculation, n bit length, k as a scaling factor for security level. The remaining parameters are chosen for balancing security, failure probability, and efficiency. For notation, we use n, q for later parts of the paper as we discuss the method to accelerate NTT.

2.1.2 Kyber number-theoretic transform (NTT)

As previously mentioned, the essential factor of Kyber multiplication is NTT. For notation and background, [19] describe the detail of Kyber multiplication and NTT algorithms. For brief discussion, NTT or the Fast Fourier Transform version reduces the complexity of polynomial multiplication from $O(N^2)$ to possibly, $O(N \log N)$. The multiplication result polynomial $h = f \cdot g$ could be calculated as function (1):

$$h = NTT^{-1}(NTT(f) \cdot NTT(g)) \quad (1)$$

We denote ω_n as the primitive n -th root of unity and a $\gamma = \sqrt{\omega_n}$. Because Kyber prime q satisfies $q \equiv 1 \pmod{2n}$, the version of the NTT algorithm we use is negative-wrap-convolution (NWC). We need to apply the pre-processing and post-processing to input and output polynomials coefficients, multiplying them by γ and γ^{-1} , respectively, as depicted in figure 1.

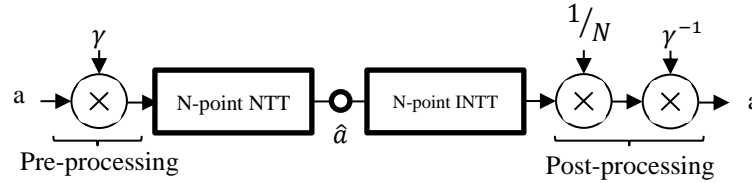


Figure 1. NWC NTT and INTT with pre-processing and post-processing

In the Kyber case, with br_7 being a bit-reversed function, \hat{h} as $NTT(h)$, polynomial multiplication in Kyber (as $\hat{f} * \hat{g} = \hat{h}$) has 128 products in function (2) [19].

$$\hat{h}_{2i} + \hat{h}_{2i+1}X = (\hat{f}_{2i}\hat{g}_{2i} + \hat{f}_{2i+1}\hat{g}_{2i+1}\omega_n^{2br_7(i)+1}) + (\hat{f}_{2i}\hat{g}_{2i+1} + \hat{f}_{2i+1}\hat{g}_{2i})X \quad (2)$$

In detail NTT calculation, we make use of the butterflies of Fast Fourier Transform (FFT) [20] [21], where NTT is implemented with Cooley-Tukey (CT) butterfly, and INTT is implemented with Gentleman-Sande (GS) butterfly. CT and GS butterfly units are efficient methods to design NTT for programmable hardware. Figure 2 depicts the general diagram for these butterfly units.

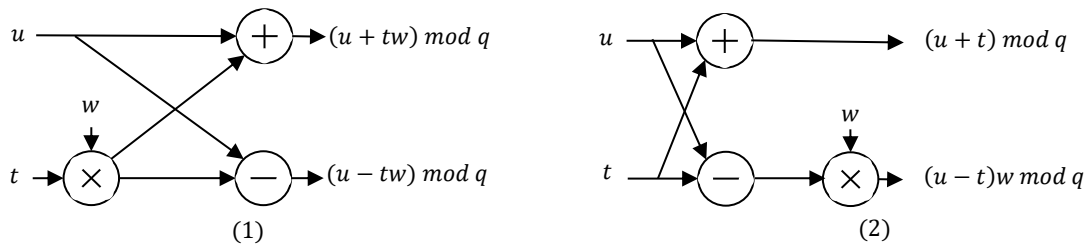


Figure 2. The diagram of Number Theoretic Transform butterfly units. (1) Cooley - Tukey Butterfly. (2) Gentleman - Sande Butterfly

Previously notable optimization for the NTT algorithm is the low-complexity NTT/INTT algorithm from [22]. The complexity of NTT operation originally is $O(N/2 \log N + N)$, with $N/2 \log N$ as the complexity of NTT and N for pre-processing. Using γ in the place of ω_n reduces the pre-processing step and complexity to $O(N/2 \log N)$. Similar to NTT, the original complexity of INTT is $O(N/2 \log N + 2N)$. The butterfly units' results are modularly multiplied by $1/2$ to eliminate the earlier post-processing step. We also need to change the twiddle factor to γ^{-1} , which removes the later step of post-processing. The complexity is finally reduced to $O(N/2 \log N)$.

In algorithms 1 and 2, we present the NTT and INTT operations, respectively. These are the base version we use to construct our hardware design. It should be mentioned that we applied minor modifications to fit our design, in reference to [5] [19] [22] [23]. Algorithm 1 shows the NTT operation using Cooley - Tukey butterfly with the low-complexity modification. The function bit-reverse is the bit-wise reversal function for the coefficients of the polynomial. Algorithm 2 shows the INTT operation using Gentleman - Sande butterfly.

Algorithm 1. Low complexity NTT operation with Cooley – Tuckey butterfly

Input: polynomial $a(a_0, a_1, \dots, a_{n-1})$ as $a(x) \in \mathbb{Z}_q[X]/(X_n + 1)$, $\omega_n \in \mathbb{Z}_q$ is the n-th primitive root of unity, $n = 2^l$ and $\gamma_{2n} = \sqrt{\omega_n}$

Output: $\hat{a} = NTT(a)$

```

1:  $\hat{a} = \text{bit-reverse}(a)$ 
2: for ( $i = 1; i < l; i++$ ) do
3:    $m = 2^i$ 
4:    $\omega_m \leftarrow \gamma_{2n}^{n/m}$ 
5:   for ( $j = 0; j < n; j = j + m$ ) do
6:      $\omega \leftarrow 1$ 
7:     for ( $k = 0; k < \frac{m}{2}; k++$ ) do
8:        $u \leftarrow \hat{a}[k + j]$ 
9:        $t \leftarrow \omega \cdot \hat{a}[k + j + m/2] \bmod q$ 
10:       $\hat{a}[k + j] = u + t \bmod q$ 
11:       $\hat{a}[k + j + m/2] = u - t \bmod q$ 
12:       $\omega \leftarrow \omega \cdot \omega_m \bmod q$ 
13:    end for
14:  end for
15: end for
16: return  $\hat{a}$ 

```

Algorithm 2. Low complexity INTT operation with Gentleman – Sande butterfly

Input: polynomial $\hat{a}(a_0, a_1, \dots, a_{n-1})$ as $\hat{a}(x) \in \mathbb{Z}_q[X]/(X_n + 1)$, $\omega_n \in \mathbb{Z}_q$ is the n-th primitive root of unity, $n = 2^l$ and $\gamma_{2n} = \sqrt{\omega_n}$

Output: $a = INTT(\hat{a})$

```

1: for ( $i = 1; i < l; i++$ ) do
2:    $m = 2^{l-i}$ 
3:    $\omega_m \leftarrow \gamma_{2n}^{-n/m}$ 
4:   for ( $j = 0; j < n; j = j + m$ ) do
5:      $\omega \leftarrow 1$ 
6:     for ( $k = 0; k < \frac{m}{2}; k++$ ) do
7:        $u \leftarrow \hat{a}[k + j]$ 
8:        $t \leftarrow \hat{a}[k + j + m/2] \bmod q$ 
9:        $\hat{a}[k + j] = \frac{u+t}{2} \bmod q$ 
10:       $\hat{a}[k + j + m/2] = \frac{u-t}{2} \cdot \omega \bmod q$ 
11:       $\omega \leftarrow \omega \cdot \omega_m \bmod q$ 
12:    end for
13:  end for
14: end for
15:  $a = \text{bit-reverse}(\hat{a})$ 
16: return  $a$ 

```

2.2. The proposed design**2.2.1. Modular reduction**

We present an improved version from KRED, K-RED-2x [20], and K2-RED [5] modular design with [24] mathematical background. KRED is similar to Montgomery modular reduction but designed to work for a special form of modulus depicted in the following equation:

$$\text{modulus } q = k \cdot 2^m + 1 \quad (3)$$

With Kyber modulus prime $q=3329$, $k = 13$ and $m = 8$. Using KRED, K-RED-2x on an input number C gives the result in $k \cdot C \bmod q$. Using K2-RED produces the result $k^2 \cdot C \bmod q$, which is explained in [5] to be more memory efficient and suitable for 12-bit modulus q . The modular unit is implemented after the multiplication step in the butterfly unit. To complement for k^2 multiple parts in the result, we pre-process ω to $\omega' = (k^{-2} \cdot \omega) \bmod q$. The methods of KRED and K2-RED are both implied to have cases where they overflow and do not give the exact match to the correct result.

By simulating and testing on Intel ModelSim, we mapped out the cases where the original K2-RED failed to give the correct result. We modify the sequence and give the algorithm an additional step to eliminate the resulting failure due to negative binaries addition. The resources increment is neglectable, while we could maintain the result accuracy. Algorithm 3 shows the Exact-KRED algorithm for Kyber. Zero-extend and signed-extend is the bit-wise extension function of the corresponding methods. The extension length is noted after the function.

Algorithm 3. Exact-KRED for Kyber NTT/INTT accelerator

Input: q modulus = 3329, binary number $C = (C_{23}, \dots, C_1, C_0)$, $k = 13$, $m = 8$

Output: $S = (k^2 * C) \bmod q$

1: $Cl \leftarrow \text{zero-extend}(C_7, \dots, C_1, C_0)$ to 16-bit

2: $Ch \leftarrow (C_{23}, \dots, C_9, C_8)$

3: $C' = ((Cl \ll 3) - Ch) + (Cl \ll 2 + Cl)$

4: $Cl' \leftarrow \text{zero-extend}(C'_7, \dots, C'_1, C'_0)$ to 12-bit

5: $Ch' \leftarrow \text{signed-extend}(C'_{23}, \dots, C'_9, C'_8)$ to 12-bit

6: $C'' = ((Cl' \ll 3) - Ch') + (Cl' \ll 2 + Cl')$

7: **If** $C'' \geq q$

8: $S \leftarrow C'' + q$

9: **Else**

10: $S \leftarrow C''$

11: **return** S

Figure 3 presents the hardware structure for Exact-KRED. By carefully pipeline the steps, we can keep the complex combinational logic down to 2 logic levels, which improves the overall speed of the design.

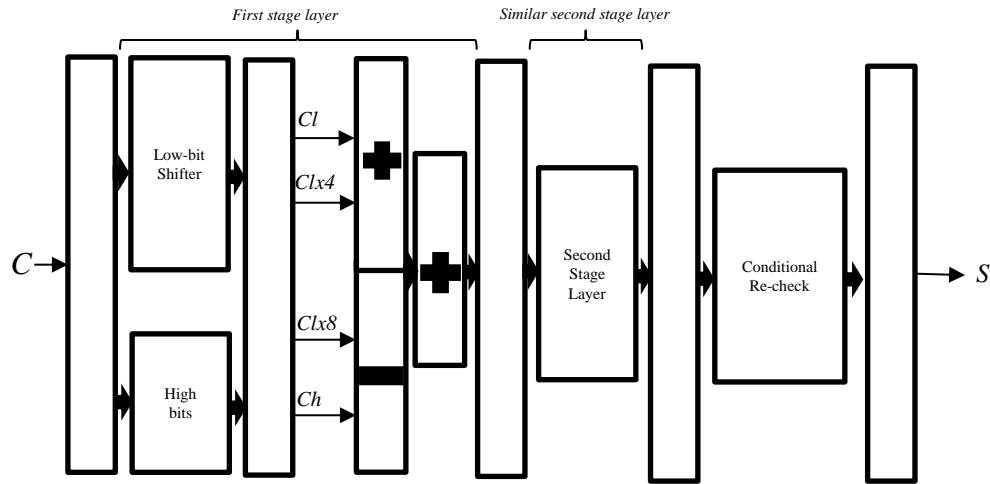


Figure 3. Exact-KRED modular reduction hardware structure

2.2.2. Polynomial NTT and INTT accelerator for Kyber

We describe a hardware design of a polynomial NTT and INTT accelerator for Kyber in two parts. First, we present the hardware structure of the modified butterfly unit adjusted to algorithms 1 and 2. Then we show the overall architecture of the accelerator with a 2x2 butterfly units' configuration and improved modular reduction hardware.

The 2x2 set of butterfly units is the main component to calculate NTT/INTT. From figure 2, we design the hardware butterfly unit suitable for both CT and GS operations. The process would cost two modular arithmetic adders, one DSP multiplier, and two Exact-KRED modular reduction units. The constant divider for modified GS unit resource consumption is trivial. The divider is modified into the Exact-KRED structure. Figure 4 gives the detailed structure of the unit. The DSP multiplier and Exact-KRED unit have 7 cycles of delay. The path delay blocks pipeline the data and are adjusted according to the total delay. The total delay for each CT/GS operation on the butterfly unit is 9 cycles, with pipelined inputs and output. Sel signal is used to select the mode of operation. Modular addition and subtraction with the two modular adders do not significantly affect speed performance due to the small 12-bit prime q .

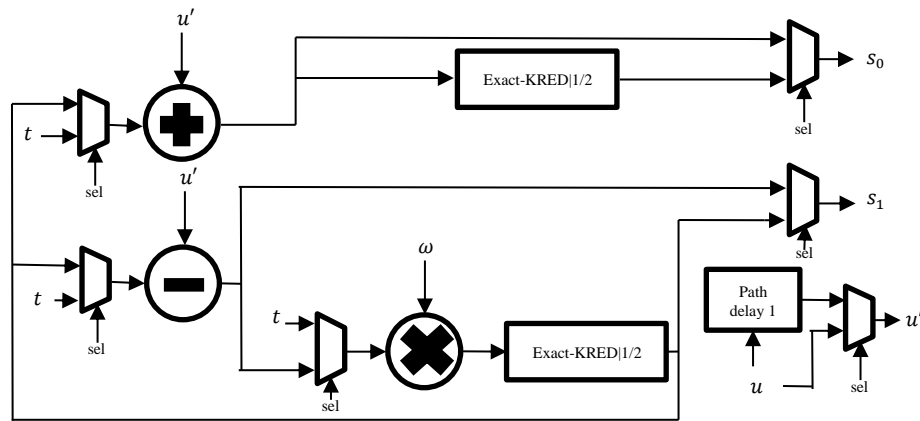


Figure 4. CT/GS uniform and modified butterfly unit hardware structure

For the overall architecture, our hardware NTT accelerator utilizes two sets of memory. One RAM for the NTT/INTT operands and three ROMs. The RAM uses Intel Cyclone V M10K in simple dual-port mode, allowing simultaneous access to two different memory addresses [25]. With dual RAM reading, our main state machine can feed two butterfly units (BU) at one operation. The computed data is feedback through the parameterized channel system back to the RAM. In every operation, two 24-bit data from each RAM is input into the butterfly unit set. The ROMs also feed the butterfly units with the corresponding factor for each operation. The value ω' is pre-computed and stored in synthesis. After calculation, the write FIFO buffers the result back to RAM. Each NTT operation cycle costs 24 clocks until the data is written back to the RAM. Figure 5 shows the general structure of the NTT/INTT accelerator for Kyber.

We adopt the sequence of NTT/INTT from [5] and [24] and utilize it with the low complexity modification in algorithms 1 and 2. We show the proposed NTT operation with Cooley – Tuckey butterfly for Kyber in algorithm 4. The butterfly units are used in Cooley – Tuckey mode (CT_{mode}). We pre-computed the value of γ_{2n} with k^{-2} to complement for Exact-KRED in ROM. Similarly, the value of γ_{2n}^{-1} with k^{-2} is pre-computed and stored in ROM for the INTT operation.

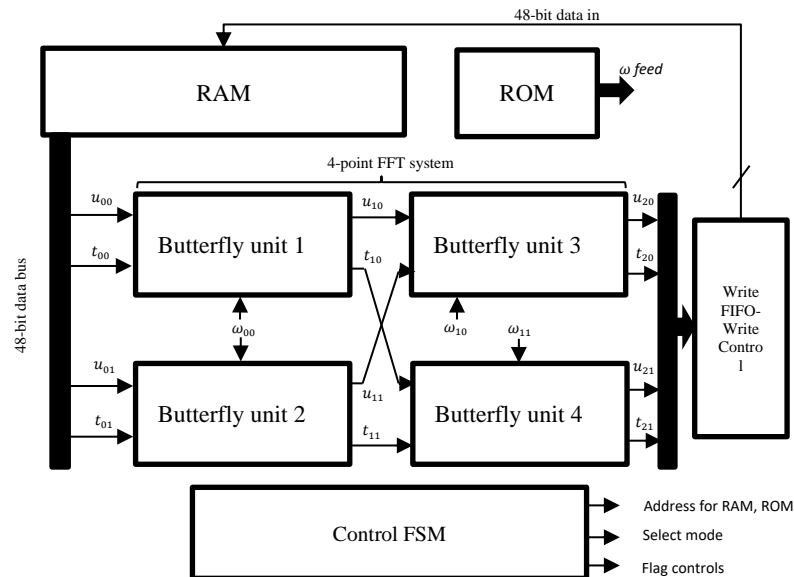


Figure 5. The hardware structure of NTT-based Polynomial Multiplier for Kyber

Algorithm 4 Proposed NTT operation with Cooley – Tuckey butterfly for Kyber

Input: polynomial $a(a_0, a_1, \dots, a_{n-1})$ as $a(x) \in \mathbb{Z}_q[X]/(X_n + 1)$, $\omega_n \in \mathbb{Z}_q$ is the n-th primitive root of unity, $n = 2^l$ and $\gamma_{2n} = \sqrt{\omega_n}$, $c = n/2$

Output: $a(x) = NTT(a)$

```

2: for (m = 1; m < n; m = 4m) do
4:   c = c/4
5:   for (j = 0; j < m; j++) do
7:     for (k = 4i · c; k < 4i · c + c; k++) do
8:        $u_{00} \leftarrow a[k], t_{00} \leftarrow a[k + c]$ 
9:        $u_{01} \leftarrow a[k + 2c], t_{01} \leftarrow a[k + 3c]$ 
X:        $\omega_{00} \leftarrow k^{-2} \cdot \gamma[m + i]$  (pre-computed)
X:        $(u_{10}, t_{10}) \leftarrow BU1(u_{00}, t_{00}, \omega_{00}, CT_{mode})$ 
X:        $(u_{11}, t_{11}) \leftarrow BU2(u_{01}, t_{01}, \omega_{00}, CT_{mode})$ 
X:        $\omega_{10} \leftarrow k^{-2} \cdot \gamma[2(m + i)]$  (pre-computed)
X:        $\omega_{11} \leftarrow k^{-2} \cdot \gamma[2(m + i) + 1]$  (pre-computed)
X:        $(u_{20}, t_{20}) \leftarrow BU3(u_{10}, u_{11}, \omega_{00}, CT_{mode})$ 
X:        $(u_{21}, t_{21}) \leftarrow BU4(t_{10}, t_{11}, \omega_{00}, CT_{mode})$ 
10:       $a[k] \leftarrow u_{20}, a[k + c] \leftarrow t_{20}$ 
10:       $a[k + 2c] \leftarrow u_{21}, a[k + 3c] \leftarrow t_{21}$ 
13:    end for
14:  end for
15: end for
16: return a(x)

```

3. RESULTS AND DISCUSSION

In this section, we share and discuss the result of our proposed design. The hardware structure and its module is synthesized with Intel Quartus Prime 18.1 Lite. The selected chip is Intel Cyclone V 5CSXFC6D6F31C6 which is readily available at our lab.

3.1. Exact-KRED modular reduction

Table 2 Exact-KRED in comparison with other modular reduction algorithms

Algorithm	Area		Speed [MHz]
	LUTs	FF	
KRED [26]	80	47	N/A
K2-RED [5]	54	30	234
Montgomery [5]	391	382	N/A
Barret-Reduction ¹ [14]	236	N/A	154
Barret-Reduction ² [14]	195	N/A	212
Barret-Reduction ² [15]	309	132	136
Exact-KRED	56³	80	237

¹Result implemented on Spartan-6 FPGA

²Result implemented on Artix-7 FPGA

³ALMs as in logic utilization from Quartus report

As Exact-KRED is pipelined from input to output with extra condition cases, its resource consumption slightly increases from other modular reduction hardware implementations. We achieved a similar result with moderate improvements compared to other KRED without exact-result in terms of speed. Compared to Montgomery reduction and multiple implements of Barret Reduction, we achieved significant improvement in area and speed.

3.2. Polynomial NTT and INTT accelerator for Kyber

Table 3 Our proposed design compared with similar previous researches

Design	Area				Speed [MHz]	NTT Cycles	SxC [ns]
	LUTs	FFs	DSPs	Memory			
Fermat prime [17] ¹	1438	1123	1	2.5	209	4774	22437
Kyber512 [15] ²	442	237	1	1.5	136	2055	15110
TW-1 BTFs [14] ²	2543	792	4	9	182	232	1274
HS-NTT [5] ²	798	715	4	2	234	1591	6799
Proposed design	1232³	1436	4	2	237	1538	6658

¹Similar number of co-efficient, different prime, implemented on Spartan-6

²Implemented on Xilinx Artix-7

³Result in ALMs as in logic utilization from Quartus report

Compared with similar implementations on Xilinx FPGAs, our implementation on Intel Cyclone V cost a similar or, in some cases, lower resource consumption than previous research. Because our work uses Intel Cyclone V, the ALMs result also includes LUTs and FFs usage, which is hard to compare with many other implements, mostly on Xilinx FPGAs. We use a term called SxC to calculate the estimated amount of time in ns needed for the implementation to complete an NTT operation. Our cycle count is calculated from the total latency of the proposed design. In [17], the implementation relies on multiple polynomial multipliers to work in parallel. In single-core, we achieved better improvement in cycles. We can achieve a better speed than [15], with only half the time for NTT. TW-4 BTFs SxC is faster than our design, with a cost of four times the memory. Compared to HS-NTT with original KRED gives close result. We intended our design to be compact and fast and prove that implementing it on an economy FPGA such as Intel Cyclone V could give similar results compared to other higher-end FPGAs.

For this proposed structure, we hope to achieve a better result in a dual NTT accelerator configuration for full hardware Kyber implementation or a polynomial multiplication design with a shared ROM usage for both accelerators. The synchronized ROM reading would further reduce memory block consumption. Taking advantage of Kyber's small prime q and 12-bit coefficients' bit-length, resources for the arithmetic operation could be optimized without compromising the security factor of the cryptography scheme.

4. CONCLUSION

We proposed the hardware design of a polynomial NTT and INTT accelerator for Kyber core dedicated to the CRYSTALS-Kyber family of post-quantum Cryptography. The design includes improved modular reduction hardware, uniform CT/GS butterfly units modified with low-complexity configuration, and an NTT accelerator architecture. The design is deeply pipelined, and we manually rebalanced the logic level to optimize for the performance.



In future studies, we would further optimize the sequence of memory access needed to fulfill the NTT and INTT operation. For example, we could analyze the optimal number of butterfly units in order to achieve better memory usage. On the other hand, more research could be put into utilizing the advantages of the NTT pre-computed twiddle factor to simplify the algorithm sequence.

REFERENCES

- [1] NIST, "Post-Quantum Cryptography Standardization", [csrc.nist.gov](https://csrc.nist.gov/Projects/post-quantum-cryptography), <https://csrc.nist.gov/Projects/post-quantum-cryptography>.
- [2] Bos, Joppe, Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé, "CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM." In 2018 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 353-367. IEEE, 2018, doi: 10.1109/EuroSP.2018.00032.
- [3] Andrzejczak, Michal, Farnoud Farahmand, and Kris Gaj, "Full Hardware Implementation of the Post-Quantum Public-Key Cryptography Scheme Round5." In 2019 International Conference on ReConfigurable Computing and FPGAs (ReConFig), pp. 1-2. IEEE, 2019, doi: 10.1109/ReConFig48160.2019.8994765.
- [4] Huang, Yiming, Miaoqing Huang, Zhongkui Lei, and Jiaxuan Wu, "A pure hardware implementation of crystals-kyber PQC algorithm through resource reuse." IEICE Electronics Express (2020) ID: 17-20200234, doi: 10.1587/elex.17.20200234.
- [5] Botros, Leon, Matthias J. Kannwischer, and Peter Schwabe, "Memory-efficient high-speed implementation of Kyber on Cortex-M4." In International Conference on Cryptology in Africa, pp. 209-228. Springer, Cham, 2019, doi: 10.1007/978-3-030-23696-0_11.
- [6] Jati, Arpan, Naina Gupta, Anupam Chattopadhyay, and Somitra Kumar Sanadhya, "A Configurable Crystals-Kyber Hardware Implementation with Side-Channel Protection." Cryptology ePrint Archive (2021).
- [7] Zhao, Yixuan, Zhiteng Chao, Jing Ye, Wen Wang, Yuan Cao, Shuai Chen, Xiaowei Li, and Huawei Li, "Optimization Space Exploration of Hardware Design for CRYSTALS-KYBER." In 2020 IEEE 29th Asian Test Symposium (ATS), pp. 1-6. IEEE, 2020. doi: 10.1109/ATS49688.2020.9301498.

- [8] Albrecht, Martin R., Christian Hanser, Andrea Hoeller, Thomas Pöppelmann, Fernando Virdia, and Andreas Wallner, "Implementing RLWE-based schemes using an RSA co-processor." *Cryptology ePrint Archive* (2018).
- [9] Sanal, Pakize, Emrah Karagoz, Hwajeong Seo, Reza Azarderakhsh, and Mehran Mozaffari-Kermani. "Kyber on ARM64: Compact Implementations of Kyber on 64-bit ARM Cortex-A Processors." *Cryptology ePrint Archive* (2021).
- [10] Seo, Hwa-jeong, Hyeok-dong Kwon, Kyoung-bae Jang, and Hyunjun Kim, "Optimized implementation of scalable multi-precision multiplication method on RISC-V processor for high-speed computation of post-quantum cryptography." *Journal of the Korea Institute of Information Security & Cryptology* 31, no. 3 (2021): 473-480, doi: 10.13089/JKIISC.2021.31.3.473.
- [11] Xing, Yufei, and Shuguo Li, "A compact hardware implementation of CCA-secure key exchange mechanism CRYSTALS-KYBER on FPGA." *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2021): 328-356, doi: 10.46586/tches.v2021.i2.328-356.
- [12] Guo, Wenbo, Shuguo Li, and Liang Kong, "An Efficient Implementation of KYBER." *IEEE Transactions on Circuits and Systems II: Express Briefs* (2021), doi: 10.1109/TCSII.2021.3103184.
- [13] Bisheh-Niasar, Mojtaba, Reza Azarderakhsh, and Mehran Mozaffari-Kermani, "High-Speed NTT-based Polynomial Multiplication Accelerator for CRYSTALS-Kyber Post-Quantum Cryptography." *Cryptol. ePrint Arch., Tech. Rep* 563 (2021): 2021.
- [14] Yarman, Ferhat, Ahmet Can Mert, Erdiñç Öztürk, and Erkay Savaş, "A hardware accelerator for polynomial multiplication operation of CRYSTALS-KYBER PQC scheme." In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1020-1025. IEEE, 2021, doi: 10.23919/DATE51398.2021.9474139.
- [15] Chen, Zhaohui, Yuan Ma, Tianyu Chen, Jingqiang Lin, and Jiwu Jing, "Towards efficient Kyber on FPGAs: A processor for vector of polynomials." In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 247-252. IEEE, 2020, doi: 10.1109/ASP-DAC47756.2020.9045459.
- [16] Zhang, Cong, Dongsheng Liu, Xingjie Liu, Xuecheng Zou, Guangda Niu, Bo Liu, and Quming Jiang, "Towards Efficient Hardware Implementation of NTT for Kyber on FPGAs." In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1-5. IEEE, 2021, doi: 10.1109/ISCAS51556.2021.9401170.
- [17] Pöppelmann, Thomas, and Tim Güneysu, "Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware." In *International conference on cryptology and information security in Latin America*, pp. 139-158. Springer, Berlin, Heidelberg, 2012, doi: 10.1007/978-3-642-33481-8_8.
- [18] Langlois, Adeline, and Damien Stehlé, "Worst-case to average-case reductions for module lattices." *Designs, Codes and Cryptography* 75, no. 3 (2015): 565-599. Avanzi, Roberto, et al. "CRYSTALS-Kyber algorithm specifications and supporting documentation." *NIST PQC Round 2.4* (2017), doi: 10.1007/s10623-014-9938-4.
- [19] Avanzi, Roberto, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé, "CRYSTALS-Kyber algorithm specifications and supporting documentation." *NIST PQC Round 2*, no. 4 (2017).
- [20] Bisheh-Niasar, Mojtaba, Reza Azarderakhsh, and Mehran Mozaffari-Kermani, "A Monolithic Hardware Implementation of Kyber: Comparing Apples to Apples in PQC Candidates." In *International Conference on Cryptology and Information Security in Latin America*, pp. 108-126. Springer, Cham, 2021, doi: 10.1007/978-3-030-88238-9_6.
- [21] Becker, Hanno, Vincent Hwang, Matthias J. Kannwischer, Bo-Yin Yang, and Shang-Yi Yang, "Neon NTT: Faster Dilithium, Kyber, and Saber on Cortex-A72 and Apple M1." *Cryptology ePrint Archive* (2021).
- [22] Zhang, Neng, Bohan Yang, Chen Chen, Shouyi Yin, Shaojun Wei, and Leibo Liu, "Highly efficient architecture of NewHope-NIST on FPGA using low-complexity NTT/INTT." *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2020): 49-72, doi: 10.13154/tches.v2020.i2.49-72.
- [23] Pöppelmann, Thomas, and Tim Güneysu, "Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware." In *International conference on cryptology and information security in Latin America*, pp. 139-158. Springer, Berlin, Heidelberg, 2012, doi: 10.1007/978-3-642-33481-8_8.
- [24] Longa, Patrick, and Michael Naehrig, "Speeding up the number theoretic transform for faster ideal lattice-based cryptography." In *International Conference on Cryptology and Network Security*, pp. 124-139. Springer, Cham, 2016, doi: 10.1007/978-3-319-48965-0_8.
- [25] Intel, "Cyclone V Device Datasheet," CV-51002 datasheet, Nov. 2019.
- [26] Nguyen, Duc Tri, Viet B. Dang, and Kris Gaj. "High-Level Synthesis in Implementing and Benchmarking Number Theoretic Transform in Lattice-Based Post-Quantum Cryptography Using Software/Hardware Codesign." In *ARC*, pp. 247-257. 2020. doi: 10.1007/978-3-030-44534-8_19.

BIOGRAPHIES OF AUTHORS

	<p>HUNG NGUYEN received the B.S. degree in Electronics and Telecommunications Engineering from Ho Chi Minh City University of Technology (HCMUT), Vietnam (2019). He is currently pursuing the MS. degree in electrical engineering at Ho Chi Minh City University of Technology (HCMUT), Vietnam.</p> <p>From 2015 to 2016, he was a Hardware Logic Designer at Arrive Technologies, Vietnam. His research interests include cryptography, AI, logic design, and hardware solution.</p>
	<p>LINH H. TRAN received the B.S. degree in Electrical and Computer Engineering from University of Illinois, Urbana – Champaign (2005), M.S. and PhD. in Computer Engineering from Portland State University (2006, 2015). Currently, he is working as a lecturer at Faculty of Electrical-Electronics Engineering, Ho Chi Minh City University of Technology – VNU HCM. His research interests include quantum/ reversible logic synthesis, computer architecture, hardware-software co-design, efficient algorithms, and hardware design targeting FPGAs and data analysis.</p>