Efficient Number Theoretic Transform Architecture for CRYSTALS-Kyber

Khalid Javeed, *Senior Member, IEEE*, and David Gregg

***Abstract*—The Number Theoretic Transform (NTT) is a central primitive to compute polynomial multiplication in a finite ring for both post-quantum cryptography (PQC) and fully homomorphic encryption (FHE) schemes. This brief presents a novel, efficient NTT hardware architecture suitable for CRYSTALS-Kyber, one of the NIST PQC standards. It is based on a new novel unified butterfly unit (UBU) developed by combining interleaved multiplication, radix-4, and resource-sharing strategies. This unit computes all butterfly operations for any generic prime modulus value and is re-configurable to any modulus length. In the proposed NTT architecture, multiple UBUs are deployed, demonstrating an area-time tradeoff. UBU and NTT architectures are synthesized and implemented over the Xilinix Artix-7 FPGA platform and results are shown for different performance evaluation metrics. The implementation results show our lightweight and high-speed designs achieve up to 5.6× and 7× improvements in resource consumption and efficiency, respectively. To the authors' knowledge, it is the first generic NTT architecture based on interleaved multiplication approaches.**

***Index Terms*—Number theoretic transform, FPGA, interleaved multiplication, post-quantum cryptography.**

I. Introduction

WITH the advent of quantum computers (QCs), current widely deployed public key schemes such as RSA [1] and elliptic curve cryptography [2] can no longer ensure their security. This is due to the computational power of QCs that can solve the underlying mathematical hard problems: integer factorization in RSA and discrete logarithm in ECC using Shor's algorithm [3]. This led to the National Institute of Standards and Technology (NIST) initiatives for standardizing various post-quantum cryptography (PQC) schemes. In this regard, CRYSTALS-Kyber [4], a lattice-based crypto scheme is popular for key exchange and encryption/decryption tasks. At the same time, Dilithium is standardized as the digital signature scheme [5]. On the other hand, fully homomorphic encryption (FHE) schemes enable operations over encrypted data and have a lot of applications in cloud computing security [6]. Lattice-based cryptography (LBC) is a popular tool for designing PQC and FHE schemes. Kyber security relies on computing module learning with error (module-LWE) problems [7]. The fundamental operation in module lattice is denoted as $As+e$, where A is a $k \times k$ polynomial matrix while s and e are k-dimensional polynomial vectors having their elements in the polynomial ring R_q. Note that Kyber security level can be adjusted by tuning different parameters.

Several researchers have proposed hardware accelerators to demonstrate the practical feasibility of PQC schemes. Polynomial multiplication over a ring is the central operation in almost all LBC schemes. The number theoretic transform (NTT) is a vital tool to accelerate polynomial multiplication and enable the practical deployment of LBC-based cryptosystems. It reduces the computational complexity of a polynomial multiplication from quadratic $\mathcal{O}(n^2)$ to quasi-linear $\mathcal{O}(n \log n)$.

The butterfly unit (BU) is the fundamental primitive in the design of NTT hardware architecture. It consists of one modular multiplication (MM), one modular addition (MA), one modular subtraction (MS), and two modular division by 2 (div-by-2) operations. The MM primitive is the most computationally-intensive part, and is critical to the overall performance of the NTT computation. The div-by-2 operations are required only in an inverse NTT transform (INTT = NTT^{-1}). Numerous hardware accelerators for NTT [8], [9], [10], [11], [12], [13], [14], [15], [16], [17] are available where the efforts have been mainly to optimize the MM primitive in the respective BU. In this regard, three approaches have been explored: Barret reduction [18], Montgomery reduction [19], and exploiting the special characteristics of a chosen prime modulus q. Montgomery reduction has seen limited deployment due to the need for domain conversion, whereas the Barret and special modulus-based reductions have been extensively used. In [8], [9], [10], NTT architectures for LBC are proposed using the Montgomery reduction while [11], [12] are based on Barret reduction and [15], [16], [20] proposed new designs by exploiting the special structure of selected modulus q [11]. Designs over special q can produce higher performance but are tied to the selected q and lack flexibility. Thus, the gain in speed is offset by the lack of flexibility. In this brief, we propose an efficient NTT architecture to support any generic q and evaluate its performance on the FPGA platform. The design uses LUTs making it more flexible to be ported to any other FPGA and even to standard ASIC cell technologies. Our main contributions in this brief are given below:

- We proposed an efficient NTT architecture that can support any generic modulus q based on a novel unified butterfly unit (UBU).
- In the UBU, we merge MM, MA, MS, and two div-by-2 operations using the interleaved multiplication (IM) [21].

Received 17 June 2024; revised 7 August 2024; accepted 17 September 2024. Date of publication 20 September 2024; date of current version 27 December 2024. This brief was recommended by Associate Editor S.-B. Ko. *(Corresponding author: Khalid Javeed.)*

Khalid Javeed is with the Department of Computer Engineering, University of Sharjah, Sharjah, UAE (e-mail: kjaveed@sharjah.ac.ae).

David Gregg is with the Lero-the Science Foundation Ireland Research Centre for Software, Trinity College Dublin, Dublin 2, D02 PN40 Ireland (e-mail: david.gregg@tcd.ie).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSII.2024.3465273.

Digital Object Identifier 10.1109/TCSII.2024.3465273

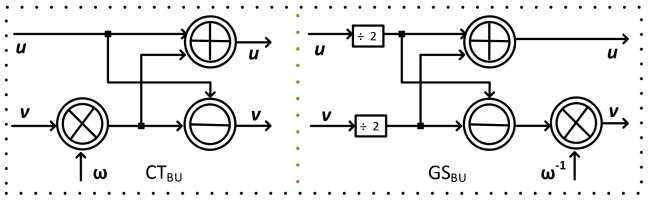

Fig. 1. CT_{BU} and GS_{BU} internal architectures.

We modify the IM algorithm to execute all BU operations required in NTT and INTT with an insignificant increase in the critical path delay and hardware resource consumption overhead. To reduce the critical path delay, we used a laddering approach to remove the data dependency between partial product generation and accumulation parts. Thus, these can be executed concurrently. To the best of the authors' knowledge, this is the first implementation of NTT using IM methods.

- We evaluated our UBU for common q sizes (12 bits) and the NTT/INTT architecture for the Kyber scheme ($n = 256$) on the Xilinx Artix-7 FPGA platform. The performance evaluation confirms that these deliver better efficiency results in comparison to the state-of-the-art.

This brief is structured as follows: Section II introduces the NTT. Section III presents our novel UBU algorithm and its mapping to the proposed hardware architecture within the overall NTT design. Section IV presents the results.

II. Background

The NTT is defined over a finite field $\mathbb{F}_q$ with integer roots whereas the Fast Fourier transform (FFT) works in a complex field $\mathbb{C}$ with complex roots. Let's say an integer polynomial p in an integer ring R_q is represented as $p(x) = Z_q[X]/(X^N+1)$, where reduction of the coefficients of this polynomial can be done by (X^N+1). The NTT and INTT of polynomial $p(x)$ are represented as $\hat{p}(x) \leftarrow \text{NTT}(p(x))$ and $p(x) \leftarrow \text{INTT}(\hat{p}(x))$, respectively. Similar to FFT computations, the forward NTT of a given polynomial can be performed using Cooley-Tukey BU (CT_{BU}), whereas, coefficient reversing operation can be avoided by using the Gentleman-Sande BU (GS_{BU}) for the INTT computation. The internal architectures of these BUs are shown in Fig. 1. It is evident from the figure that CT_{BU} and GS_{BU} require one MM, MA, and MS operations in different orders. Moreover, in addition to these, the GS_{BU} also requires two div-by-2 operations. MM is the core operation in both CT_{BU} and GS_{BU} computations and it can greatly influence the overall performance of the NTT/INTT primitive. Therefore its optimization is crucial in accelerating the LBC-based cryptosystems.

III. Proposed UBU Module

The proposed novel UBU module is based on IM [21] method. The IM method works in an iterative manner where partial products are generated, added, and reduced in each iteration. Numerous modifications [22], [23] have been proposed to design efficient MM modules for ECC. However, ECC works on large operands (256-512 bits) so straightforward adoption of these proposals for MM operation in CT_{BU} and GS_{BU} is not possible. Moreover, there are MA, MS, and div-by-2 operations required in these BU units.

Algorithm 1: Proposed Unified Butterfly Unit (UBU)

Input: $\mathcal{V}, \mathcal{U}, \omega$
Output: $\mathcal{V}^{i+1}, \mathcal{U}^{i+1}$
1 Initialization: $t = 0, t_1 = \mathcal{V}, t_2 = 2 \times \mathcal{V} \bmod q$
2 $\alpha = \begin{cases} \log_2 q + 2, & \log_2 q \text{ even} \\ \log_2 q + 1, & \log_2 q \text{ odd} \end{cases}$
// **OP-1:** MM of $\mathcal{V}$ and $\omega(\mathcal{V} \bigotimes \omega)$
// MM started
3 **for** $(i = 0; i \leq \alpha - 2; i \leftarrow i + 2)$ **do**
4 **switch** $(\omega_{(i+2:i)})$ **do**
5 **when** $000 \mid 111 \Longrightarrow d \leftarrow 0$
6 **when** $001 \mid 010 \mid 101 \mid 110 \Longrightarrow d \leftarrow t_1$
7 **else** $\Longrightarrow d \leftarrow t_2$
8 **end**
9 $t_1 = \beta \bigodot t_1 \ mod \ q$
10 $t_2 = \beta \bigodot t_2 \ mod \ q$
11 $t = t(\oplus|\ominus)d \ mod \ q$
12 **end**
// OP-1 (MM) completed
// **OP-2:** MA of t and U $(t \oplus \mathcal{U})$
13 $\mathcal{U}^{next} = t \oplus \mathcal{U}$
// **OP-3:**MS of t and U $(t \ominus \mathcal{U})$
14 $\mathcal{V}^{next} = t \ominus \mathcal{V}$
15 **return** $\mathcal{V}^{next}, \mathcal{U}^{next}$

We modified the IM algorithm so that it can perform all required BU operations. Our proposed UBU algorithm is given in Algorithm 1. Our modifications eliminate data dependencies in critical operations so that these can be executed concurrently with low hardware footprints. We represent MM, MA, MS operations with $\otimes$, $\oplus$, and $\ominus$, respectively, while $\odot$ represents a modular multiply-by-4 operation. It is worth mentioning that $\odot$ operation is required in each iteration of the modified algorithm in the partial product generation part. The given algorithm takes four inputs: two coefficients of the polynomial $\mathcal{V}, \mathcal{U}$, a twiddle factor ω, and a modulus q. It produces two new coefficients $\mathcal{V}^{next}, \mathcal{U}^{next}$ after completing the required operations. Three operations OP-1 ($\otimes$), OP-2 ($\oplus$), and OP-3 ($\oplus$) are specified in steps 3, 13, and 14, respectively. The specified order of these operations is for computing CT_{BU}, where the result of OP-1 ($\mathcal{V} \otimes \omega$) is $\oplus$ and $\ominus$ with $\mathcal{U}$ and $\mathcal{V}$, respectively. However, these operations can be sequenced to facilitate the execution of GS_{BU}. OP-1 is the most time-critical operation comprised of steps 1 to 11, where steps 3 to 11 run iteratively whereas steps 1 and 2 are required only once for variable initialization and to determine the bit length α of a modulus q. In most of the PQC schemes, $\log_2 q$ results in an even number so we need to append two zeros to the left of the MSB bit of a multiplier ω, i.e., $\alpha = \log_2 q + 2$.

In step 1, pre-computation of a modular multiply-by 2 of multiplicand $\mathcal{V}$ is performed in addition to loading of t_1 and t_2 with zero and $\mathcal{V}$, respectively. To execute OP-1, our algorithm starts scanning the multiplier (ω) from the least-significant-bit (LSB) and forms groups of three bits (step 3), where the MSB of the current group acts as an LSB of the subsequent group. Therefore, due to radix-4, each iteration processes two bits of the multiplier. Steps 9, 10, and 11 are the main execution steps in performing OP-1. These three steps

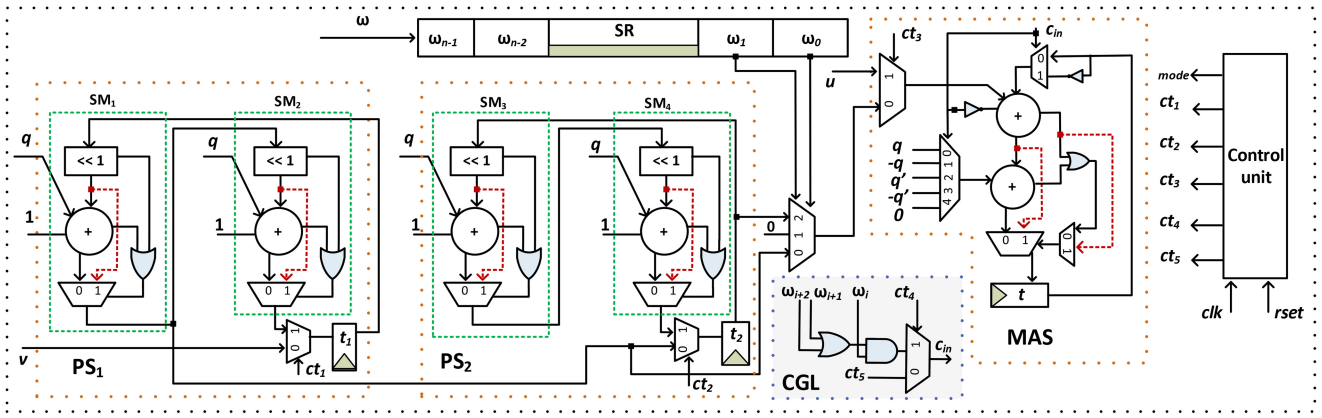

Fig. 2. Proposed Unified Butterfly Architecture.

have no data dependencies and can operate in parallel. The possible partial products t_1, t_2 are left-shifted and reduced in each iteration, where the amount of shift is represented by $\log_2 \beta$. Due to radix-4, we select $\beta = 4$ in the given algorithm. Step 11 is the $\oplus$ or $\ominus$ of a respective partial product from the accumulator t contents. Due to the processing of two bits of the multiplier in each iteration, the total number of iterations required to complete OP-1 is $\lceil \alpha/2 \rceil + 1$. Once the OP-1 is complete, the two remaining operations ($\oplus$ and $\ominus$) of BU can be performed. These are specified in steps 14 and 15. The div-by-2 operation required in INTT is merged in $\oplus$ and $\ominus$ operations. The internal working details of $\otimes$, $\odot$, $\oplus$, $\ominus$, and div-by-2 operations are given in the following section.

A. UBU Hardware Design

A compact hardware architecture to execute the proposed UBU method is presented in Fig. 2. The proposed UBU hardware architecture consists of several computational blocks: two partial product shifters PS_1 and PS_2, one modular addition subtraction (MAS), one shift register (SR), three data storage registers t, t_1, t_2, several multiplexers, carry generation logic (CGL), and a controller. PS_1, PS_2, and MAS are the main computational modules whereas multiplexers and the controller select appropriate inputs and generate required control signals, respectively. We reduce the critical path delay by running PS1, PS2, and MAS in parallel. In addition, to minimize the hardware footprints, the MAS unit is shared among different UBU operations. Details of these units are as follows:

1) PS_1 and PS_2: These units are responsible for executing steps 9 and 10 of the algorithm where possible partial products are multiplied by β and reduced by the given modulus q. Thus these modules are used only in the computation of OP-1 ($\otimes$). We adopted radix-4 so $\beta = 2^2$, hence partial products are two-bit left-shifted and reduced. As both steps are identical so PS_1 and PS_2 blocks are the same. Initially, registers t and SR are loaded with zero and ω, respectively. However, at the first clock cycle, we use PS_1 block to compute $2 \times \mathcal{V} \bmod q$ by loading register t_1 with $\mathcal{V}$. This way, we integrated the pre-computation step required in OP_1 in its normal execution. The output is available after one clock cycle and is stored in register t_2. Thus, after two clock cycles, register t_1 and t_2 are loaded with the required values mentioned in step 1 of Algorithm 1. Internal architecture of each PS unit is comprised of two identical sub-modules: SM_1 and SM_2 for PS_1, SM_3 and SM_4 for PS_2. Each SM_i unit computes a single-bit left-shift reduction. The input is shifted-left and subtracted from q. Then these two results are multiplexed and fed into the next sub-module. Thus each PS unit behaves as a two-bit shift reducer. These blocks remain idle during the OP-2 and OP-3 execution hence dedicated to OP-1 primitive.

2) MAS: The MAS block performs MA or MS operations of the two input operands. Note that our MM primitive is based on IM, where these operations are required internally in OP-1 to $\oplus|\ominus$ the shifted partial products from the accumulator t contents as specified in step 11 of Algorithm 1. To reduce hardware footprints of the proposed UBU algorithm, we reuse this block to perform OP-2 and OP-3 of the BU instead of deploying a dedicated unit. Based on the given control signals in Table I, it either performs OP-1 or executes OP-2 and OP-3 operations in a single clock cycle. OP-1 is completed in $\lceil \alpha/2 \rceil$ cycles where $\alpha = \log_2 q$. OP-2 and OP-3 can then be mapped to MAS for their execution in sequential order. The addition/subtraction of the partial products is controlled by the CGL logic which takes three bits of multiplier ω. As MAS takes one cycle thus $(\alpha/2 + 2)$ clock cycles are required to complete one BU operation in CT_{BU} and GS_{BU}. To execute INTT, the execution order of these operations is changed which can be easily achieved by configuring our UBU. We adopt the same strategy as [20] to integrate one div-by-2 operation in our MAS unit while eliminating the second one. This is done by checking even and odd values of polynomial coefficients and adding/subtracting $(q + 1)/2$ denoted as q' in Fig. 2.

Complete execution details of NTT and INTT BUs on the proposed UBU hardware are elaborated in Table I. The NTT/INTT operation is selected by the mode signal. Based on this signal, the BU operations are executed in the specified order. Table I lists the execution and data flow for all the internal BU operations for CT_{BU} and GS_{BU} required in NTT and INTT computations. A control word (CW) indicates control signals required to execute internal butterfly operations.

B. NTT Architecture

NTT is an essential tool widely deployed to speed up polynomial multiplication in LBC systems. The proposed UBU has the potential to work for any modulus type and

TABLE I
EXECUTION FLOW OF BUTTERFLY OPERATIONS ON THE PROPOSED UBU ARCHITECTURE

NTT-CT$_{BU}$						INTT- GS$_{BU}$				
$\#cc$	PS$_1$	PS$_2$	MAS	CW	out	PS$_1$	PS$_2$	MAS	CW	out
1	$t_1 = \mathcal{V}$	-	0	000xxx	-	-	-	$t = t \ominus \mathcal{V}$	1xx101	MS
2	-	$t_2 = t_1$	0	010xxx	-	$t_1 = t$	-	-	100xxx	-
3	$t_1 = \beta \odot t_1$	$t_2 = \beta \odot t_2$	$t = t(\oplus\|\ominus)d$	01101x	-	$t_2 = t_1$	-	-	110xxx	-
4	$t_1 = \beta \odot t_1$	$t_2 = \beta \odot t_2$	$t = t(\oplus\|\ominus)d$	01101x	-	$t_1 = \beta \odot t_1$	$t_2 = \beta \odot t_2$	$t = t(\oplus\|\ominus)d$	11101x	-
$\alpha/2$	$t_1 = \beta \odot t_1$	$t_2 = \beta \odot t_2$	$t = t(\oplus\|\ominus)d$	01101x	MM	$t_1 = \beta \odot t_1$	$t_2 = \beta \odot t_2$	$t = t(\oplus\|\ominus)d$	11101x	-
$(\alpha/2+1)$	-	-	$t = t \oplus \mathcal{U}$	0xx100	MA	$t_1 = \beta \odot t_1$	$t_2 = \beta \odot t_2$	$t = t(\oplus\|\ominus)d$	11101x	MM
$(\alpha/2+2)$	-	-	$t = t \oplus \mathcal{U}$	0xx101	MS			$t = t \oplus \mathcal{U}$	1xx100	MA

clock cycle (cc), control word (CW) = $\{mode, ct_1, ct_2, ct_3, ct_4, ct_5\}$,$\alpha = \log_2 q$, $\beta = 2^2$. CW is the list of control signals indicated in the proposed UBU architecture in Fig. 2. $\oplus$, $\ominus$, $\otimes$, and $\odot$ demonstrate MA, MS, MM, and left-shift mod q operations.

TABLE II
THE NTT DESIGN ($N = 256$ AND $\lceil \log_2 q \rceil = 12$-BIT) ON ARTIX-7 FPGA WITH COMPARISON TO PRIOR DESIGNS

Ref.	LUTs	#FFs	#Slices	#DSPs	#BRAMs	ESC[a]	Freq. (MHz)	$\#cc$	Time (us)	ADP[b]	$\mathcal{E}$[c]	q type
Ours $K = 2^2$	676	633	314	0	0	314	307.5	2016	6.55	2056.7	486.2	generic
Ours $K = 2^3$	1137	1176	575	0	0	575	306.4	1009	3.28	1886.01	530.22	generic
Ours $K = 2^4$	2081	2237	911	0	0	911	304.7	504	1.65	1503.15	665.27	generic
[13]	737	290	371	6	4	1771	115	474	4.68	8288.3	120.65	generic
[20]	1549	788	635	4	32	7435	159	228	1.43	10632.1	94.04	fixed
[12]	533	514	198*	1	1.5	598	246	1030	4.18	2499.6	400.06	fixed
[11]	4619	4166	1641	16	8	4841	273	84	0.31	1501.1	666.35	fixed
[16]	9508	2684	2713*	16	35	11313	172	69	0.4	4525.2	220.98	fixed
[17]	5181	4833	1468	16	0	3068	227	143	0.63	1932.8	517.37	generic
[15]	609	640	232*	2	2	832	257	490	1.9	1580.8	632.59	fixed
[14]	801	717	290*	4	2	1090	222	324	1.46	1591.4	628.38	fixed

* $\#Slices = \#LUTs \times 0.25 + \#FFs \times 0.125$ [11].
[a] ESC = $\#Slices + (\#DSPs \times 100 + \#BRAMs \times 200)$ [11], [b]ADP = ESC $\times$ Time (us), [c]$\mathcal{E}$ = throughput/ESC.

length and thus can be used in the design of any NTT architecture. However, for evaluation purposes, we used it in the NTT architecture for CRYSTALS-Kyber. In Kyber, $N = 256$ and $q = 3329$, where $q - 1 = 2^8 \cdot 13$. The NTT and INTT formulas for Kyber are given as follows:

$$\hat{p}(x) : (NTT(p(x))) = \sum_{j=0}^{\frac{N}{2}-1} p[j].\omega^{(2i+1).j} \bmod q \quad (1)$$

$$p(x) : \left(INTT(\hat{p}(x))\right) = \frac{2}{N} \sum_{j=0}^{\frac{N}{2}-1} \hat{p[j]}.\omega^{-i.(2j+1)} \bmod q \quad (2)$$

CT$_{BU}$ is utilized for NTT while to avoid the coefficient reversing, GS$_{BU}$ is used in the computation of INTT. Our proposed UBU can be configured for these BU types and perform their internal operations in the same cycle count.

The NTT and INTT are done recursively by dividing the given polynomial into smaller polynomials. There are total $\log N - 1$ stages where at each stage $\frac{N}{2}$ butterfly operations are required. At each stage, a polynomial is divided into two equal parts where coefficients are processed in the pre-defined fixed order. We deploy several copies of the UBU unit to exploit the available parallelism at each stage. The given architecture in Fig. 3 is comprised of K number of UBUs, a register module (RM), read and write logic (rlogic, wlogic), and a control unit (CU). We choose K a power-of-two to process odd and even numbers of coefficients. To start the NTT operation, the RM modules are loaded with polynomial coefficients, twiddle factors (ω), and its inverse ω^{-1}. Thus, $2K$ coefficients of a given polynomial are read from the RM and fed into their respective UBU$_{1-K}$ modules. These UBUs operate in parallel and output their results in $(\alpha + 2)$ clock

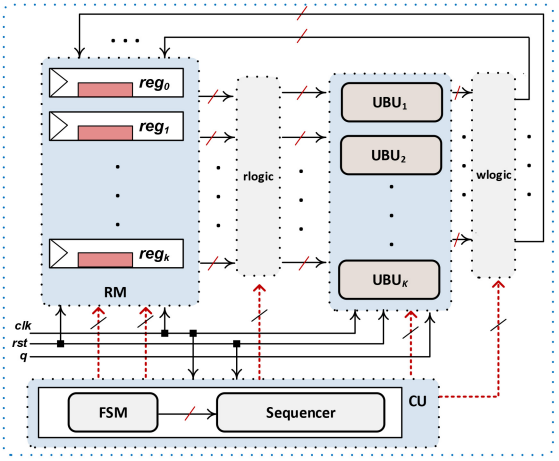

Fig. 3. Proposed NTT/INTT architecture.

cycles. The rlogic and wlogic are responsible for avoiding data hazards by reading and writing correct registers. In the case of NTT, each BU is configured to execute CT$_{BU}$ operations on the input coefficients and ω. Whereas, in the case of INTT, GS$_{BU}$ operations require ω^{-1}.

IV. IMPLEMENTATION AND RESULTS

The proposed NTT architecture is implemented using the Xilinx Vivado tool targeting Xilinx Artix-7 FPGA (XC7A350T), a popular implementation platform for PQC algorithms. We evaluate the design for three K sizes (2^2, 2^3, and 2^4) to explore tradeoffs between resource consumption and computational speed. Note that our design is DSP and BRAMs free, in contrast to almost all listed designs in

Table II. So we consider equivalent slice count (ESC) calculated as #slices + (#DSPs × 100 + #BRAMs × 200) [11]. Area-delay product (ADP) and efficiency ($\mathcal{E}$) are calculated as ESC × time (μs) and throughput (TP)/ESC, respectively. Test vectors for functional verification are generated and captured using customized Python implementation.

Table II also shows the performance of other similar designs on the same FPGA platform. Our lightweight design ($K = 2^2$), balanced design ($K = 2^3$), and low latency design ($K = 2^4$) consume 314, 575, and 911 slices, run at 307.5, 306.4, and 304.7 MHz, compute NTT/INTT operation in 6.55, 3.28, and 1.65 μs, respectively. These designs have lower ESC values than other designs. The lowest ESC is delivered by our lightweight design ($K = 2^2$). It has 5.6×, 23.67×, 1.90×, 15.41×, 36×, 9.77×, 2.6×, and 3.18× lower ESC values in comparison to [11], [12], [13], [15], [16], [17], [20], and [14], respectively. The highest throughput design ($K = 2^4$) delivers 5.52×, 7.08×, 1.66×, 3.01×, 1.28×, 1.05×, and 1.05× higher efficiency ($\mathcal{E}$) values as compared to [12], [13], [15], [16], [17], [20], and [14], respectively. It produces the same efficiency and ADP values compared to [11], the best design regarding ADP and efficiency values. Note that a design with low ADP and higher $\mathcal{E}$ values demonstrates its superiority. However, the proposed design consumes 5.31× lower FPGA slices than [11]. Moreover, [11] is a fixed design developed by exploiting the special structure of Kyber prime so it cannot be utilized for any other prime. Moreover, the proposed design with $K = 2^4$ delivers better ADP and efficiency results than the designs with $K = 2^2$ and $K = 2^3$. The proposed novel UBU is also implemented on Artix-7 FPGA as a standalone unit. It consumes 49 slices (114 LUTs + 80 FFs), runs at 312 MHz, and completes one 12-bit BU operation in 9 clock cycles.

In future work, the proposed UBU module is to be used in the development of NTT architectures for other LBC-based cryptosystems such as CYSTALS-Dilithium [5], new signatures [24], and for FHE [6] schemes. Furthermore, resistance against side-channel attacks, power and energy consumption evaluation, and error detection are to be evaluated.

V. Conclusion

Lattice-based cryptosystems are gaining popularity and will soon be deployed widely. A design that can support a generic prime offers significant reconfigurability. This brief presents an NTT architecture using a novel unified butterfly unit (UBU) for CRYSTALS-Kyber. It combines interleaved multiplication, critical path splitting, and resource-sharing strategies. It delivers better area-delay product and efficiency results on the FPGA platform. Thus, it has great potential to be deployed to accelerate a polynomial multiplication operation in LBC-based cryptosystems.

References

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[2] V. S. Miller, "Use of elliptic curves in cryptography," in *Proc. Conf. Theory Appl. Cryptogr. Techn.*, 1985, pp. 417–426.

[3] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proc. 35th Annu. Symp. Found. Comput. Sci.*, 1994, pp. 124–134.

[4] R. Avanzi et al., "CRYSTALS-Kyber algorithm specifications and supporting documentation, version 3.01" *NIST PQC Round*, vol. 2, Nat. Inst. Stand. Technol., Gaithersburg, MD, USA, document NIST PQC Round-3-20210131, 2019.

[5] D. Moody, *NIST PQC Standardization Update*, Nat. Inst. Stand. Technol., Gaithersburg, MD, USA, 2021.

[6] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. 41st Annu. ACM Symp. Theory Comput.*, 2009, pp. 169–178.

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

[8] 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, no. 17, pp. 20200234–20200234, 2020.

[9] R. Paludo and L. Sousa, "Number theoretic transform architecture suitable to lattice-based fully-homomorphic encryption," in *Proc. IEEE 32nd Int. Conf. Appl.-Specif. Syst., Archit. Process. (ASAP)*, 2021, pp. 163–170.

[10] A. C. Mert, E. Karabulut, E. Öztürk, E. Savaş, and A. Aysu, "An extensive study of flexible design methods for the number theoretic transform," *IEEE Trans. Comput.*, vol. 71, no. 11, pp. 2829–2843, Nov. 2022.

[11] M. Li, J. Tian, X. Hu, and Z. Wang, "Reconfigurable and high-efficiency polynomial multiplication accelerator for CRYSTALS-Kyber," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 8, pp. 2540–2551, Aug. 2023.

[12] 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.

[13] M. Bisheh-Niasar, R. Azarderakhsh, and M. Mozaffari-Kermani, "Instruction-set accelerated implementation of CRYSTALS-Kyber," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 11, pp. 4648–4659, Nov. 2021.

[14] M. B. Niasar, R. Azarderakhsh, and M. M. Kermani, "High-speed NTT-based polynomial multiplication accelerator for post-quantum cryptography," in *Proc. IEEE 28th Symp. Comput. Arithmetic (ARITH)*, 2021, pp. 94–101.

[15] C. Zhang et al., "Towards efficient hardware implementation of NTT for Kyber on FPGAs," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2021, pp. 1–5.

[16] F. Yaman, A. C. Mert, E. Öztürk, and E. Savaş, "A hardware accelerator for polynomial multiplication operation of CRYSTALS-Kyber PQC scheme," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2021, pp. 1020–1025.

[17] L. Ma, X. Wu, and G. Bai, "Parallel polynomial multiplication optimized scheme for CRYSTALS-Kyber post-quantum cryptosystem based on FPGA," in *Proc. Int. Conf. Commun., Inf. Syst. Comput. Eng. (CISCE)*, 2021, pp. 361–365.

[18] P. Barrett, "Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor," in *Proc. Conf. Theory Appl. Cryptogr. Techn.*, 1986, pp. 311–323.

[19] P. L. Montgomery, "Modular multiplication without trial division," *Math. Comput.*, vol. 44, no. 170, pp. 519–521, 1985.

[20] W. Guo, S. Li, and L. Kong, "An efficient implementation of KYBER," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 69, no. 3, pp. 1562–1566, Mar. 2022.

[21] G. R. Blakely, "A computer algorithm for calculating the product AB modulo M," *IEEE Trans. Comput.*, vol. 100, no. 5, pp. 497–500, May 1983.

[22] K. Javeed and D. Gregg, "Point multiplication accelerator for arbitrary Montgomery curves," *IEEE Embed. Syst. Lett.*, early access, May 9, 2024, doi: 10.1109/LES.2024.3399071.

[23] K. Javeed, "FPGA implementation of area-time aware ECC scalar multiplication core," in *Proc. 30th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, 2023, pp. 1–4.

[24] *Post Quantum Cryptography (Digital Signatures)—Round 1 Additional Signatures*, Nat. Inst. Stand. Technol., Gaithersburg, MD, USA, 2023.