IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 28, 2021, accepted January 12, 2022, date of publication January 25, 2022, date of current version February 3, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3145988

Configurable Mixed-Radix Number Theoretic Transform Architecture for Lattice-Based Cryptography

PHAP DUONG-NGOC, (Member, IEEE), AND HANHO LEE, (Senior Member, IEEE)
Department of Information and Communication Engineering, Inha University, Incheon 22212, South Korea

Corresponding author: Hanho Lee (hhlee@inha.ac.kr)

This work was supported by the Ministry of Science and ICT (MSIT) under the Information Technology Research Center (ITRC) support program (IITP-2021-0-02052) supervised by the Institute for Information & Communications Technology Planning & Evaluation (IITP), in part by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2021R1A2C1011232), and in part by the IITP grant funded by the Korea government (MSIT) (No. 20210007790012003).

ABSTRACT Lattice-based cryptography continues to dominate in the second-round finalists of the National Institute of Standards and Technology post-quantum cryptography standardization process. Computational efficiency is primarily considered to evaluate promising candidates for final round selection. In lattice-based cryptosystems, polynomial multiplication is the most expensive computation and critical to improve the performance. This paper proposes an efficient number theoretic transform (NTT) architecture to accelerate the polynomial multiplication. The proposed design applies mixed-radix multi-path delay feedback architecture and flexibly adopts various polynomial sizes. Configurable NTT design is realized to perform forward and inverse NTT computations on a unified hardware, which is then used to develop an efficient polynomial multiplier. The proposed architectures were successfully accelerated on several Xilinx FPGA platforms to directly compare with state-of-the-art works. The implementation results show that the proposed NTT architectures have comparable area-time product and demonstrate 1.7∼17× performance improvement, and the proposed polynomial multipliers achieve higher performance compared with previous works. Experimental results confirmed the proposed design's applicability for high-speed large-scale data cryptoprocessors.

INDEX TERMS Lattice-based cryptography, number theoretic transform, mixed-radix, multi-path delay feedback, post-quantum cryptography.

I. INTRODUCTION

In the explosive era of internet-of-things (IoT) and rapid development of next generation networks, post-quantum cryptography (PQC) has attracted increasing interest for securing data privacy against potential quantum attacks. The National Institute of Standards and Technology (NIST) second-round report [1] showed that lattice-based cryptography (LBC) schemes have become promising candidates for future standardization. These schemes include public-key encryption (PKE) protocols, key-establishment mechanisms (KEM), and digital signatures with strong theoretical security guarantees, such as CRYSTALS-Kyber (shortly Kyber) [2], Saber [3], CRYSTALS-Dilithium [4], and NewHope [5]. Polynomial multiplication is the critical bottleneck in LBC systems. Naïve polynomial multiplication in n-point cyclotomic rings is expensive with time complexity of $\mathcal{O}(n^2)$ operations. Fortunately, the number theoretic transform (NTT) is a strong tool with efficient memory utilization that performs the polynomial multiplication in $\mathcal{O}(n \log n)$ operations. Thus, designing an efficient NTT architecture has top priority to accelerate cryptoprocessors, particularly for multi-connected cryptosystems and large-scale data encrypting applications, e.g., IoT [6], biometrics [7], and video-based facial images [8].

There are many approaches to design NTT architectures based on the trade-off between throughput rate and hardware complexity. Conventional methods construct a mutual butterfly circuit and iteratively execute NTT stages based on a memory unit. In each stage, groups of coefficients are fetched from the memory unit, transformed, and written back to the memory in appropriate orders. However, the memory-based NTT architecture is constrained by the cost of memory access operations. The memory access scheme changes from stage

The associate editor coordinating the review of this manuscript and approving it for publication was Weizhi Meng.

to stage and the memory read/write operations are complex due to data-dependency between adjacent NTT stages.

Several previous studies have investigated strategies to reduce the access pattern complexity of memory-based approaches. Xing and Li proposed an NTT architecture with four multipurpose butterfly units (BUs) and memory ping-pong strategy to settle the non in-place writing issue between adjacent stages [9]. The ping-pong technique helped simplify the memory access operations but doubling the memory size. Zhang *et al.* presented a low-complexity compact NTT/INTT architecture for highly efficient NewHope-NIST on an FPGA device [10]. Their NTT architecture's outstanding strength came from the address conflict-free multi-bank memory access scheme, which was efficient with a small number of parallel radix-2 BUs. Yaman *et al.* subsequently proposed three different hardware architectures (lightweight, balanced, high-performance) for polynomial multiplication in Kyber [11]. Their unified BU architecture could support butterfly operations and point-wise multiplication (PWM). The NTT, INTT, and polynomial multiplication could be done using a unified hardware design with different numbers of execution clock cycles (CCs). Recently, Bisheh-Niasar *et al.* have presented the state-of-the-art FPGA-based implementation of high-speed NTT architecture for Kyber [12]. Four BUs have been grouped in a 2×2 BU array together with several optimization strategies to speed-up the NTT computation. Bisheh-Niasar *et al.* have then proposed a reconfigurable resource-efficient NTT architecture supporting the Kyber [13]. In which, multiple BUs have been paralleled based on the memory ping-pong strategy to adopt various Kyber configurations. However, when increasing the number of parallel BUs for high speed, the memory banks will double and the routing becomes more complex. In pipeline executing aspect, the aforementioned memory-based NTT architectures might cause some bubble cycles between computation rounds due to the data-dependency of adjacent stages. Furthermore, the final NTT results are only produced at the last computational stage after a delay of $\log n$ stages, which causes a big gap between two consecutive polynomials. These shortcomings are the obstacle to improve performance of the NTT design.

To speed up the NTT computation, some previous studies deployed multiple BUs in parallel. Feng *et al.* implemented d lanes of BUs (d is set up to 16) to target at high-speed polynomial multiplier [14]. Mert *et al.* also proposed a flexible NTT design for various parameter sets and increased the number of processing elements up to 32 at the expense of more hardware resources [15]. The deployment of multiple BUs in parallel aims to reduce the number of execution CCs. Nevertheless, this strategy often requires high memory bandwidth making the on-chip memory access pattern complicated, which are challenging to improve the clock frequency for a high-speed NTT design.

A different approach would be to develop highly pipelined NTT architectures based on systolic array technique. Rentería-Mejía and Velasco-Medina proposed a high-radix multi-path delay commutator NTT architecture to accelerate the ring learning with error based cryptoprocessors [16]. Tan and Lee subsequently proposed an efficient multi-path delay feedback (MDF) NTT architecture specifically for short-term security parameters [17]. Recently, Duong and Lee have implemented the MDF NTT architectures of k-parallel data paths for 1024-point polynomial [18], which confirmed the reasonable choice of k can achieve high efficiency. However, these works have not fully investigated the flexibility and reconfiguration of NTT/INTT architecture for different lattice-based cryptography schemes.

This paper focuses on implementing an efficient NTT architecture specifically for high-speed computing environments. The proposed approach deploys all computational stages in fully pipelined and parallel manner. The NTT design transforms polynomials sequentially, in which polynomial coefficients are generated every CC with multiple parallel data paths. In addition, advanced cryptography protocols can support various security levels and perform NTT or INTT on-demand out-of-turn between parties. Thus, expanding previous designs from specific to more generic and configurable settings would be significant for advanced cryptosystems. In NIST's current view, Kyber is the most promising LBC candidate for PKE/KEM standardization at the end of the third round [1]. However, other LBC schemes and their variants are still worthy in different research areas based on their novel ideas, different security standards and potential for further improvement. This work illustrates the reconfigurable capability of the proposed NTT architecture by employing parameter sets (n, q) of polynomial degree n and modulo prime q such as (1024, 12289) and (512, 12289) in NewHope [5], (256, 3329) in Kyber [2] of the second and the third round NIST PQC submissions, respectively. The proposed NTT engine operates as a major accelerator and flexibly switches between parameter sets. These parameter sets satisfy various security strength categories required in NIST PQC Call for Proposals [19].

We summarize the main contributions of present paper as follows:

1) Building on the prior NTT architecture [18], we select a rational radix value (i.e., $k = 4$) and propose a flexible mixed-radix MDF NTT architecture supporting various parameter sets of (n, q). The proposed NTT architecture has four parallel data paths and is fully pipelined for high throughput implementation. Output polynomials are produced sequentially after execution time of $\frac{n}{4}$ CCs. Additional multiplexers select the configuration corresponding to the given parameter set.
2) A configurable NTT/INTT architecture is realized to perform NTT and INTT computations on a unified hardware. The configurable design completes the NTT and INTT computations in the same number of execution CCs and provides a versatile tool for implementing cryptographic algorithms and reducing the hardware costs for high performance cryptosystems.

3) We develop a polynomial multiplier using the configurable NTT/INTT architecture. The proper scheduling of configurable modules performs the polynomial multiplication effectively. Experiment verified that the proposed architecture achieved higher performance and better efficiency compared with previous works.

The remainder of this paper is organized as follows. Section II gives the background of NTT. Section III proposes a flexible mixed-radix MDF NTT architecture and a configurable NTT/INTT based polynomial multiplier. Section IV presents implementation results and discussion. Finally, Section V summarizes and concludes the paper.

II. NUMBER THEORETIC TRANSFORM

NTT is a form of fast Fourier transform (FFT) in a finite field with integers and provides an effective tool to perform the polynomial multiplication in LBC schemes. Algorithm 1 describes a fast iterative radix-2 NTT computation in ring $R_q = Z_q/(x^n + 1)$, where n is a power of two and q is a prime number [20]. This approach applies the negative wrapped convolution method to avoid zero-padding and eliminate modulo $(x^n + 1)$ in polynomial multiplication. $\psi \in Z_q$ is defined as the square root of ω (i.e., the primitive n-th root of unity), where $\psi^2 = \omega \bmod q$. Powers of ψ (denoted as ψ^i, $i = 1 \sim n - 1$) are pre-computed and listed in $\Psi[j]$ where j is the bit reversal of i. We apply the Cooley-Tukey (CT) and Gentleman-Sande (GS) algorithms to perform NTT and INTT respectively, which help to avoid expensive reordering steps [21]. The function *ModMult*() performs the modular multiplication by q. For polynomial $a = (a[0], \ldots, a[n-1])$ in R_q as an example, the NTT and INTT forms are respectively defined as follow,

$$A[i] = NTT_{\psi}^{CT}(a) = \psi^i \sum_{j=0}^{n-1} a[j]\omega^{ij} \bmod q \tag{1}$$

Algorithm 1 Iterative Radix-2 NTT Algorithm [20]

Input: $a(x) \in Z_q[x]/(x^n + 1)$, Ψ lists n powers of ψ in bit-reversed order
Output: $A = NTT_n(a)$

```
A ← a
for (m = 1; m < n; m = 2 * m) do
    for (i = 0; i < m; i = i + 1) do
        W ← Ψ[m + i]
        for (j = i*n/m; j < (2i+1)n/2m; j = j + 1) do
            temp ← ModMult(W, A[j + n/m], q)
            A[j + n/m] ← A[j] − temp mod q
            A[j] ← A[j] + temp mod q
        end for
    end for
end for
return A
```

$$a[i] = INTT_{\psi^{-1}}^{GS}(A) = n^{-1}\psi^{-i} \sum_{j=0}^{n-1} A[j]\omega^{-ij} \bmod q. \tag{2}$$

When merging the powers of ψ^i and ψ^{-i} into ω^{ij} and ω^{-ij} respectively, the polynomial multiplication of a and b can be computed as follow,

$$c = a \times b = INTT_{\psi^{-1}}^{GS}(NTT_{\psi}^{CT}(a) \circ NTT_{\psi}^{CT}(b)). \tag{3}$$

However, implementation of this algorithm on hardware platforms is challenging to achieve desired throughput. Therefore, we propose a mixed-radix NTT algorithm to adopt the proposed method for high-speed hardware accelerators.

III. PROPOSED MIXED-RADIX NTT ARCHITECTURE

A. PROPOSED MIXED-RADIX $2^{k_1}/2^{k_2}$ NTT ALGORITHM

Algorithm 2 performs the NTT computation of n-point polynomial using the mixed-radix method, assuming that n is decomposed into two components: radix-2^{k_1} and radix-2^{k_2}. Hence, 2^{k_1}-point NTT is performed 2^{k_2} times to accomplish the first transformation. A subsequent reordering function reorders the coefficients for the next stage. The second transformation performs 2^{k_2}-point NTT 2^{k_1} times. And the last reordering function generates the final output in bit-reversed order. With the pre-computed twiddle factor (TF) constants listed in bit-reversed order, 2^{k_2} radix-2^{k_1} NTT computations use the same first 2^{k_1} TF constants, and 2^{k_1} radix-2^{k_2} NTTs use remaining 2^{k_1} groups of 2^{k_2}-1 TF constants respectively. The proposed algorithm has ability to parallel 2^{k_2} radix-2^{k_1} NTT computations efficiently. Radix-2^{k_2} NTT operation can transforms received coefficients with parallel BUs in each stage and generates result in bit-reversed order.

Algorithm 2 Proposed Mixed-Radix $2^{k_1}/2^{k_2}$ NTT Algorithm

Input: $a(x) \in Z_q[x]/(x^n + 1)$, $n = 2^{k_1} \times 2^{k_2}$
Output: $A = NTT_n(a)$

```
for i = 0 to 2^k2 − 1 do
    for j = 0 to 2^k1 − 1 do
        temp[j] ← a[j × 2^k2 + i]
    end for
    b[i × 2^k1] ←NTT_2^k1(temp)          ▷ radix-2^k1 NTT
end for
B ← reorder(b)
for j = 0 to 2^k1 − 1 do
    for i = 0 to 2^k2 − 1 do
        temp[i] ← B[i × 2^k1 + j]
    end for
    b[j × 2^k2] ←NTT_2^k2(temp)          ▷ radix-2^k2 NTT
end for
A ← reorder(b)
return A
```

Fig. 1 illustrates the data-flow of mixed-radix NTT approach, which clearly shows butterfly operations and data dependency between adjacent stages. Algorithm 1 is used to

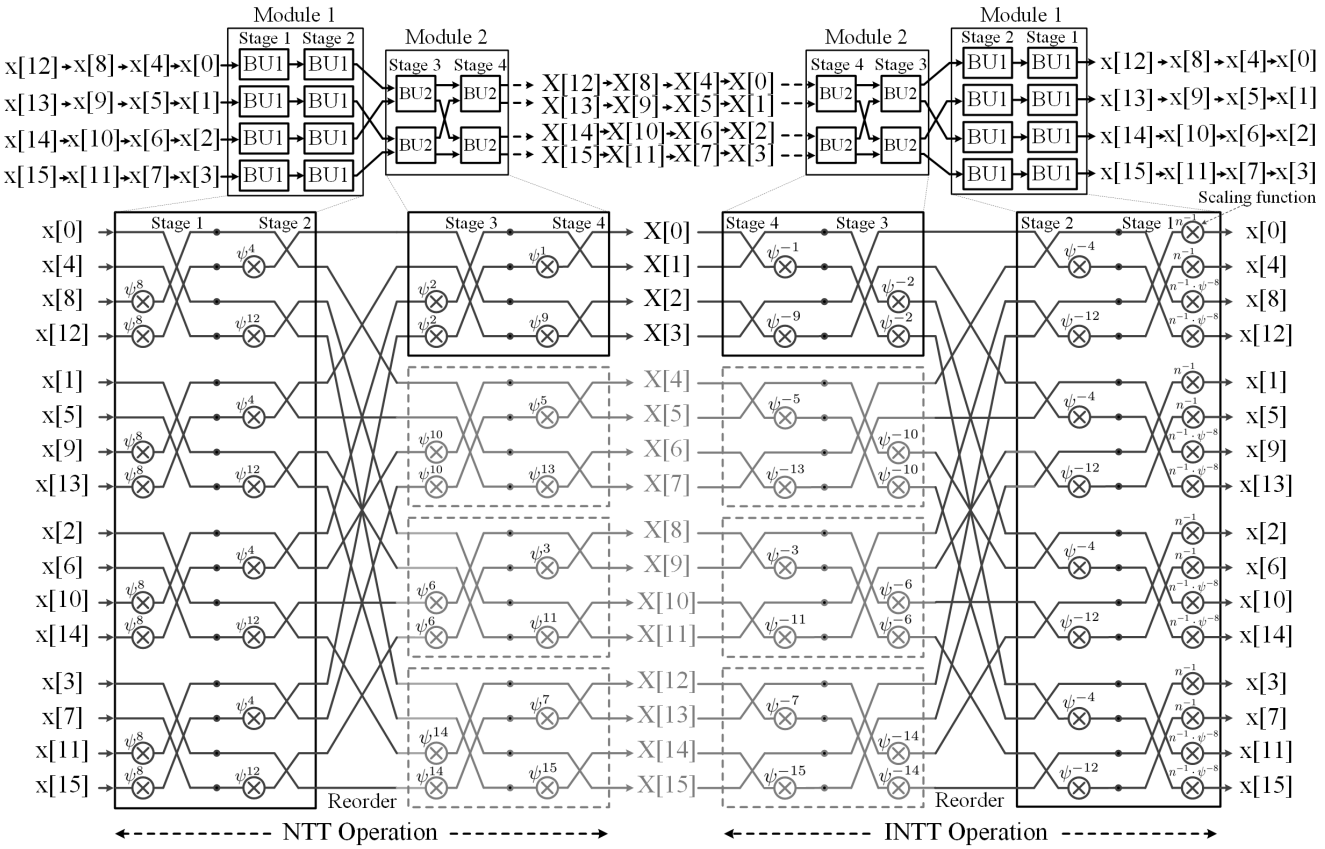

FIGURE 1. Dataflow graph of mixed-radix NTT and INTT operations when n is 16 ($2^2/2^2$). Module 1 parallels radix-2^{k_1} computations whereas Module 2 sequentially performs radix-2^{k_2} computations (faint lines). Scaling factor n^{-1} is merged into Stage 1 BU1 operation to complete the INTT computation.

perform partial NTT computations for radix-2^{k_1} and radix-2^{k_2}. Each partial NTT respectively requests TFs ψ in Ψ at j-th step as follows,

$$\text{For radix-}2^{k_1}\begin{cases} W \leftarrow \Psi[m+i] \text{ where:} \\ \bullet\ m = 1 \ll j, \text{ with } j = 0 \sim k_1 - 1 \\ \bullet\ i = 0 \sim m - 1 \end{cases}$$

and

$$\text{For radix-}2^{k_2} = \begin{cases} W \leftarrow \Psi[2^{k_1} \times m + i + iter] \text{ where:} \\ \bullet\ m = 1 \ll j, \text{ with } j = 0 \sim k_2 - 1 \\ \bullet\ i = 0 \sim m - 1 \\ \bullet\ iter = 0 \sim 2^{k_1+i} - 1, \end{cases}$$

where TFs in each stage are divided into m groups and each group with corresponding order i contains multiple BUs. The number of groups doubles when the stage index increases by one, and the number of BUs in each group halves. Therefore, each radix-2^{k_1} NTT uses the same group of TF constants for 2^{k_2} computational instances, whereas each radix-2^{k_2} NTT uses different groups of TF constants for k_2 computational stages run by *iter* indices respectively.

For example, Fig. 2 illustrates the order of TF and inverse TF assigned to NTT and INTT computational stages when

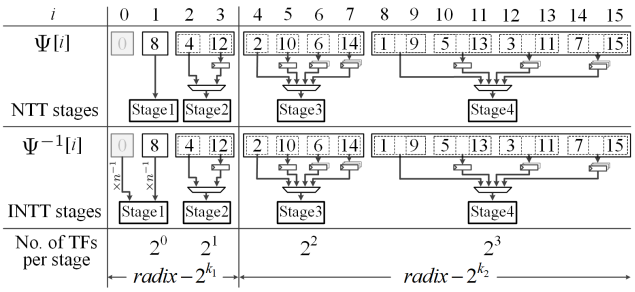

FIGURE 2. The TF and inverse TF constants are assigned to corresponding stages of the NTT and INTT operations for example in Fig. 1, respectively.

n is 16. Assuming $k_1 = k_2 = 2$, four partial $\text{NTT}_{2^{k_1}}$ computations use the same group of TF constants with the consecutive orders through two computational stages (e.g., $\Psi[1]$ for Stage 1, $\Psi[2]$ and $\Psi[3]$ for Stage 2). Meanwhile, four partial $\text{NTT}_{2^{k_2}}$ computations require different groups of TF constants for each computation (e.g., $\Psi[4]$, $\Psi[8]$ and $\Psi[9]$ for two respective stages of first $\text{NTT}_{2^{k_2}}$). *iter* points to the order in Ψ corresponding to each stage of each $\text{NTT}_{2^{k_2}}$ computation. The same order scheme of inverse TF constants is used for INTT computation except that the scaling factor n^{-1}

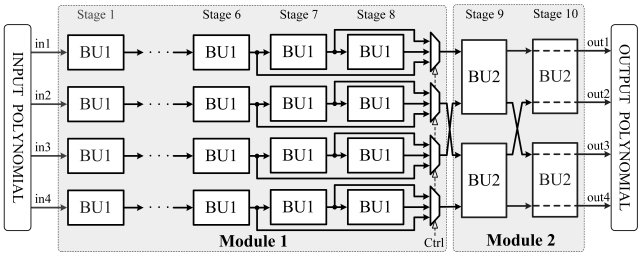

FIGURE 3. Proposed flexible mixed-radix MDF NTT architecture for 256, 512, and 1024-point polynomials ($k_2 = 2$). Module 2 bypasses "Stage 10" when supporting Kyber parameter set [2].

is early multiplied by inverse TF constants for Stage 1 INTT computation.

B. OVERALL FLEXIBLE NTT ARCHITECTURE DESIGN

Fig. 3 illustrates the proposed flexible mixed-radix MDF NTT architecture based on the aforementioned algorithms intended for speed-optimized LBC schemes. Considering the evaluation results of prior study [18], we select the optimal radix value ($k_2 = 2$) and design a flexible NTT architecture for various parameter sets (n, q). The proposed NTT architecture includes two modules with respective BU1 and BU2. Module 1 adopts the MDF architecture to performs k_1 computational stages with four parallel data paths. This module continuously receives input vectors in normal order and generates four coefficients per CC. Module 2 with two parallel BU2s in each stage directly transforms four coefficients received from Module 1. The design methodology naturally eliminates the reordering steps in Algorithm 2. After k_2 computational stages, Module 2 generates the final results in bit-reversed order. Four additional multiplexers select the data direction corresponding to the given parameter set. Specially, the NTT operation in Kyber [3] is slightly different by using n-th roots of unity instead of $2n$-th roots. However, the NTT computation on a 256-point polynomial can be performed on two separate 128-point classic ones according to the parity of index. The proposed NTT architecture concurrently performs two 128-point NTTs in 7 stages by bypassing the last stage (i.e., Stage 10).

Fig. 4 examines BU1 and BU2 structures with the optimization of critical path. Module 1 deploys serial k_1 BU1s in each data path to perform 2^{k_1}-point NTTs. BU1 structure composes of modular multiplier (MM), modular adder (MA), modular subtractor (MS), and first-in-first-out (FIFO). Meanwhile, each Module 2 stage constructs two BU2s in parallel to concurrently transform four coefficients received from Module 1. BU2 structure differs from BU1 in requiring only MM, MA, and MS for the butterfly circuit. TF constants are distinctly assigned into BUs corresponding to the NTT stage. The same first 2^{k_1} TF constants are used for parallel data paths in Module 1's k_1 stages. Module 2 uses the next 2^{k_1} TFs for the first stage and remaining 2^{k_1+1} TFs for the last stage. The design methodology facilitates the TF assignment through various computational stages.

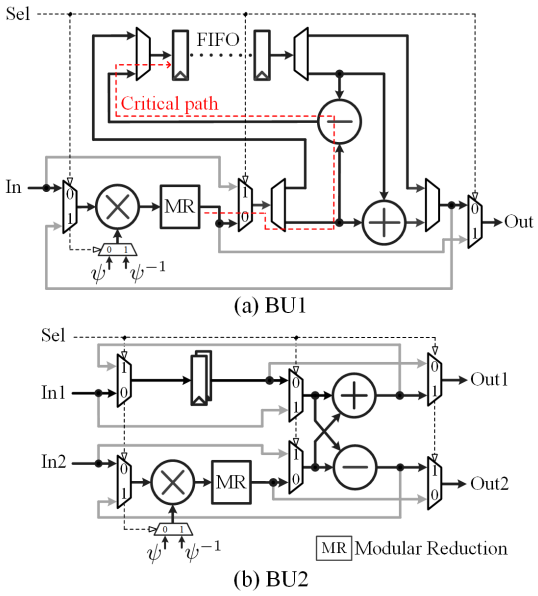

FIGURE 4. Proposed configurable BU structures: (a) BU1 is for MDF-based computations in Module 1 and (b) BU2 is for parallel operations in Module 2. Faint lines are used for INTT operation.

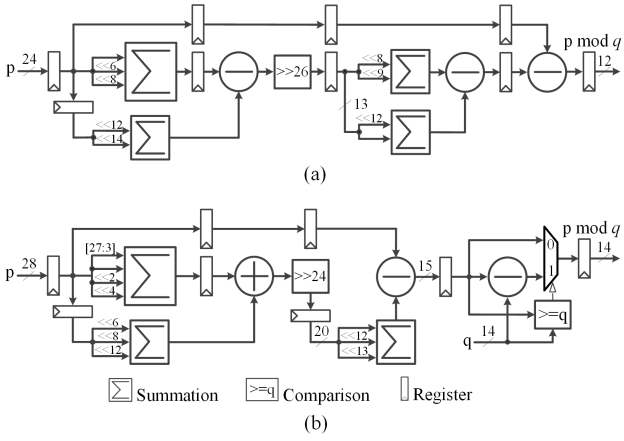

FIGURE 5. Architectures for modular reduction of primes (a) $q = 3329$ and (b) $q = 12289$ ((b) modified from [18]).

Additionally, modular reduction (MR) is the most time-consuming component of these BUs. The realization of MRs of specific primes was effectively executed in [22]. We realize and modify MR implementation for $q = 3329$ and 12289 as shown in Fig. 5 (a) and (b), respectively. The compact MR architectures are fully pipelined with the same latency (i.e., 5 CCs) using only bit-shift and addition to avoid expensive integer multiplications.

The idea behind the mixed-radix MDF NTT architecture is to implement a fully pipelined design without reordering buffers. The continuous transformation between modules completely removes redundant cycles and eliminates additional memory for intermediate results. Taking $n = 1024$ for example, Fig. 6 describes the timing diagram of the proposed NTT architecture. For the very first input polynomial, the

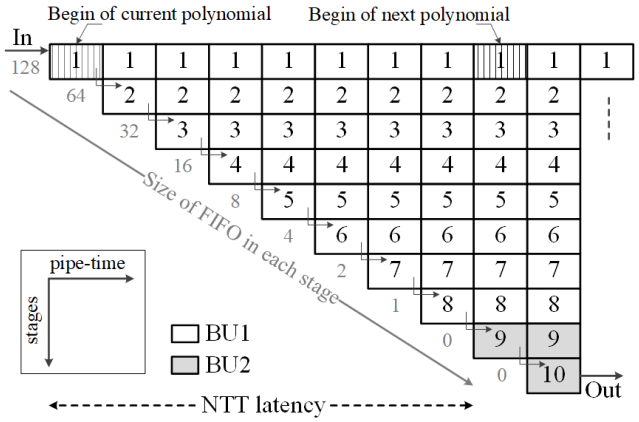

FIGURE 6. Pipeline timing diagram of NTT operation for Fig. 3 when n is 1024. The size of FIFOs halves gradually in first eight stages which comprises CCs of NTT latency.

number of execution CCs through ten NTT stages is calculated by the sum of FIFO delay (255 CCs), pipelined BU stages (1 CC for multiplier, 5 CCs for MR, and 1 CC for modular addition/subtraction), and pipeline registers between stages (9 CCs), i.e., $255 + 70 + 9 = 334$ CCs. Because the pipeline is fulfilled, the NTT design accepts the next input polynomial after $\frac{n}{4}$ CCs. Hence, the NTT design generates four coefficients every CC and transforms input vectors every $\frac{n}{4}$ CCs in sequential manner. Moreover, the proposed NTT architecture is flexible to support various parameter sets with additional multiplexers. The reasonable choice of k_2 provides an effective trade-off between throughput and resource utilization for high performance LBC systems.

C. CONFIGURABLE NTT/INTT ARCHITECTURE AND EFFICIENT POLYNOMIAL MULTIPLIER

Fig. 7 shows a configurable NTT/INTT architecture to perform the NTT and INTT computations on a unified hardware. The INTT operation differs from the NTT with mirror-symmetric data-flow topology and scaling step by n^{-1}. The proposed configurable architecture includes Module 1, Module 2, NTT/INTT Select units, and Controller unit. Module 1 and 2 can exchange execution order, in that the Module 1 first stage in the NTT operation becomes the last stage of INTT computation. The Controller determines which configuration mode is selected. Additional multiplexers are added to change the direction of data paths, shown as faint lines in Fig. 4. Scaling factor n^{-1} in the INTT computation is prior-merged into the first two addresses of Ψ^{-1} as illustrated in Fig. 2 to eliminate redundant cycles. Hence, the configurable architecture performs the NTT and INTT computations in the same number of $\frac{n}{4}$ execution CCs. In terms of hardware utilization, the configurable architecture consumes more logic circuits to implement additional multiplexers compared with the flexible design. The configurable NTT/INTT architecture is significant to develop an

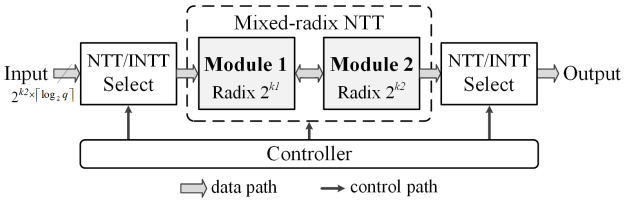

FIGURE 7. Block diagram of proposed configurable NTT/INTT architecture.

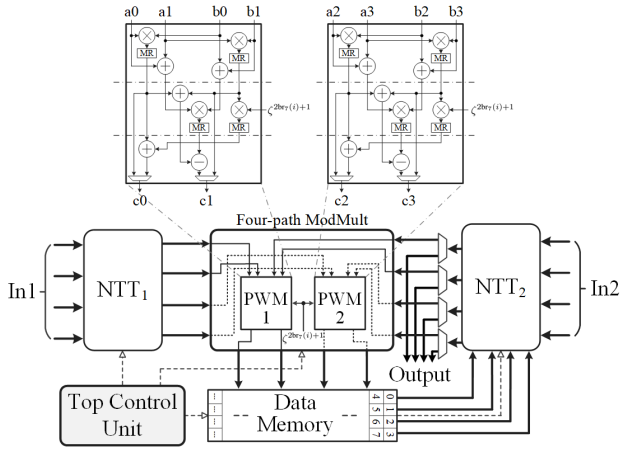

FIGURE 8. Top-level view of the proposed polynomial multiplier architecture. Some of pipeline registers in Four-path ModMult are omitted for the sake of simplicity.

efficient polynomial multiplier as presented in the following paragraph.

Fig. 8 shows a polynomial multiplier architecture using the proposed NTT and INTT architectures. Applying the equation (3), the proposed polynomial multiplier initially uses NTT_1 and NTT_2 to perform forward transformations on two input polynomials In1 and In2, respectively. Subsequently, Four-path ModMult unit executes the PWM. The Data Memory block is required to store PWM results. Then, only the NTT_2 is configurable to perform the INTT computation on demand to produce polynomial multiplication results. Additional multiplexers are used to select the final output of polynomial multiplication. The operation of functional modules is orchestrated by Top Control Unit. Configurable NTT/INTT architecture is beneficial to reduce the hardware complexity of one NTT operation in the polynomial multiplication. Additionally, the Four-path ModMult is specifically designed to support the basecase PWM in Kyber beside performing the classic PWM. The operation of basecase PWM can reduce the number of integer multiplications from five to four [23] as follows,

$$\begin{aligned}
&\hat{h}_{2i} + \hat{h}_{2i+1}X \\
&\quad = (\hat{f}_{2i} + \hat{f}_{2i+1}X)(\hat{g}_{2i} + \hat{g}_{2i+1}X) \bmod X^2 - \zeta^{2\text{br}_7(i)+1} \\
&\quad = \hat{f}_{2i}\hat{g}_{2i} + \hat{f}_{2i+1}\hat{g}_{2i+1}\zeta^{2\text{br}_7(i)+1} \\
&\qquad + (\hat{f}_{2i} + \hat{f}_{2i+1})(\hat{g}_{2i} + \hat{g}_{2i+1}) - \hat{f}_{2i}\hat{g}_{2i} + \hat{f}_{2i+1}\hat{g}_{2i+1},
\end{aligned}$$

where $\zeta = 17$ is the first primitive 256-th root of unity and $\text{br}_7(i)$ is the bit reversal of the unsigned 7-bit integer. Experiment showed that the Four-path ModMult unit is fully pipelined and performs the basecase PWM in 17 CCs whereas the classic PWM of remaining parameter sets is completed in 6 CCs with four parallel modular multiplications. Every clock cycle, four coefficients are generated, concatenated and sequentially stored in the Data Memory block. The memory writing and reading are independent and simply scheduled by the Top Control Unit.

IV. IMPLEMENTATION RESULTS AND DISCUSSION

The proposed mixed-radix MDF NTT architectures were modeled using the Verilog hardware description language and synthesized using the Xilinx Vivado 2020.1. To directly compare with related works on similar FPGA platforms, the implementation results were placed-and-routed on four 28nm FPGA devices: (1) a Xilinx Zynq-7000 (xc7z020clg484) that has 53K look-up table (LUT) elements, 106K flip-flops (FFs), 220 digital signal processing (DSP) slices, and 140 Block RAMs (BRAMs); (2) a Xilinx Virtex-7 (xc7vx485tffg1761) that has 303K LUTs, 607K FFs, 2800 DSPs, and 1030 BRAMs; (3) a Xilinx Artix-7 (xc7a200tfbg676) that has 135K LUTs, 269K FFs, 740 DSPs, and 365 BRAMs; and (4) a Xilinx Spartan-7 (xc7s100fgga676) that has 64K LUTs, 128K FFs, 160 DSPs, and 120 BRAMs. Resource consumption and achievable clock frequency were obtained with default place-and-route settings. We introduced area-time product (ATP) and hardware efficiency metrics to enable fair comparison with previous works due to various hardware resource types.

Table 1 shows key implementation results of the proposed flexible NTT architectures compared with previous studies for various parameter sets (n, q). The second and third columns of this table show the numbers of CCs and achievable clock frequencies. The fourth column shows the execution time of NTT designs that their latency is calculated as Latency (μs) = CCs/Freq. (MHz). For $n = 1024$, the proposed NTT architecture operates approximately 9.6×, 12×, 8×, and 2× faster than that of [9], [10], [12], and [15] respectively. Xing and Li [9] proposed a ping-pong NTT architecture that used four BUs and required a large number of CCs (i.e., 1280). Zhang *et al.* [10] used only two parallel BUs for the iterative computation, which utilized hardware resources effectively but consumed many CCs (i.e., 2569). Although Mert *et al.* [15] significantly reduced the number of CCs (i.e., 200) by paralleling 32 processing elements, their NTT architecture operated at lower clock frequency and required more hardware resources. Bisheh-Niasar *et al.* [12] grouped two NTT stages into each computational round by constructing the 2 × 2 BU array. Their approach proposed to reduce the access pattern complexity but still required a large number of CCs (i.e., 1591). For $n = 512$, the proposed NTT architecture runs approximately 12× and 2× faster than that of [10] and [15], respectively. For $n = 256$, Yaman *et al.* [11] deployed 16 BUs in high-performance unified hardware architecture and significantly reduced the CC number of the NTT operation (i.e., 69). Bisheh-Niasar *et al.* [12] deployed 2 × 2 BU array and improved the access pattern to reduce the computational cycle (i.e., 324 CCs). Meanwhile, Bisheh-Niasar *et al.* [13] employed two configurable BUs in parallel, which required a larger number of CCs (i.e., 474) and performed the NTT computation at low clock frequency. However, our fully pipelined NTT design has smallest CC number and outperforms that of [11], [12], and [13] approximately 1.7×, 6×, and 17× acceleration, respectively. Thus, the proposed NTT architecture achieves superior performance compared to previous approaches.

To compare efficacy among NTT architectures, we evaluated ATP metric of the trade-off between area requirement and latency. The fifth through eleventh columns report the utilized numbers of LUTs, FFs, DSPs, and BRAMs and their corresponding ATP values. The ATP values are measured by product of latency and utilized number of LUTs, FFs, and DSPs, and denoted as ATP_LUT, ATP_FF, and ATP_DSP, respectively. The proposed NTT architecture does not require additional memory, so we omit the ATP value of BRAM. As shown in the Table 1, the state-of-the-art NTT architecture for $n = 1024$ in [12] has smallest overall ATP value (see [13]). For $n = 512$, the NTT architecture in [10] has smallest ATP values measured by LUTs, FFs, and DSPs, which benefiting from the conflict-free memory access operation specifically for two parallel radix-2 BUs. Our NTT architectures have comparable ATPs with slightly higher values but do not consume any BRAM compared with six and two that in [10] and [12] for $n = 1024$, and five that in [10] for $n = 512$, respectively. For $n = 256$, the proposed NTT architecture has slightly better ATP value of LUTs, comparable ATP values of FFs, DSPs and without requiring BRAM compared with previous studies. Next, we evaluate and compare the speed of various NTT designs through achievable throughput in the following paragraph.

We introduce the throughput metric to measure the amounts of bits passing through the NTT accelerator for a second as follows,

$$\textit{Throughput (bps)} = \frac{\textit{Number of bits}}{\textit{Latency (s)}}. \tag{4}$$

The twelfth column of Table 1 compares the throughput of various NTT designs. The proposed NTT architectures achieve highest throughput among the NTT designs of various parameter sets. Specifically compared with state-of-the-art studies, our NTT designs can deliver significant throughput approximately 12× and 8× that of [10] and [12] for $n = 1024$, 12× that of [10] for $n = 512$, and 1.7×, 6×, and 17× that of [11], [12], and [13] for $n = 256$, respectively.

Table 2 compares the configurable NTT/INTT based polynomial multipliers with previous works. Hardware efficiency is introduced as a fair comparison metric due to different modulo prime sizes among the polynomial multipliers. The hardware efficiency is used to evaluate the throughput that

TABLE 1. Implementation results for the proposed flexible NTT architectures compared with previous approaches.

Design	CCs	Freq. (MHz)	Latency (μs)	LUT		FF		DSP		BRAM	Throughput (Mbps)	Device
				#LUTs	ATP_LUT[a]	#FFs	ATP_FF[a]	#DSPs	ATP_DSP[a]			
$n = 1024; q = 12289$												
Xing [9]	1280	153	8.37	4823	4.3	2901	3.9	8	2.1	0	1673	Zynq-7000
Zhang [10]	2569	244	10.53	847	0.9	375	0.6	2	0.7	6	1330	Zynq-7000
This work	**256**	**293**	**0.87**	**10765**	**1**	**7146**	**1**	**36**	**1**	**0**	**16023**	**Zynq-7000**
Bisheh-Niasar [12]	1591	234	6.80	798	0.6	715	0.8	4	0.9	2	2059	Artix-7
Mert [15]	200	125	1.60	17188	3.0	No data	-	96	5.0	48	8750	Virtex-7
This work	**256**	**302**	**0.85**	**10758**	**1**	**7146**	**1**	**36**	**1**	**0**	**16516**	**Virtex-7**
$n = 512; q = 12289$												
Zhang [10]	1289	245	5.26	741	1.0	330	0.6	2	0.8	5	1330	Zynq-7000
This work	**128**	**293**	**0.44**	**8713**	**1**	**6538**	**1**	**32**	**1**	**0**	**16023**	**Zynq-7000**
Mert [15]	108	125	0.86	16983	4.0	No data	-	96	6.1	48	8102	Virtex-7
This work	**128**	**302**	**0.42**	**8712**	**1**	**6538**	**1**	**32**	**1**	**0**	**16516**	**Virtex-7**
$n = 256; q = 3329$												
Yaman [11]	69	172	0.40	9508	4.0	2684	1.0	16	1.0	35	7478	Artix-7
Bisheh-Niasar [12]	324	222	1.46	801	1.2	717	1.0	4	0.9	2	2056	Artix-7
Bisheh-Niasar [13]	474	115	4.12	737	3.2	290	1.2	6	3.9	4	728	Artix-7
This work	**64**	**265**	**0.24**	**3918**	**1**	**4292**	**1**	**26**	**1**	**0**	**12422**	**Artix-7**

[a] Area-time product (ATP) is normalized to this work, smaller value is better.

TABLE 2. Comparison of the proposed polynomial multipliers with other approaches.

Item	Wang [24]	TW	Feng [14]	TW	Yaman [11]	TW
Device[b]	A-7		S-6	S-7	A-7	
n	1024		512		256	
q size	14	14	23	14	12	12
LUT	944	23261	18K	5.7K	9508	9211
FF	467	14901	slices[c]	slices	2684	9810
DSP	3	76	128	68	16	60
BRAM	3	1	2.5	1	35	1
CCs	11455	846	412	454	256	246
Freq. (MHz)	141	257	233.1	261	172	265
Latency (μs)	81.24	3.29	1.77	1.74	1.49	0.93
Norm. Eff_LUT	1	1	1[d]	2.0	1	1.7
Norm. Eff_FF	1	0.8			1	0.4
Norm. Eff_DSP	1	1	1	1.2	1	0.4
Norm. Eff_BRAM	1	74	1	1.5	1	56

TW: This work.
[b] : S-6: Spartan-6; S-7: Spartan-7; A-7: Artix-7.
[c] : Each Spartan slice contains four LUTs and eight FFs.
[d] : Efficiency metric is calculated for utilized Slices.

TABLE 3. FPGA resource breakdown of the proposed polynomial multipliers on Zynq-7000 FPGA platform.

Module	LUT (%)	FF (%)	DSP (%)	BRAM
$n = 1024; q = 12289$				
NTT_1	10765 (46)	7146 (48)	36 (47)	0
NTT_2	11482 (49)	7146 (48)	36 (47)	0
⌊ BU1	227	165	1	
⌊ BU2	254	192	1	
⌊ MR	147	114	0	
Four-path ModMult	601 (3)	462 (3)	4 (6)	0
Top Control Unit and Data Memory	415 (2)	147 (1)	0	1
$n = 512; q = 12289$				
NTT_1	8713 (46)	6538 (48)	32 (47)	0
NTT_2	9405 (49)	6538 (48)	32 (47)	0
Four-path ModMult	601 (3)	462 (3)	4 (6)	0
Top Control Unit and Data Memory	384 (2)	139 (1)	0	1
$n = 256; q = 3329$				
NTT_1	3918 (42)	4292 (44)	26 (43)	0
NTT_2	4217 (46)	4292 (44)	26 (43)	0
⌊ BU1	153	151	1	
⌊ BU2	186	172	1	
⌊ MR	81	112	0	
Four-path ModMult	749 (8)	1103 (11)	8 (14)	0
Top Control Unit and Data Memory	327 (4)	123 (1)	0	1

one FPGA hardware unit can deliver and defined as follows,

$$Efficiency = \frac{Throughput}{Utilized\ resource}. \quad (5)$$

The efficiency values of utilized LUTs, FFs, DSPs, and BRAMs are calculated by the equation (5), normalized, and denoted as Eff_LUT, Eff_FF, Eff_DSP, and Eff_BRAM, respectively. For $n = 1024$, Wang *et al.* [24] proposed a hardware accelerator for shared polynomial multiplication, which traded parameterization and used one configurable BU to reduce the hardware complexity. However, their polynomial multiplier required a large number of CCs (i.e., 11455). The proposed polynomial multiplier outperforms [24] approximately 25× speed-up with much higher BRAM efficiency value. For $n = 512$, Feng *et al.* [14] implemented a high-speed polynomial multiplier on the Spartan-6 FPGA platform. The fifth column of this table shows that the proposed polynomial multiplier consumes 5.7K slices

(19105 LUTs and 13677 FFs) with higher efficiency values than that of [14]. Differ from Spartan-6, Spartan-7 has some extended features of the 7 series family such as in DSP and BRAM. However, our register-transfer logic design only used basic logic elements except the 36Kb BRAM, which operated in simple dual-port mode and was better suited to the Data Memory structure than the 18Kb BRAM included in Spartan-6. For $n = 256$, the unified hardware architecture in [11] required 256 CCs to complete the polynomial multiplication. Our polynomial multiplier runs faster, has better LUT and BRAM, but worse FF and DSP efficiency metrics compared with [11]. However, the key generation, encryption, and decryption processes in Kyber [2] show that the NTT, PWM, and INTT are performed on-demand. In which, only one or even no NTT computation are required for the two input polynomials right before the PWM. It means that the proposed approach can perform the polynomial multiplication in [2] more efficiently in a highly pipelined manner. Regarding memory usage, the proposed approach allocates one 36Kb BRAM unit in simple dual-port mode for the Data Memory unit (512 addresses of 72-bit). The Data Memory only consumes 64, 128, and 256 addresses of 36Kb BRAM unit for 256, 512, and 1024-point polynomials, respectively.

Table 3 shows the utilized FPGA resource breakdown of the proposed polynomial multipliers. Except for BRAM, the NTT modules occupy most of the hardware resources, and the NTT_2 consumes more LUTs for configurable function than the NTT_1. For $n = 256$, the Four-path ModMult unit consumes more LUTs, FFs, and DSPs for the specific PWM than the classic PWM in cases of $n = 1024$ and 512. Percentage values indicate the utilized resource proportion of respective modules in polynomial multipliers. Additionally, we report the hardware consumption of submodules such as configurable BUs in Fig. 4 and MR units in Fig. 5. Please notice that the BU1 implementation result is reported with one FIFO register. The MR operation of two modulo primes can share some of bit-shift operations.

Thus, the proposed NTT architecture can achieve superior throughput with comparable efficiency compared to previous approaches. Although different parameter sets were implemented and compared, the proposed polynomial multiplier is primarily directed towards supporting potential LBC schemes for the third round NIST finalists.

V. CONCLUSION

The paper proposed an efficient mixed-radix MDF NTT architecture preferable for high-performance large-scale data cryptoprocessors. Flexible design adapted the NTT architecture to various parameter sets (n, q) and the reasonable choice of radix values helped achieve high performance. The proposed configurable NTT/INTT architecture offers a versatile tool to effectively perform expensive polynomial multiplication in LBC schemes.

For future works, the proposed NTT architectures could be improved with various levels of parallelism in stages and customized for high degree large modulus polynomial constructors. Coming study is applying the proposed configurable NTT/INTT architecture to accelerate the NIST lattice-based PQC finalists, particularly on co-designed software and hardware platforms.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their constructive and insightful comments.

REFERENCES

[1] G. Alagic, J. Alperin-Sheriff, D. Apon, D. Cooper, Q. Dang, J. Kelsey, Y.-K. Liu, C. Miller, D. Moody, R. Peralta, and R. Perlner, "Status report on the second round of the NIST post-quantum cryptography standardization process," NIST, Gaithersburg, MD, USA, Tech. Rep. 8309, Jul. 2020. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/ir/2020/NIST.IR.8309.pdf

[2] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehle, "CRYSTALS-kyber: Algorithm specifications and supporting documentation," NIST PQC Round 3 Submission, Gaithersburg, MD, USA, Ver. 3.02, Aug. 2021, pp. 1–43. [Online]. Available: https://csrc.nist.gov/Projects/postquantum-cryptography/round-3-submissions

[3] A. Basso, J. M. B. Mera, J.-P. D'Anvers, S. S. R. A. Karmakar, M. V. Beiren-Donck, and F. Vercauteren, "SABER: Mod-LWR based KEM," NIST PQC Round 3 Submission, Gaithersburg, MD, USA, Ver. 2, Oct. 2020, pp. 1–44.

[4] S. Bai, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehle, "CRYSTALS-dilithium: Algorithm specifications and supporting documentation," NIST PQC Round 3 Submission, Gaithersburg, MD, USA, Ver. 3.1, Feb. 2021, pp. 1–38.

[5] E. Alkim, R. Avanzi, J. Bos, L. Ducas, A. de la Piedra, T. Pöppelmann, P. Schwabe, and D. Stebila, "NewHope: Algorithm specification and supporting documentation," NIST PQC Round 2 Submission, Gaithersburg, MD, USA, Ver. 1.1, Apr. 2020, pp. 1–47.

[6] A. Lohachab, A. Lohachab, and A. Jangra, "A comprehensive survey of prominent cryptographic aspects for securing communication in post-quantum IoT networks," *Internet Things*, vol. 9, Mar. 2020, Art. no. 100174.

[7] T. N. Tan and H. Lee, "High-secure fingerprint authentication system using ring-lwe cryptography," *IEEE Access*, vol. 7, pp. 23379–23387, 2019.

[8] P. Duong-Ngoc, T. N. Tan, and H. Lee, "Efficient NewHope cryptography based facial security system on a GPU," *IEEE Access*, vol. 8, pp. 108158–108168, 2020.

[9] Y. Xing and S. Li, "An efficient implementation of the NewHope key exchange on FPGAs," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 3, pp. 866–878, Mar. 2020.

[10] N. Zhang, B. Yang, C. Chen, S. Yin, S. Wei, and L. Liu, "Highly efficient architecture of NewHope-NIST on FPGA using low-complexity NTT/INTT," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2020, pp. 49–72, Mar. 2020.

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

[12] M. Bisheh-Niasar, R. Azarderakhsh, and M. Mozaffari-Kermani, "High-speed NTT-based polynomial multiplication accelerator for post-quantum cryptography," in *Proc. IEEE 28th Symp. Comput. Arithmetic (ARITH)*, Jun. 2021, p. 563.

[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, doi: 10.1109/TCSI.2021.3106639.

[14] X. Feng, S. Li, and S. Xu, "RLWE-oriented high-speed polynomial multiplier utilizing multi-lane Stockham NTT algorithm," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 3, pp. 556–559, Mar. 2020.

[15] A. C. Mert, E. Karabulut, E. Ozturk, E. Savas, and A. Aysu, "An extensive study of flexible design methods for the number theoretic transform," *IEEE Trans. Comput.*, early access, Aug. 19, 2020, doi: 10.1109/TC.2020.3017930.

[16] C. P. Rentería-Mejía and J. Velasco-Medina, "High-throughput ring-LWE cryptoprocessors," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 8, pp. 2332–2345, Aug. 2017.
[17] T. N. Tan and H. Lee, "Efficient-scheduling parallel multiplier-based ring-LWE cryptoprocessors," *Electron.*, vol. 8, 413, pp. 1–13, Apr. 2019.
[18] P. Duong-Ngoc, Y. Kim, and H. Lee, "Efficient k-parallel pipelined NTT architecture for post quantum cryptography," in *Proc. Int. SoC Design Conf. (ISOCC)*, Yeosu, South Korea, Oct. 2020, pp. 212–213.
[19] NIST. (Aug. 2016). *Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process*. [Online]. Available: https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/Call-for-Proposals
[20] P. Longa and M. Naehrig, "Speeding up the number theoretic transform for faster ideal lattice-based cryptography," in *Cryptology and Network Security* (Lecture Notes in Computer Science), vol. 10052. Cham, Switzerland: Springer, Nov. 2016, pp. 124–139.
[21] P. Thomas, T. Oder, and G. Tim, "High-performance ideal lattice-based cryptography on 8-bit ATxmega microcontrollers," in *Proc. LATINCRYPT* in Lecture Notes in Computer Science, vol. 9230. Cham, Switzerland: Springer, Aug. 2015, pp. 346–365.
[22] U. Banerjee, T. S. Ukyab, and A. P. Chandrakasan, "Sapphire: A configurable crypto-processor for post-quantum lattice-based protocols," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 4, pp. 17–61, Aug. 2019.
[23] Y. Xing and S. Li, "A compact hardware implementation of CCA-secure key exchange mechanism CRYSTALS-KYBER on FPGA," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2021, no. 2, pp. 328–356, Feb. 2021.
[24] W. Wang, S. Tian, B. Jungk, N. Bindel, P. Longa, and J. Szefer, "Parameterized hardware accelerators for lattice-based cryptography and their application to the HW/SW co-design of qTESLA," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2020, no. 3, pp. 269–306, Jun. 2020.

PHAP DUONG-NGOC (Member, IEEE) received the B.S. degree in electronic and telecommunication engineering from the Danang University of Technology, in 2009, and the M.S. degree in electronic engineering from Danang University, Vietnam, in 2015. He is currently pursuing the Ph.D. degree in information and communication engineering with Inha University. His research interests include algorithms and architectures for post-quantum cryptography and homomorphic encryption.

HANHO LEE (Senior Member, IEEE) received the M.Sc. and Ph.D. degrees in electrical and computer engineering from the University of Minnesota, Minneapolis, USA, in 1996 and 2000, respectively. He was a member of the technical staff at Lucent Technologies (Bell Labs Innovations), Allentown, from April 2000 to August 2002, and an Assistant Professor with the Department of Electrical and Computer Engineering, University of Connecticut, USA, from August 2002 to August 2004. He was a Visiting Scholar at Bell Labs, Alcatel-Lucent, Murray Hill, NJ, USA, from August 2010 to August 2011. He has been with the Department of Information and Communication Engineering, Inha University, since August 2004, where he is currently a Professor. His research interests include algorithms and architectures for cryptography, forward error correction coding, and digital signal processing.