High-Speed NTT-based Polynomial Multiplication Accelerator for CRYSTALS-Kyber Post-Quantum Cryptography

Mojtaba Bisheh-Niasar[1], Reza Azarderakhsh[1,2], and Mehran Mozaffari-Kermani[3]

[1] Department of Computer and Electrical Engineering and Computer Science,
Florida Atlantic University, FL, USA
{mbishehniasa2019,razarderakhsh}@fau.edu
[2] PQSecure Technologies, LLC, Boca Raton, FL , USA
[3] Department of Computer Science and Engineering, University of South Florida,
FL, USA
mehran2@usf.edu

Abstract. This paper demonstrates an architecture for accelerating the polynomial multiplication using number theoretic transform (NTT). Kyber is one of the finalists in the third round of the NIST post-quantum cryptography standardization process. Simultaneously, the performance of NTT execution is its main challenge, requiring large memory and complex memory access pattern. In this paper, an efficient NTT architecture is presented to improve the respective computation time. We propose several optimization strategies for efficiency improvement targeting different performance requirements for various applications. Our NTT architecture, including four butterfly cores, occupies only 798 LUTs and 715 FFs on a small Artix-7 FPGA, showing more than 44% improvement compared to the best previous work. We also implement a coprocessor architecture for Kyber KEM benefiting from our high-speed NTT core to accomplish three phases of the key exchange in 9, 12, and 19 μs, respectively, operating at 200 MHz.

Keywords: FPGA, hardware architecture, Kyber, lattice-based cryptography, NTT, post-quantum cryptography.

1 Introduction

The security of classical public-key cryptosystems relies on the underlying NP-hard problems like integer factorization, discrete logarithm, and elliptic curve discrete logarithm. However, these problems can be solved when a large-scale quantum computer is build using quantum algorithms such as Shor's algorithm [1]. Hence, the National Institute of Standards and Technology (NIST) started a post-quantum cryptography standardization process in 2016, noting that in round-3 of this competition, the four key encapsulation mechanisms (KEM) finalists, i.e., Classic-McEliece, Kyber, NTRU, and Saber, were announced in July

2020. Among all promising candidates, lattice-based cryptography is a very attractive alternative, mainly because of offering a good trade-off between security and efficiency.

Kyber KEM [2] is part of the Cryptographic Suite for Algebraic Lattices (CRYSTALS) and shares a common framework with the Dilithium signature scheme [3]. Kyber bases its security on the hardness assumptions over module learning with errors (Module-LWE) and is believed to be quantum-resistant. The main characteristic of Kyber is polynomial multiplication over a polynomial ring as $\mathbb{Z}_{3329}[X]/\left\langle X^{256}+1\right\rangle$, providing a significant increase in efficiency. Hence, the most computationally intensive operation, i.e., matrix-vector and vector-vector multiplication, can be optimized with the fast number-theoretic transform (NTT), which can reduce computational complexity from $O(n^2)$ to roughly $O(n\log n)$. Since the implementation of NTT-based multiplication is still a performance bottleneck in lattice-based cryptography, improving NTT efficiency has recently received significant attention.

Reducing the computational complexity of polynomial multiplication is essential for faster key encapsulation and optimization of the resource utilization of the entire cryptosystem. This acceleration of polynomial multiplication would be challenging for various applications due to their resource constraints, strict performance, and flexibility requirements. However, for a widely-deployed cryptosystem, the overall complexity consisting of the utilized resource and the required latency will have to be minimal to be standardized by NIST [4]. To address these challenges, hardware implementation of the cryptosystem will be critical since it accelerates the core arithmetic operation occupying limited resources.

Overall, there are two possible strategies to deploy hardware accelerators: (i) hardware/software co-design approaches and (ii) pure hardware architectures. Although hardware/software co-design approaches are more flexible and easier to develop compared to pure hardware architectures, they may not lead to the best performance. Most hardware accelerators focus on the FPGA platform to take advantage of its reconfigurability. FPGA can provide an appropriate balance between flexibility and performance, which is especially important for a rapidly evolving field like PQC.

1.1 Related Work

There are prominent works to accelerate polynomial multiplication in the literature. The work of [5] proposed the negative wrapped convolution (NWC) to eliminate the overhead of zero padding in the polynomial multiplication over $\mathbb{Z}_q[X]/\langle X^n+1\rangle$. The authors in [6] introduced low-complexity NTT by merging the pre-processing of NTT into butterfly operations. Furthermore, low-complexity INTT is proposed in [7] to avoid post-processing overhead. Longa *et al.* in [8] proposed the KRED and KRED-2X reduction algorithms to speed up the NTT computation. This work also reduces post-processing computation of INTT at the cost of more memory utilization. Furthermore, employing Cooley-Tukey (CT) and Gentleman-Sande (GS) butterfly configurations reduces bit-reverse operation, which was implemented in [9]. The authors in [10] presented a processor benefiting from polynomial vector structure in the Kyber algorithm to reduce memory access overhead.

A flexible and scalable NTT architecture was presented in [11,12]. Furthermore, the work of [13] implemented a scalable NTT architecture on RISC-V. In [14], a low-power NTT was proposed to reduce the required latency.

Although a compact design of NTT employing only one butterfly core requires few hardware resources, it is too slow to provide high throughput requirements of high-performance applications. The work of [15] employed four butterfly cores for NewHope implementation. However, increasing the number of butterfly cores in unmerged implementations increases memory access overhead. Hence, merging NTT layers was studied in [16] using 2×2 butterfly structure. This design was customized in [17] for NewHope using KRED and KRED-2X reductions in their proposed architecture. The authors in [18,19] used the same architecture for Kyber KEM, employing the high-level synthesis (HLS) approach. Implementing KRED and KRED-2X modular reductions increases the performance in software platforms, while it doubles the occupied resources in hardware. Furthermore, the required memory for the precomputed values is increased to store two sets of constants. Additionally, the authors in [20] implemented 3-layer merged NTT for NewHope by RISC-V ISA features, while they claimed using this method for Kyber cannot improve efficiency. The prior hardware NTT designs have so far been fixed in throughput. Furthermore, since the same butterfly configurations are used for both NTT and INTT, a bit-reverse function is required.

Implementing Ring-LWE has been increased since it offers high-performance and compact architecture compared to both PQC schemes [21,22,23] and even pre-quantum cryptosystems [24,25]. Although many efforts towards the HLS [26] and the hardware/software co-design implementation of PQC accelerators have been made [9,20,27,28], there are merely a few developed pure hardware architectures for Kyber KEM. The first hardware implementation of Kyber is reported in [29], employing an RTL-based methodology providing good performance and smaller area consumption compared to the HLS-based approach. Furthermore, the authors in [30] proposed an architecture of Kyber, which heavily relies on BlockRAM primitives between components. Recently, the work of [31] implemented a compact FPGA-based architecture occupying only 3 BRAMs.

Fig. 1 shows a performance and resource utilization comparison between software, hardware/software, and pure hardware implementations of Kyber. Software benchmarking [32,33,34] reports 60-80% of the overall required cycle for hashing and sampling while hardware/software accelerators can reduce it. However, Keccak latency can be hidden by pure hardware design when it works in a parallel fashion with the NTT core. A wide range of NTT computation (25-90%) has been reported in the literature for the hardware/software approach since different optimization perspectives have been targeted. Based on the aforementioned discussions, implementation gaps are identified in accelerating and compacting the NTT core in pure hardware architecture to reduce the total time and required resources.

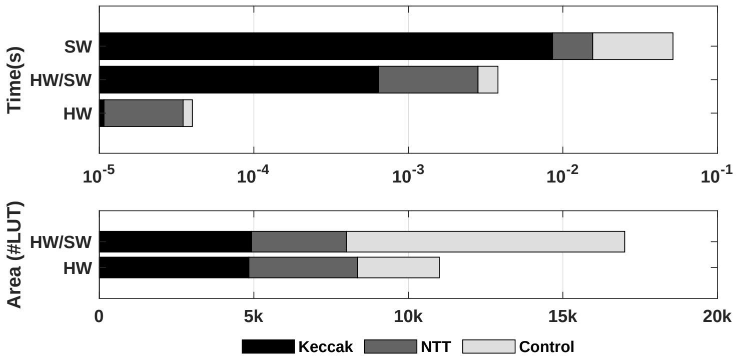

Fig. 1. Performance (in log_{10}) and resource utilization comparison in three different Kyber implementation approaches: software (SW), hardware/software (HW/SW), and hardware (HW). Kyber architecture is breakdown into three main cores, including Keccak (hashing and sampling), NTT (polynomial multiplication), and Control (controller and all other required functions).

1.2 Contributions

Polynomial multiplication computations take a significant portion of Kyber KEM latency on hardware implementation. Therefore, to improve the efficiency of Kyber, one should increase efficiency on the NTT core, providing higher throughput using fewer hardware resources. This paper proposes algorithmic optimizations and hardware optimizations to design an efficient pure hardware architecture of high-speed polynomial multiplication core (PMC) on FPGA to accelerate Kyber KEM. Algorithmic optimizations include modular reduction and efficient NTT computation. The hardware optimizations are achieved by designing a reconfigurable butterfly core (BF), judicious rearrangement of the sequence of the operations to leverage pipelining and parallelism at multiple layers within each unit's implementation.

The contributions and novelties of this paper are as follows:

1. We propose a hardware-friendly modular reduction algorithm, which requires few resources without the additional cost of memory utilization. Reductions are only carried out after multiplications to avoid occupying other resources.
2. We propose an improved reconfigurable hardware architecture for NTT and INTT with highly efficient modular reduction. This reconfigurability supporting both decimation-in-frequency (DIF) and decimation-in-time (DIT) NTT algorithm avoids utilizing additional resources for the same computations while reduces the pre-processing cost of NTT and post-processing cost of INTT. The proposed architecture significantly reduces the overall area and memory consumption with no impact on performance.
3. We implement a parameterized design of the NTT module using VHDL and prototype it on an Artix-7 FPGA. Our NTT core shows an efficiency

improvement by 44% with at least 25% and 80% fewer Slice and BRAM resource utilization.

4. We propose a high-performance coprocessor architecture for lattice-based public-key cryptography with Kyber KEM as a case study. Our result utilizes the proposed high-speed NTT core and outperforms all reported implementations by reducing the total time.

The rest of the paper is organized as follows. In Section 2, we discuss the preliminaries of lattice-based cryptography and the relevant mathematical background based on the Kyber algorithm. In Section 3, our proposed algorithms and architectures are discussed. Furthermore, the details of FPGA implementations of Kyber KEM are provided in Section 4. We discuss our results and compare to the counterparts in Section 5. Finally, we conclude the paper in Section 6.

2 Preliminaries

In this section, Kyber protocols and relevant mathematical background are briefly described.

2.1 The Kyber Protocol

Kyber is an IND-CCA secure KEM [35], including three algorithms, i.e., key generation, encryption, and decryption. In key generation, a matrix $\mathbf{A}$ and a secret key $\mathbf{s}$ are sampled from a uniform and binomial distribution, respectively. Then a public key is computed by multiplication between $\mathbf{A}$ and $\mathbf{s}$ in the NTT domain and adding noise to the product. In encryption, a message m should be added to the product of the public key and a sampled random $\mathbf{r}$ in the normal domain to generate a vector v. Additionally, another polynomial multiplication is performed between $\mathbf{r}$ and uniform distribution to compute matrix $\mathbf{u}$. The encryption output, called ciphertext ct, is composed of compression of $\mathbf{u}$ and v, while the message can then be decrypted by recovering an approximation of v by computing the product of secret key and $\mathbf{u}$.

All polynomials in the Kyber scheme have 256 coefficients over k-dimensional vectors, where $k = 2, 3, 4$ indicates the three different post-quantum security levels. Kyber uses these functions to construct a Chosen Plaintext Attack (CPA) secure public-key encryption scheme. Moreover, a CCA-secure Kyber KEM can be constructed using an adapted Fujisaki-Okamoto transformation [36]. For details, we refer interested readers to [2].

2.2 Polynomial Multiplication

Polynomial multiplication is the bottleneck of lattice-based cryptography, which can either be done using NTT or schoolbook polynomial multiplication algorithm. The former can be exploited to compute polynomial multiplication efficiently over a polynomial ring $\mathbb{Z}_q[X]/\langle X^n + 1\rangle$. The NTT is a generalization

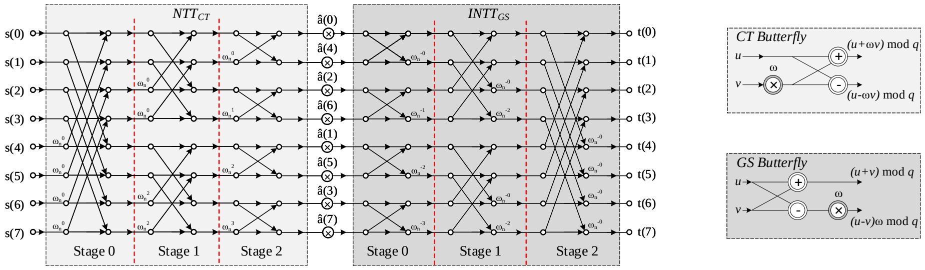

Fig. 2. An 8-point NTT-based polynomial multiplication: (Left) Dataflow graph including CT butterfly-based NTT, point-wise multiplication, and GS butterfly-based INTT. Polynomial $\hat{a}$ is in NTT domain and s and t are in normal domain. (Right) CT and GS butterfly configurations.

of a fast Fourier transform (FFT) defined in a finite field. Let f be a polynomial of degree n, where $f = \sum_{i=0}^{n-1} f_i X^i$ and $f_i \in \mathbb{Z}_q$, and ω_n be n-th primitive root of unity such that $\omega_n^n = 1 \bmod q$. The forward NTT is defined by $\hat{f} = NTT(f)$, such that $\hat{f}_i = \sum_{j=0}^{n-1} f_j \omega_n^{ij} \bmod q$. Furthermore, the inverse NTT is shown by $f = INTT(\hat{f})$, such that $f_i = n^{-1} \sum_{j=0}^{n-1} \hat{f}_j \omega_n^{-ij} \bmod q$. An NTT-based polynomial multiplication between f and g can be performed such that $f.g = \mathrm{INTT}(\mathrm{NTT}(f) \circ \mathrm{NTT}(g))$.

To avoid applying the NTT of length $2n$ with n zero padding of inputs, NWC [5] is proposed at the cost of pre-processing of NTT and post-processing of INTT. Let $\psi = \sqrt{\omega_n}$ be a primitive $2n$-th root of unity. Pre-processing of NTT includes multiplication between the coefficients of the input polynomials and ψ^i, while the post-processing of INTT is multiplication between the coefficients of the output polynomial and ψ^{-i}.

NTT computation can be implemented by CT or GS butterfly. Algorithm 1 presents the NTT computation. The bit-reverse function in line 1 is the bit-wise reversal of the binary representation of the coefficient index. The CT butterfly computation is shown in lines 8-11. Furthermore, performing CT butterfly for NTT and GS for INTT can avoid the bit-reverse permutation [8]. Fig. 2 illustrates an 8-point NTT-based multiplication employing both CT and GS butterfly operations

In order to perform point-wise multiplication in Kyber, we have to compute 128 degree-2 polynomial multiplications such that $(\hat{a}_{j,2i} + \hat{a}_{j,2i+1}X) \cdot (\hat{s}_{2i} + \hat{s}_{2i+1}X) = (\hat{a}_{j,2i}\hat{s}_{2i} + \hat{a}_{j,2i+1}\hat{s}_{2i+1}\omega_n^{2\mathsf{br}_7(i)+1}) + (\hat{a}_{j,2i}\hat{s}_{2i+1} + \hat{a}_{j,2i+1}\hat{s}_{2i})X$, where br_7 is the bit reversal function.

2.3 Modular Reduction

Different modular reductions can be implemented in butterfly core, including Barrett reduction and Montgomery reduction. A variant of Montgomery reduction was introduced by [8], benefiting from a special form of prime $q = k \cdot 2^m + 1$.

Algorithm 1 Iterative In-Place NTT Algorithm Based on Cooley-Tukey Butterfly [37]

```
Input: a polynomial $a(x) \in \mathbb{Z}_q[X]/(X_n + 1)$, $n$-th primitive root of unity $\omega_n \in \mathbb{Z}_q$, $n = 2^l$
Output: $\hat{a}(x) = \mathrm{NTT}_{\omega_n}(a) \in \mathbb{Z}_q[X]/(X_n + 1)$
 1: $\hat{a} \leftarrow$ bit-reverse($a$)
 2: for $(i = 1;\ i < l;\ i++)$ do
 3:   $m = 2^{l-i}$
 4:   $\omega_m \leftarrow \omega_n^{n/m}$
 5:   for $(j = 0;\ j < n;\ j = j + m)$ do
 6:     $\omega \leftarrow 1$
 7:     for $(k = 0;\ k < m/2;\ k++)$ do
 8:       $T \leftarrow \omega \cdot \hat{a}[k + j + m/2] \bmod q$
 9:       $U \leftarrow \hat{a}[k + j]$
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 $\hat{a}(x)$
```

This method includes two functions, i.e., KRED and KRED-2X, which take any integer C and return an integer D such that $D \equiv k \cdot C \bmod q$ and $D \equiv k^2 \cdot C \bmod q$, respectively. However, we can eliminate the extra factor of k^s with $s \in \{1, 2\}$ by replacing $k^{-s} \cdot \omega_n^{ij}$ instead of ω_n^{ij} in line 8 of Algorithm 1. Although these functions do not compute the exact value of $C \bmod q$, they can close the output range to the exact value. In Kyber with $q = 3329$, we have $k = 13$ and $m = 8$. These functions do not need any multiplications in hardware and can be achieved by shifter and adder.

3 Proposed Architecture for High-speed Polynomial Multiplier

3.1 Modular Reduction

Implementing KRED and KRED-2X requires to store $k^{-1} \cdot \omega_n^{ij}$ and $k^{-2} \cdot \omega_n^{ij}$ in ROM. Furthermore, the KRED-2X returns $k^2 \cdot C_0 - k \cdot C_1 + C_2$ where C_0, C_1, and C_2 are the m-bit chunks of input C. Thus, for $k = 13$ it needs 5 shifting and 7 additions to output a 16-bit data. However, it allows output to grow up to 32 bits. Hence, we propose K^2-RED reduction, a modified version of the KRED algorithm, presented in Algorithm 2. It includes two steps of performing KRED, so its output is $k^2 \cdot C \bmod q$. This reduction needs 4 shift and 6 addition operations and keeps output width to 12 bits. Furthermore, we do not need to implement another reduction unit in the butterfly core by implementing this reduction after multiplication, and the required memory is halved. Fig. 3 shows

Algorithm 2 Proposed K^2-RED Reduction Algorithm

```
Input: A binary number C = $(c_{23}, \ldots, c_0)_2$, $k = 13$, $m = 8$, $q = 3329 = k \cdot 2^m + 1$
Output: $C^{''} = k^2 C \bmod q$
    Step 1:
    $C_l = (c_7, \ldots, c_0)_2$
    $C_h = (c_{23}, \ldots, c_8)_2$
    $C^{'} \leftarrow k \cdot C_l - C_h$
    Step 2:
    $C_l^{'} = (c_7^{'}, \ldots, c_0^{'})_2$
    $C_h^{'} = (c_{15}^{'}, \ldots, c_8^{'})_2$
    $C^{''} \leftarrow k \cdot C_l^{'} - C_h^{'}$
    return $C^{''}$
```

the reduction architecture of a 24-bit input using Algorithm 2 to compute a 12-bit output.

3.2 Reconfigurable Butterfly Core

To avoid the bit-reverse cost in polynomial multiplication, two different butterfly configurations, i.e., CT and GS, are required for NTT and INTT, respectively. Hence, a reconfigurable butterfly core is proposed to support both CT and GS operations and reduce required hardware resources. Furthermore, we implement a 2 × 2 butterfly core to merge two layers of NTT/INTT and perform two butterfly operations in each layer.

The proposed architecture for PMC is depicted in Fig. 3 employing four butterfly cores. Each butterfly core includes a multiplication, a modular reduction, an addition, and a subtraction, while there are also some registers to balance the pipeline latency in different configurations. The signal *mode* chooses between NTT and INTT operations. It also supports point-wise multiplication, polynomial addition, and polynomial subtraction employing an additional control logic which is not shown in Fig. 3 for brevity. When *mode* is set to 0, the butterfly works in CT configuration in the NTT computation and computes $u + v\omega$ and $u - v\omega$. The butterfly cores are reconfigured when $mode = 1$ for GS in INTT operation, while its output is manipulated compared to standard GS to reduce required memory. The proposed architecture supports both even or odd numbers of layers employing pipeline stages. Hence, to support an odd number of layers, *mode* is set to 2 for the first butterfly row in the last layer of computation to only pass the data. The proposed algorithm to perform NTT is shown in Algorithm 3 for even layers.

In each cycle, four coefficients are read from NTT RAM to fed cores, and their outputs are buffered in four serial-in, parallel-out shift registers with different lengths. The results are written back to the NTT RAM sequentially. The address and data flow of NTT RAM for read and write operation in every clock cycle are given in Fig. 4 for $n = 128$. After 4 cycles, the first buffer is full, and 4 coefficients can be stored in the RAM. The same scenario is performed after one cycle for the

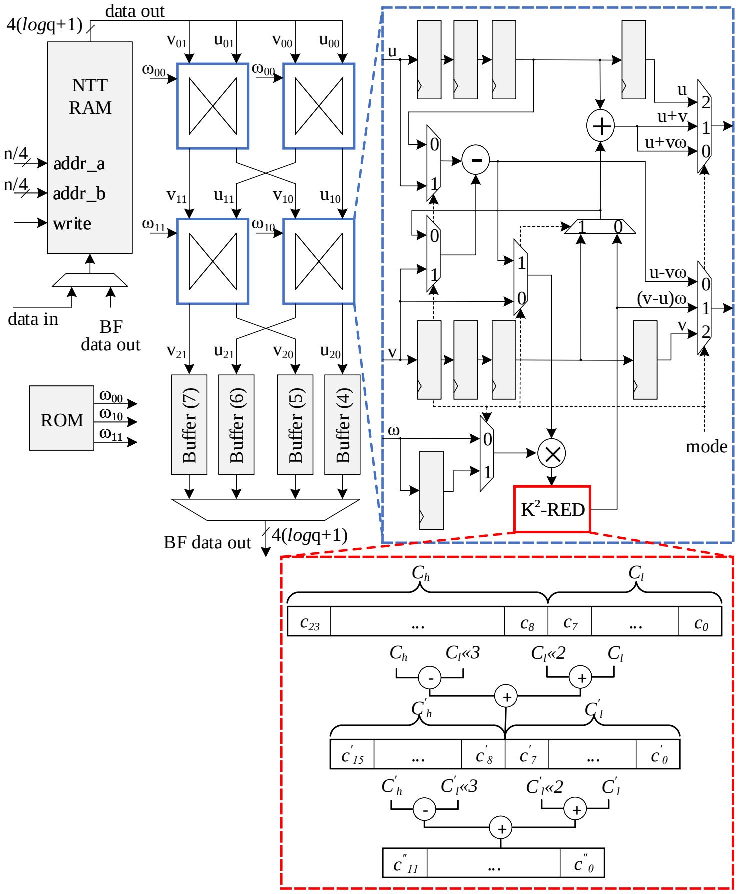

Fig. 3. Proposed polynomial multiplication architecture employing 2×2 reconfigurable butterfly cores and K^2-RED reduction

second and then for the third and fourth buffer, and its first 4-coefficients will be stored. Each round of NTT includes $\frac{n}{4}$ reading and storing while there are fully pipelined to increase throughput. The pipeline latency between read and write sequences consists of 2 cycles for reading from RAM, 8 cycles for two butterfly operations, and 4 cycles for buffering the results in registers. Furthermore, to avoid any memory conflict, we consider 6 idle cycles between each round.

The required twiddle factors for NTT are stored in a ROM. Based on the symmetry property of twiddle factors in NTT and INTT, i.e., ω_n^i and ω_n^{-i} respectively, we have $\omega_n^{-i} = -\omega_n^{n-i}$. Hence, to reduce the required memory, we can use NTT twiddle factors for INTT by (i) reversing the order of reading ROM, and (ii) computing $v-u$ instead of $u-v$ in GS configuration. Our proposed architecture can perform NTT and INTT operations in around $\frac{n}{8}\log n$ and $\frac{n}{8}(\log n+1)$ cycles for even and odd number of layers, respectively.

Algorithm 3 Proposed NTT Algorithm Based on Cooley-Tukey Butterfly

Input: a polynomial $a(x) \in \mathbb{Z}_q[X]/\langle X_n + 1\rangle$, n-th primitive root of unity $\omega_n \in \mathbb{Z}_q$, $n = 2^l$

Output: $a(x) = \mathrm{NTT}_{\omega_n}(a) \in \mathbb{Z}_q[X]/\langle X_n + 1\rangle$

```
for ($s = 0$, $s < log(n)$, $s = s + 2$) do
    $m = 2^s$
    $t = t \gg 2$
    for ($i = 0$, to $i < m$, $i + +$) do
        for ($j = 4i \cdot t$, $j < 4i \cdot t + t$, $j + +$) do
            $u_{00} \leftarrow a_j$, $v_{00} \leftarrow a_{j+t}$, $u_{01} \leftarrow a_{j+2t}$, $v_{01} \leftarrow a_{j+3t}$
            $\omega_{00} \leftarrow \psi_{k-2}[m + i]$
            $(u_{10}, u_{11}) \leftarrow BF\_CT(u_{00}, v_{00}, \omega_{00})$
            $(v_{10}, v_{11}) \leftarrow BF\_CT(u_{01}, v_{01}, \omega_{00})$
            $\omega_{10} \leftarrow \psi_{k-2}[2 \times (m + i)]$, $\omega_{11} \leftarrow \psi_{k-2}[2 \times (m + i) + 1]$
            $(u_{20}, u_{21}) \leftarrow BF\_CT(u_{10}, v_{10}, \omega_{10})$
            $(v_{20}, v_{21}) \leftarrow BF\_CT(u_{11}, v_{11}, \omega_{11})$
            $a_j \leftarrow u_{20}, a_{j+t} \leftarrow v_{20}, a_{j+2t} \leftarrow u_{21}, a_{j+3t} \leftarrow v_{21}$
        end for
    end for
end for
return $a(x)$
```

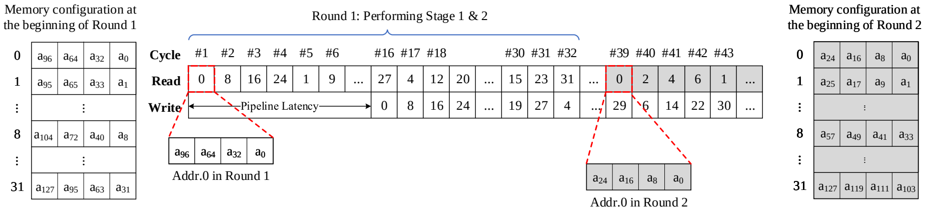
Memory configuration at the beginning of Round 1
0 | a_{96} | a_{64} | a_{32} | a_0
1 | a_{95} | a_{65} | a_{33} | a_1
⋮ | ⋮
8 | a_{104} | a_{72} | a_{40} | a_8
⋮ | ⋮
31 | a_{127} | a_{95} | a_{63} | a_{31}
Round 1: Performing Stage 1 & 2
Cycle #1 #2 #3 #4 #5 #6 #16 #17 #18 #30 #31 #32 #39 #40 #41 #42 #43
Read 0 8 16 24 1 9 ... 27 4 12 20 ... 15 23 31 ... 0 2 4 6 1 ...
Write Pipeline Latency 0 8 16 24 ... 19 27 4 ... 29 6 14 22 30 ...
a_{96} a_{64} a_{32} a_0
Addr.0 in Round 1
a_{24} a_{16} a_8 a_0
Addr.0 in Round 2
Memory configuration at the beginning of Round 2
0 | a_{24} | a_{16} | a_8 | a_0
1 | a_{25} | a_{17} | a_9 | a_1
⋮ | ⋮
8 | a_{57} | a_{49} | a_{41} | a_{33}
⋮ | ⋮
31 | a_{127} | a_{119} | a_{111} | a_{103}

Fig. 4. Memory Address and Data flow when NTT operation is performed.

3.3 Area/Performance Trade-offs

The main goal of the proposed architecture is to achieve high-speed computation employing small area requirements. However, we can target different area/performance trade-offs by increasing the number of PMC, taking advantage of polynomial vector structure in the Kyber algorithm. Since NTT/INTT can be computed for odd and even coefficients of each polynomial in Kyber separately, two PMC can be implemented for each polynomial vector. Hence, for Kyber-512 having 2 polynomial vectors, increasing the number of implemented PMC from 1 to 2 or 4 can drastically reduce to a half or a quarter of NTT/INTT latency.

Nevertheless, implementing more PMC needs more bandwidth for feeding the butterfly cores and storing their results. On the other hand, due to the data width limitation for BRAM, one BRAM cannot support two PMCs. Thus, the number of utilized BRAM should be matched with PMC to provide the required bandwidth by implementing more BRAMs in parallel.

Table 1. Implementation results for different modular reduction algorithms

Reduction	CPD	Area				Output
Algorithm	[ns]	#LUTs	#FFs	#Slices	#DSPs	Width
Barrett Reduction	1.34	59	31	26	2	12
Montgomery [19]	2.10	391	382	91	1	12[1]
KRED [19]	1.99	80	47	31	0	16[1]
K^2-RED	0.91	54	30	18	0	12

[1] Our estimation by re-implementing this work.

4 Architecture of CRYSTAL-Kyber

The proposed highly optimized architecture for Kyber coprosessor can compute all the operations described in the Kyber protocol. It includes a PMC, Keccak, binomial sampler, rejection sampler, and compress/decompress units. The architecture of Kyber is designed to perform in constant time.

The Keccak used in SHA3 standard is Keccak-f[1600], which performs four functions, including SHA3-256, SHA3-512, SHAKE-128, and SHAKE-256 during KEM. To design a high-performance architecture, we modify the high-speed core implementation of the Keccak provided by [38]. It requires 24 clock cycles to execute 24 rounds of the Keccak sponge function computation. We also develop a dedicated SIPO and PISO for interfacing with this core in its input and output, respectively. The SIPO takes data in 64-bit width and delivers 1344-bit data to the Keccak core, while the PISO takes 1344-bit data from the core and divides it into 21 chunks of 64-bit width.

Since CT configuration is used in NTT, we assume that the input polynomials are in normal order, while the public and secret keys are in bit-reverse order. Hence, the point-wise multiplication works in bit-reverse order in the NTT domain, and the results are transformed back to the normal domain with normal order employing GS configuration.

In order to reduce the total cycle, operations are performed in a parallel fashion. Hence, the latency of samplers can be entirely absorbed by the Keccak core. To accelerate the KEM computation, we duplicate PMC to maximize the polynomial multiplication speed, while NTT/INTT is independently performed for odd and even coefficients.

5 Implementation Results And Comparisons

Our proposed architecture is synthesized with Xilinx Vivado 2019.2 and implemented on a Xilinx Artix XC7A100T-3 FPGA device which is recommended by NIST.

5.1 Implementation Results of NTT Core

Table 1 reports implementation results for different alternative reduction algorithms for $q = 3,329$. As one can see, our proposed K^2-RED algorithm is more

Table 2. Implementation results for different NTT implementation on FPGA

Work	Butterfly	NTT/INTT	Freq	Time	Area				Speedup	A×C	A×T
		[CCs]	[MHz]	[μs]	#LUTs	#FFs	#DSPs	#BRAM	Ratio		
$n = 1,024$, $q = 12,289$											
[7][A]	2	2,825[1]/2,825[1]	244	11.58	847	375	2	6	1.70	2.4 (45.8%)	9.8 (44.9%)
[15][Z]	4	2,688[2]/2,688[2]	153	5.52	4823	2901	8	0	2.58	13.0 (90.0%)	84.7 (93.6%)
[12][V]	32	200/-	125	1.60	17,188	-	96	48	0.24	3.4 (61.8%)	27.5 (80.4%)
[17][Z]	2×2	2,616/-	150	17.44	2832	1381	8	10	2.57	7.4 (82.4%)	49.4 (89.1%)
[18][Z]	2×2	2,032/-	188	10.81	898	1117	4	10	1.59	1.8 (27.8%)	9.7 (44.3%)
This Work[A]	**2×2**	**1,591/1,591**	**234**	**6.80**	**798**	**715**	**4**	**2**	**1.00**	**1.3**	**5.4**
$n = 256$, $q = 3,329$											
[27][Z]	2	1,935/1,930	-	-	2908	170	9	0	5.97	5.6 (94.6%)	25.3 (95.3%)
[13][V]	1	43,756/-	-	-	417	462	0	0	135.05	18.2 (98.4%)	82.2 (98.5%)
[20][A]	1	6,868/6,367	59	116.41	-	-	-	-	79.73	-	-
[30][A]	2	1,834/-	155	11.83	-	-	-	-	8.10	-	-
[31][A]	2	512/576	161	3.18	1,737	1,167	2	3	2.18	0.9 (66.7%)	5.5 (78.2%)
This Work[A]	**2×2**	**324/324**	**222**	**1.46**	**801**	**717**	**4**	**2**	**1.00**	**0.3**	**1.2**

[A] Implemented on Artix-7 Platform.

[V] Implemented on Virtex-7 Platform.

[Z] Implemented on Zynq-7000 Platform.

[1] This number is obtained by adding the reported cycles for the butterfly operations (i.e., 2569 cycles) with $n/4 = 256$ cycles for the scramble function.

[2] This number is obtained by adding the reported butterfly cycles (i.e., 1280 cycles) with 1280 and 128 cycles for the scramble function and pre/post-processing.

compact compared to other algorithms and maintains the output of 12 bits to reduce required memory. It also requires half of precomputed twiddle factors compared with KRED since the latter needs storing $k^{-1} \cdot \omega_n^{ij}$ and $k^{-2} \cdot \omega_n^{ij}$ in ROM for reduction.

Table 2 reports area and time specifications for our PMC core in NTT and INTT mode. Other state-of-the-art NTT designs with the merged-layer NTT structure are also listed. Additionally, we report the results for both Kyber with $q = 3,329$, $n = 256$, and NewHope with $q = 12,289$, $n = 1024$ to show the superiority of the proposed architecture in different schemes. For comparison, $A\times T$ are reported, where A and T are the utilized LUT and time in μs, respectively. It should be noted that we assume the same operating frequency in computing $A \times T$ as our architecture for the works which do not report frequency. An operating frequency in a limited range is mostly considered to reduce the required power. Thus, $A \times C$ can be computed for a fair comparison, where C is the required clock cycles.

The results show our proposed architecture is the fastest and smallest architecture for $n = 1024$. Although the work of [17] and [18] implemented 2×2 butterfly structure, they use the KRED algorithm over a fixed butterfly configuration. Nonetheless, our proposed reduction algorithm reduces required resources, especially in terms of occupied BRAM, and increases the maximum operating frequency. Furthermore, employing reconfigurable PMC eliminates the bit-reverse function and the pre-processing and post-processing cost. For instance, [17] and [18] need 1,330 and 1,324 cycles for only butterfly operations, respectively, while ours requires 1,320 cycles. In [17], the reduction unit is implemented by DSP block, which results in increasing the number of utilized DSP two times that of ours. Our architecture approximately improves 90% $A \times T$ and reduces 2.57×

the total time for NTT computation comapred to [17]. Although [18] implements the reduction unit without DSP block, this design needs larger area and more cycles. Hence, our proposed design achieves 44% $A \times T$ improvement and 1.59× speedup compared to [18].

The work of [7] occupies two butterfly cores and a highly optimized reduction hardware tailored only for the special value. However, this approach requires more BRAM and LUT to implement a low-complexity NTT utilizing 2 DSPs. As a result, our architecture achieves a speedup factor of 1.70× and improves $A \times T$ by almost 45%.

Our results for Kyber parameters show a significant improvement requiring only 1.46 μs. Since Kyber parameters have been changed during round 2 of the NIST competition, we only list previous works implementing Kyber v-3 parameters for a fair comparison. The work in [27] optimized an NTT core based on hardware/software approach over RISC-V architecture, while it works at 45 MHz on the ASIC platform. If this design runs at the same frequency as ours, its $A \times T$ and total time are 21× and 5.97× greater than our proposed design. The works in [13] and [20] also presented an NTT architecture over RISC-V, which requires considerably greater cycle count, while our optimized design achieves 135.05× and 79.73× speedup, respectively. The FPGA-based design was proposed in [30] employing Montgomery reduction. The required hardware resources for the NTT core were not reported; however, our design reduces the required cycles achieving a speedup factor of 8.1. In [31], two butterfly cores for even and odd coefficients are used employing 2 DSPs at the cost of utilizing 2.17× and 1.63× more LUT and FF. Our result shows 2.18× faster computing and 78.2% $A \times T$ improvement compared to [31].

5.2 Implementation Results of CRYSTAL-Kyber

Table 3 lists the detailed resource consumption, performance results (frequency, required cycles, and execution time), and hardware efficiency of Kyber coprocessor designs in terms of $A \times T$ for all NIST security levels. The total time is the required time for a key encapsulation and a key decapsulation (Encaps + Decaps), as the key generation can be done offline. We utilize 2, 3, and 4 PMCs in our proposed architecture for security levels 1, 3, and 5, respectively. As one can see, our design requires 10,502 LUTs, 9,859 FFs, 8 DSPs, and 13 BRAMs for NIST security level 1. It also runs at 200 MHz and performs the whole Kyber protocol in almost 31 μs.

There are several hardware/software implementations targeting Kyber KEM in the literature. However, a direct comparison is not possible between the listed hardware implementations due to the varying techniques of different FPGA generations, targeting different optimization goals, and using different design methodologies. The work in [9] implemented a configurable coprocessor based on a RISC-V architecture that can be used for multiple lattice-based schemes including Kyber. Its architecture performs almost 91 KEM per second for Kyber-512, which is 353× slower than our design. Our proposed design also achieves

Table 3. FPGA Implementation results and comparison with state-of-the-art

Work	Area					KeyGen/Encaps/Decaps	Freq	Total Time	$A \times T$
	#LUTs	#FFs	#Slices	#DSPs	#BRAMs	[CCs]	[MHz]	[μs]	
Kyber-512									
[26]^{V,1}	1,977,896	194,126	NA	0	0	-/31,669/43,018	67	1,115	2,214.2 (99.9%)
[9]^{A}	14,975	2,539	4,173	11	14	74,519/131,698/142,309	25	10,960	164.4 (99.8%)
[27]^{Z}	23,947	10,847	NA	21	32	150,106/193,076/204,843	-	-	47.6 (99.3%)
[20]^{A}	1,842	1,634	NA	5	34	710,000/971,000/870,000	59	31,203	57.5 (99.4%)
[30]^{A,1}	88,901	NA	141,825	354	202	-/49,015/68,815	155	760	67.8 (99.5%)
[31]^{A}	7,412	4,644	2,126	2	3	3,768/5,079/6,668	161	73	0.54 (34.0%)
[29]^{A}	11,864	10,348	3,989	8	15	-/3,025/4,395	210	35	0.42 (21.4%)
This work^{A}	**10,502**	**9,859**	**3,547**	**8**	**13**	**1,882/2,446/3,754**	**200**	**31**	**0.33**
Kyber-768									
[9]^{A}	14,975	2,539	4,173	11	14	111,525/177,540/190,579	25	14,725	220.5 (99.8%)
[27]^{Z}	23,947	10,847	NA	21	32	273,370/325,888/340,418	-	-	79.8 (99.4%)
[30]^{A,1}	110,260	NA	167,293	292	202	-/77,481/102,113	155	1,159	127.7 (99.6%)
[31]^{A}	7,412	4,644	2,126	2	3	6,316/7,925/10,049	161	112	0.83 (43.4%)
[29]^{A}	11,884	10,380	3,984	8	15	-/4,065/5,555	210	46	0.54 (13.0%)
This work^{A}	**11,783**	**10,424**	**3,952**	**12**	**14**	**2,667/3,251/4,805**	**200**	**40**	**0.47**
Kyber-1024									
[9]^{A}	14,975	2,539	4,173	11	14	148,547/223,469/240,977	25	18,578	278.2 (99.7%)
[27]^{Z}	23,947	10,847	NA	21	32	349,673/405,477/424,682	-	-	99.4 (99.2%)
[20]^{A}	1,842	1,634	NA	5	34	2,203,000/2,619,000/2,429,000	59	85,559	157.6 (99.5%)
[30]^{V,1}	132,918	NA	172,489	548	202	-/107,054/135,553	192	1,264	167.9 (99.6%)
[31]^{A}	7,412	4,644	2,126	2	3	9,380/11,321/13,908	161	157	1.16 (35.3%)
[29]^{A}	12,183	12,441	4,511	8	15	-/5,785/7,395	210	63	0.76 (1.3%)
This work^{A}	**13,347**	**11,639**	**4,585**	**16**	**16**	**3,459/4,122/6,257**	**185**	**56**	**0.75**

^{A}Implemented on Artix-7 Platform.

^{V}Implemented on Virtex-7 Platform.

^{Z}Implemented on Zynq-7000 Platform.

^{1}Different architectures for Encaps and Decaps are used.

99.8% improvements in terms of $A \times T$. In [27], another RISC-V-based architecture was proposed to accelerate NTT-based schemes. This design requires 64× more cycles for encapsulation and decapsulation while consuming 2.3×, 1.1×, 2.6×, and 2.1× more LUTs, FFs, DSPs, and BRAMs, respectively. Additionally, [20] proposed a RISC-V design to accelerate Kyber KEM employing customized instructions. Although the design of [20] is lightweight, its required latency is significantly greater than ours. Thus, our hardware implementation of Kyber is around 1,000 times faster and 180 times more efficient than their hardware/software implementation. An HLS evaluation was proposed in [26] for Kyber-512 employing different implementations for encapsulation and decapsulation. However, this approach comes at a considerably far larger area consumption. Hence, our design achieves almost 7,000 times better $A \times T$ compared to HLS-based implementation.

In [30], a pure hardware architecture was proposed to reuse the resources. In this design, several BRAMs are implemented between modules to facilitate their interface, which requires 202 BRAMs. Nevertheless, our design achieves 24.5× faster KEM and improves 99.5% $A \times T$ while occupying 8.4×, 44.2×, and 13.5× fewer LUTs, DSPs, and BRAMs compared to [30], respectively. The high-speed implementation of Kyber was reported in [29] for two different platforms, i.e., Artix-7 and Virtex-7. In security level 1, our proposed architecture reduces 11.4% of total time and improves 21.4% $A \times T$ on the same platform. Besides,

Table 4. Comparison with other PQC schemes in NIST security level 1.

Protocol	Platform	Area					Freq	Time
		#LUTs	#FFs	#Slices	#DSPs	#BRAMs	[MHz]	[us]
SIKEp434 [22]	Virtex-7	12,818	18,271	5,527	195	32	249.6	8,800
Frodo-640 [39]	Artix-7	6,881	5,081	1,947	16	12.5	149	2,621
LightSaber [40]	UltraScale+	23,686	9,805	NA	0	2	150	60
Kyber-512 [This work]	**Artix-7**	**10,502**	**6,859**	**3,547**	**8**	**15**	**200**	**31**

our design reduces required cycles by 16% and 21% in security levels 3 and 5 by implementing parallel PMCs to accelerate NTT computation. Moreover, our design has 2.35× and 34% better time and $A \times T$, respectively, compared to compact design in [31] in security level 1, while ours utilizes 1.4×, 2.1×, 4×, 4.3× more LUTs, FFs, DSPs, and BRAMs, respectively.

Table 4 lists other PQC scheme results implemented on the FPGA platform for NIST security level 1. Elkhatib *et al.* in [22] implemented a supersingular isogeny-based KEM performed in 8.8 ms. Howe *et al.* [39] presented a flexible FrodoKEM architecture that performs 825 and 710 encapsulations and decapsulation. The work of [40] proposed an instruction-set coprocessor for Saber, which can be extended for LightSaber and FireSaber.

The experimental result shows that taking advantage of the proposed PMC to implement lattice-based KEM schemes as full-hardware architecture results in high-speed and efficient design. For Kyber KEM, our coprocessor architecture outperforms all the reported implementations in the literature. The efficiency of our proposed implementation already has performance levels comparable to or even significantly better than pre-quantum algorithms [41,42].

6 Conclusion

This paper proposed a high-performance and efficient architecture for NTT-based polynomial multiplication and lattice-based public-key cryptography coprocessor with Kyber KEM as a case study. We optimize the implementation of the NTT core by merging the layers and an efficient reduction unit by creating a configurable butterfly core. Besides, we propose a coprocessor architecture that can perform all KEM operations for Kyber. Overall, our NTT core shows more than 44% improvement in terms of $A \times T$. The proposed Kyber coprocessor architecture also performs key generation, encapsulation, and decapsulation in 9, 12, and 19 μs for a security level comparable to AES-128, respectively, on an Artix-7 FPGA.

Acknowledgment

The authors would like to thank the reviewers for their comments. This work is supported in parts by a grant from NSF-1801341.

References

1. P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pp. 124–134, 1994.
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 specification and supporting documentation (version 3.0). submission to the NIST post-quantum cryptography standardization project," 2020.
3. NISTIR 8309, "Status report on the second round of the NIST post-quantum cryptography standardization process," *National Institute of Standards and Technology*, 2020.
4. NIST, "Submission requirements and evaluation criteria for the post-quantum cryptography standardization process," *National Institute of Standards and Technology*, 2016.
5. T. Pöppelmann and T. Güneysu, "Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware," in *Progress in Cryptology - LATINCRYPT 2012 - 2nd International Conference on Cryptology and Information Security in Latin America, Santiago, Chile, October 7-10, 2012. Proceedings*, pp. 139–158, 2012.
6. S. S. Roy, F. Vercauteren, N. Mentens, D. D. Chen, and I. Verbauwhede, "Compact Ring-LWE cryptoprocessor," in *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, pp. 371–391, 2014.
7. 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. Cryptogr. Hardw. Embed. Syst.*, vol. 2020, no. 2, pp. 49–72, 2020.
8. P. Longa and M. Naehrig, "Speeding up the number theoretic transform for faster ideal lattice-based cryptography," in *Cryptology and Network Security - 15th International Conference, CANS 2016, Milan, Italy, November 14-16, 2016, Proceedings*, pp. 124–139, 2016.
9. U. Banerjee, T. S. Ukyab, and A. P. Chandrakasan, "Sapphire: A configurable crypto-processor for post-quantum lattice-based protocols (extended version)," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 1140, 2019.
10. Z. Chen, Y. Ma, T. Chen, J. Lin, and J. Jing, "Towards efficient Kyber on FPGAs: A processor for vector of polynomials," in *25th Asia and South Pacific Design Automation Conference, ASP-DAC 2020, Beijing, China, January 13-16, 2020*, pp. 247–252, 2020.
11. A. C. Mert, E. Karabulut, E. Öztürk, E. Savas, M. Becchi, and A. Aysu, "A flexible and scalable NTT hardware: Applications from homomorphically encrypted deep learning to post-quantum cryptography," in *2020 Design, Automation & Test in Europe Conference & Exhibition, DATE 2020, Grenoble, France, March 9-13, 2020*, pp. 346–351, 2020.
12. A. C. Mert, E. Karabulut, E. Öztürk, E. Savas, and A. Aysu, "An extensive study of flexible design methods for the number theoretic transform," *IEEE Transactions on Computers*, pp. 1–1, 2020.
13. E. Karabulut and A. Aysu, "RANTT: A RISC-V architecture extension for the number theoretic transform," in *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, pp. 26–32, 2020.

14. T. Fritzmann and J. Sepúlveda, "Efficient and flexible low-power NTT for lattice-based cryptography," in *IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2019, McLean, VA, USA, May 5-10, 2019*, pp. 141–150, 2019.
15. Y. Xing and S. Li, "An efficient implementation of the NewHope key exchange on FPGAs," *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 67-I, no. 3, pp. 866–878, 2020.
16. C. Du, G. Bai, and X. Wu, "High-speed polynomial multiplier architecture for Ring-LWE based public key cryptosystems," in *Proceedings of the 26th edition on Great Lakes Symposium on VLSI, GLVLSI 2016, Boston, MA, USA, May 18-20, 2016*, pp. 9–14, 2016.
17. P.-C. Kuo, W.-D. Li, Y.-W. Chen, Y.-C. Hsu, B.-Y. Peng, C.-M. Cheng, and B.-Y. Yang, "High performance post-quantum key exchange on FPGAs," *IACR Cryptology ePrint Archive*, p. 690, 2017.
18. D. T. Nguyen, V. B. Dang, and K. Gaj, "A high-level synthesis approach to the software/hardware codesign of NTT-based post-quantum cryptography algorithms," in *International Conference on Field-Programmable Technology, FPT 2019, Tianjin, China, December 9-13, 2019*, pp. 371–374, 2019.
19. D. T. Nguyen, V. B. Dang, and K. Gaj, "High-level synthesis in implementing and benchmarking number theoretic transform in lattice-based post-quantum cryptography using software/hardware codesign," in *Applied Reconfigurable Computing. Architectures, Tools, and Applications - 16th International Symposium, ARC 2020, Toledo, Spain, April 1-3, 2020, Proceedings [postponed]*, pp. 247–257, 2020.
20. E. Alkim, H. Evkan, N. Lahr, R. Niederhagen, and R. Petri, "ISA extensions for finite field arithmetic accelerating Kyber and NewHope on RISC-V," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2020, no. 3, pp. 219–242, 2020.
21. H. Seo, M. Anastasova, A. Jalali, and R. Azarderakhsh, "Supersingular isogeny key encapsulation (sike)round 2 on ARM Cortex-M4," *IEEE Transactions on Computers*, pp. 1–1, 2020.
22. R. Elkhatib, R. Azarderakhsh, and M. Mozaffari Kermani, "Highly optimized montgomery multiplier for SIKE primes on FPGA," in *27th IEEE Symposium on Computer Arithmetic, ARITH 2020, Portland, OR, USA, June 7-10, 2020*, pp. 64–71, 2020.
23. M. Anastasova, R. Azarderakhsh, and M. Mozaffari Kermani, "Fast strategies for the implementation of SIKE round 3 on ARM Cortex-M4," *IACR Cryptol. ePrint Arch.*, vol. 2021, p. 115, 2021.
24. M. Bisheh Niasar, R. E. Khatib, R. Azarderakhsh, and M. Mozaffari Kermani, "Fast, small, and area-time efficient architectures for key-exchange on Curve25519," in *27th IEEE Symposium on Computer Arithmetic, ARITH 2020, Portland, OR, USA, June 7-10, 2020*, pp. 72–79, 2020.
25. M. Bisheh Niasar, R. Azarderakhsh, and M. Mozaffari Kermani, "Efficient hardware implementations for elliptic curve cryptography over Curve448," in *Progress in Cryptology - INDOCRYPT 2020 - 21st International Conference on Cryptology in India, Bangalore, India, December 13-16, 2020, Proceedings*, pp. 228–247, 2020.
26. K. Basu, D. Soni, M. Nabeel, and R. Karri, "NIST post-quantum cryptography-A hardware evaluation study," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 47, 2019.
27. T. Fritzmann, G. Sigl, and J. Sepúlveda, "RISQ-V: Tightly coupled RISC-V accelerators for post-quantum cryptography," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2020, no. 4, pp. 239–280, 2020.

28. G. Xin, J. Han, T. Yin, Y. Zhou, J. Yang, X. Cheng, and X. Zeng, "VPQC: A domain-specific vector processor for post-quantum cryptography based on RISC-V architecture," *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 67-I, no. 8, pp. 2672–2684, 2020.
29. V. B. Dang, F. Farahmand, M. Andrzejczak, K. Mohajerani, D. T. Nguyen, and K. Gaj, "Implementation and benchmarking of round 2 candidates in the NIST post-quantum cryptography standardization process using hardware and software/hardware co-design approaches," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 795, 2020.
30. Y. Huang, M. Huang, Z. Lei, and J. Wu, "A pure hardware implementation of CRYSTALS-Kyber PQC algorithm through resource reuse," *IEICE Electronics Express*, vol. advpub, 2020.
31. Y. Xing and S. Li, "A compact hardware implementation of CCA-secure key exchange mechanism CRYSTALS-KYBER on FPGA," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2021, no. 2, pp. 328–356, 2021.
32. L. Botros, M. J. Kannwischer, and P. Schwabe, "Memory-efficient high-speed implementation of Kyber on Cortex-M4," in *Progress in Cryptology - AFRICACRYPT 2019 - 11th International Conference on Cryptology in Africa, Rabat, Morocco, July 9-11, 2019, Proceedings*, pp. 209–228, 2019.
33. E. Alkim, Y. A. Bilgin, M. Cenk, and F. Gérard, "Cortex-m4 optimizations for {R, M} LWE schemes," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2020, no. 3, pp. 336–357, 2020.
34. M. J. Kannwischer, J. Rijneveld, P. Schwabe, and K. Stoffelen, "PQM4: post-quantum crypto library for the ARM Cortex-M4," 2018.
35. J. W. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS-Kyber: A CCA-secure module-lattice-based KEM," in *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018*, pp. 353–367, 2018.
36. E. Fujisaki and T. Okamoto, "Secure integration of asymmetric and symmetric encryption schemes," in *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, pp. 537–554, 1999.
37. C. Du and G. Bai, "Towards efficient polynomial multiplication for lattice-based cryptography," in *IEEE International Symposium on Circuits and Systems, ISCAS 2016, Montréal, QC, Canada, May 22-25, 2016*, pp. 1178–1181, 2016.
38. G. Bertoni, J. Daemen, S. Hoffert, M. Peeters, and G. V. Assche, "Keccak in VHDL," 2020.
39. J. Howe, M. Martinoli, E. Oswald, and F. Regazzoni, "Exploring parallelism to improve the performance of frodokem in hardware." Cryptology ePrint Archive, Report 2021/155, 2021.
40. S. S. Roy and A. Basso, "High-speed instruction-set coprocessor for lattice-based key encapsulation mechanism: Saber in hardware," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2020, no. 4, pp. 443–466, 2020.
41. M. Bisheh-Niasar, R. Azarderakhsh, and M. Mozaffari-Kermani, "Area-time efficient hardware architecture for signature based on Ed448," *IEEE Transactions on Circuits and Systems II: Express Briefs*, pp. 1–1, 2021.
42. M. Bisheh Niasar, R. Azarderakhsh, and M. Mozaffari Kermani, "Optimized architectures for elliptic curve cryptography over Curve448," 2020.