# High-performance area-efficient polynomial ring processor for CRYSTALS-Kyber on FPGAs

Zhaohui Chen[ab], Yuan Ma[b*], Tianyu Chen[b], Jingqiang Lin[b], Jiwu Jing[a]

[a]School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing, China, 100049

[b]State Key Laboratory of Information Security, Institute of Information Engineering, CAS, Beijing, China, 100093

*Corresponding author

Tel.: +86-13466572260, E-mail: mayuan@iie.ac.cn

Mailing address: University of Chinese Academy of Sciences, No.19(A) Yuquan Road, Shijingshan District, Beijing, China 100049

E-mail addresses: chenzhaohui17@mails.ucas.ac.cn (Z. Chen), mayuan@iie.ac.cn (Y. Ma), chentianyu@iie.ac.cn (T. Chen), linjingqiang@iie.ac.cn (J. Lin), jwjing@ucas.ac.cn (J. Jing).

**Abstract: The quantum-resistant attribute is a new design criterion for cryptography algorithms in the era of quantum supremacy. Lattice-based cryptography is proved to be secure against quantum computing. CRYSTALS-Kyber is a lattice-based promising candidate in the post-quantum cryptography standardization process. This paper proposes a high-performance polynomial ring processor for the CRYSTALS-Kyber algorithm. The processor executes optimized polynomial ring arithmetic, which cuts off over 20%/50% on the times of modular multiplication/addition compared with the straightforward implementations. Besides, the forward and inverse Number Theoretic Transform (NTT) reuse the control logic with the help of an efficient configurable butterfly unit to minimize the area of the finite state machine. Further, the underlying dual-column sequential storage scheme breaks the bottleneck of memory accessing. To evaluate the performance, a fully pipelined architecture is implemented on a low-cost FPGA platform. Benefiting from these optimizations, the Kyber1024 processor can perform NTT operation for a 4-dimensional polynomial vector in 17.1$\mu s$, and it achieves speedup by a factor of 2.1 compared with the state-of-the-art implementation.**

**Keywords: Post-quantum cryptography, Polynomial ring, CRYSTALS-Kyber, Number-Theoretic Transform, FPGA**

# 1. Introduction

Since Diffie and Hellman proposed the public key cryptography (PKC) algorithm in 1976 [1], it has been one of the foundations of information security. Most standardized PKC technologies such as digital signature, authentication and TLS/SSL key exchange are based on large integer factoring and discrete logarithm problem. As a disruptive challenge, quantum computers would completely break these cryptography systems with Shor's algorithm [2]. To seek for appropriate substitutes, the National Institute of Standards and Technology (NIST) called for post-quantum public-key encryption (PKE), key encapsulation mechanism and digital signature schemes in 2016. Interest in lattice-based cryptography (LBC) has increased due to the quantum-resistant properties and the potential for high-speed implementation [3, 4] with relatively small key and ciphertext size [5].

Regev [6] introduced Learning With Errors (LWE) problem supported by a theoretical proof of security over lattice. However, a large parameter matrix $\mathbf{A}$ limits its efficiency. Lyubashevsky et al. [7] proposed Ring-Learning With Errors (Ring-LWE) over polynomial ring to avoid the large matrix. Although Ring-LWE is more practical than the standard LWE, its algebraic structure might enable threatening attacks [8]. Module-Learning With Errors (Module-LWE) hardness assumption proposed in [9] provides a trade-off between security and efficiency with a scalable polynomial vector (polyvec) structure. As a competitive instance, CRYSTALS-Kyber (Kyber) [10] algorithm provides similar security parameters and baseline

performance compared with the relatively mature Ring-LWE-based scheme named NewHope [11]. As the most computation-intensive operation, multiplication over polynomial narrows the bottlenecks in high-performance applications. Both Kyber and NewHope set specific polynomial parameters to speed up the crucial operation with a recursive algorithm named Number-Theoretic Transform (NTT). The butterfly algorithm unit can reduce the computational complexity of NTT from $O(n^2)$ to $O(n \log n)$ [12].

There has been increased interest in implementing LBC on hardware towards both high-throughput [13–17] and compact [18–20] applications. However, there are still problems to be solved for the state-of-the-art processors. First, there exist miscellaneous operations in polynomial ring arithmetic logic [21]. Although the proposed algorithm optimization in [22, 23] reduces the latency, a straightforward implementation doubles the area of control and arithmetic logic on hardware. The processors in [24, 25] implement configurable butterfly units (CBU) to avoid the pre-scale. However, these modules need more modular adders. Moreover, the signal flow of in-place NTT requires additional order-reverse operations. In the typical implementations [26, 27], it costs about 1024 clock cycles for NewHope1024 in each NTT computation. A recent work [25] reduces the computation cost, but the order-reverse operation is still not eliminated unless both participants of the key exchange arranged with omitting the order-reverse operation in advance. Besides, NTT requires high-bandwidth memory accessing, reading coefficients of polynomial rings takes a lot of clock cycles [19, 28]. Benefiting from the polyvec structure in Kyber algorithm, the memory accessing scheme can be further optimized. The efficient polyvec processor [29] for Kyber achieves high efficiency. However, its performance is not satisfying in some real-time processing systems. Thus, implementing a high-performance area-efficient processor is still hard work.

In this paper, we design and implement an FPGA-based high-performance processor for operations over polynomial rings. This work is based on a preliminary work published in [29]. The contributions of this paper are summarized as the following:

1. Optimize polynomial ring arithmetic. This optimization eliminates the miscellaneous operations, thus saving more than 20% modular multiplication operations and more than 50% modular addition operations compared with the typical implementations.
2. Optimize NTT signal flow. The proposed signal flow reuses the loop control logic, which saves nearly 50% of the resource compared with the straightforward implementation. It even reduces 17% of the combinational circuit compared with the control unit of a compact implementation [29].
3. Develop dual-column sequential storage and bit-reversed address accessing. These techniques keep the datapath free of bubble and avoid redundant latency caused by order-reverse.
4. Design a compact CBU with a DSP slice. It supports Cooley-Tukey butterfly-based forward NTT, Gentlemen-Sande butterfly-based inverse NTT and other meta operations.
5. Implement and evaluate the pipelined processor. The fully pipelined processors can perform 7 commands over polynomial rings in the Kyber algorithm. The maximum frequency is 240MHz for Kyber1024 processor, and it can perform NTT for 4-dimensional polyvec in 17.1$\mu s$. Compared with [29], this high-performance processor achieves 2.1x speedup.

The remainder of the paper is organized as follows: Section 2 provides a brief mathematical background of Kyber and polynomial ring arithmetic optimization. Section 3 introduces our optimized NTT signal flow and novel storage structure. Section 4 presents the CBU and modular reduction arithmetic. Section 5 describes the hardware architecture and implementation technology on FPGAs. We evaluate the high-performance processor and compare the results with the counterparts in Section 6. Finally, Section 7 concludes the proposed work.

# 2. Polynomial ring arithmetic and its optimization

### *2.1 Symbol Definition*

The ring of integer polynomials modulo $X^n+1$ is represented as $R_q = Z_q[X]/(X^n+1)$ in which $n$ is the dimension and $q$ is the modulus for polynomial coefficients. For a polynomial $a \in R_q$, we suggest that $a = \sum_{i=0}^{n-1} a_i x^i, a_i \in Z_q$. Let $ab$ and $a \circ b$ denote polynomial multiplication and point-wise multiplication both over $R_q$, respectively. A $k$-dimensional polyvec is written as bold lower-case letters like $\mathbf{s}$. A bold upper-case letter like $\mathbf{A}$ is $k \times k$-dimensional polynomial matrices (polymatrices). By default, all the elements in polyvecs or polymatrices are polynomials over $R_q$. Vectors will be column-wise by default. For a vector $\mathbf{a}$ (or matrix $\mathbf{A}$), its transpose is $\mathbf{a}^T$ (or $\mathbf{A}^T$). The centered binomial distribution is defined as $\beta_\eta$ for some positive integer $\eta$. A polynomial is sampled from $\beta_\eta$ if each coefficient is sampled from $\beta_\eta$, and a $k$-dimensional polyvec can be sampled according to $\beta_\eta^k$. Note that the coefficients sampled from $\beta_\eta$ are much smaller than the modulus $q$. Throughout the paper, the normal-font NTT refers to both the general technique and the corresponding algorithm.

### *2.2 Kyber Algorithm and Parameter Sets*

Kyber PKE is a candidate in NIST post-quantum cryptography standardization consisting of key generation, encryption and decryption algorithms. Given $\mathbf{A}$ as a global polymatrix with coefficients sampled from uniform distribution in NTT domain, a simplified version is shown as follows.

**Table 1**
Parameter comparisons between Kyber and NewHope in NIST standardization.

| Algorithm | Parameter n/k/q | Quantum Security(bit) | Ciphertext Size (Byte) |
|---|---|---|---|
| Kyber1024 | 256/4/7681 | 218 | 1504 |
| Kyber512 | 256/2/7681 | 102 | 800 |
| NewHope1024 | 1024/1/12289 | 233 | 2208 |
| NewHope512 | 512/1/12289 | 101 | 1120 |

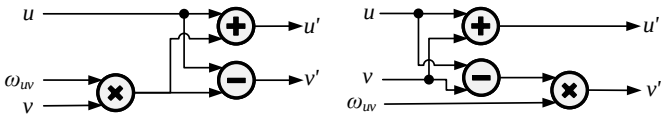
- $Kyber.KeyGen(\mathbf{A})$: Choose two polyvec $\mathbf{s}, \mathbf{e}$ from $\beta_\eta^k$ and compute $\mathbf{t} = \mathbf{As} + \mathbf{e}$. The public key is ($\mathbf{A}$, $\mathbf{t}$) and the private key is $\mathbf{s}$.
- $Kyber.Enc(\mathbf{A}, \mathbf{t}, \mathbf{m})$: The sender encodes the message $m$ to polynomial $\overline{m}$ with an encoder designed to tolerate introduced errors by mapping "0" bits to 0 and "1" bits to $\lceil q/2 \rfloor$. Sample polyvec $\mathbf{r}, \mathbf{e}_1$ from $\beta_\eta^k$ and $e_2$ from $\beta_\eta$. The ciphertext then consists of polyvec $\mathbf{u} = \mathbf{A}^T\mathbf{r} + \mathbf{e}_1$ and polynomial $v = \mathbf{t}^T\mathbf{r} + e_2 + \overline{m}$.
- $Kyber.Dec(\mathbf{s}, \mathbf{u}, v)$: The recipient computes $m' = v - \mathbf{s}^T\mathbf{u}$ and recover the original message $m$ from polynomial $m'$ using a decoder. The decoder is used to decode a coefficient to "1" bit if the coefficient of $m'$ is closer to $\lceil q/2 \rfloor$ than to 0, and decode to a "0" bit otherwise.

Generally speaking, $m'$ is not equal to $\overline{m}$, because the error polyvec $\mathbf{e}_1$ and error polynomial $e_2$ are introduced in the encryption process. If the private key is correct, the decrypted $m$ is equal to the original plaintext with a negligible probability of decryption failure, because the encoder and decoder can tolerate the errors. Kyber provides different post-quantum security levels which enables a fair comparison with the NewHope algorithm as shown in Table 1. In this paper, we focus on Kyber1024 with paranoid parameters and Kyber512 with light parameters. The most expensive operations are over fixed polynomial ring $Z_{7681}[X]/\left(X^{256}+1\right)$.

### *2.3 Multiplication over polynomial rings*

Arithmetic over polynomial ring is the crucial point of performance optimization. In 2012, NTT was introduced into the optimization of polynomial multiplication [21,30]. With the help of forward and inverse NTT, polynomial multiplication in the normal domain can be transferred to point-wise multiplication in the NTT domain. In $Z_{7681}[X]/\left(X^{256}+1\right)$, we can find an $n$-th root of unity $\omega$ and its modular square root $\psi$ such that $\psi^2 \equiv \omega \bmod q$.

Forward $n$-dimensional NTT is defined as $\mathrm{NTT}^\omega(a)_i = \sum_{j=0}^{n-1} a_j\omega^{ij} \bmod q, i = 0, 1, \ldots, n-1$. In practice, an efficient variant named negacyclic NTT is widely used to accelerate polyvec multiplication. We calculate polynomial multiplication as $\mathrm{PwM}_\psi(ab) = n^{-1}\mathrm{NTT}^{\omega^{-1}}(\mathrm{NTT}^\omega(\mathrm{PwM}_\psi(a)) \circ \mathrm{NTT}^\omega(\mathrm{PwM}_\psi(b)))$. Point-wise multiplying polynomial $a$ by $(1, \psi, \cdots, \psi^{n-1})$ is denoted as $\mathrm{PwM}_\psi(a)$ (define $\mathrm{PwM}_{\psi^{-1}}(a)$ similarly). For $i = 0, 1, \ldots, n-1$, Eq. (1) shows the forward negacyclic NTT algorithm.



(a) Cooley-Tukey (CT) (b) Gentlemen-Sande (GS)

**Fig. 1.** Comparison between the butterfly structures.

$$\begin{aligned}\text{Forward negNTT}(a)_i =& \mathrm{NTT}^\omega(\mathrm{PwM}_\psi(a))_i \\ =& \sum_{j=0}^{n-1} \psi^j a_j \omega^{ij} \bmod q\end{aligned} \tag{1}$$

The forward negacyclic NTT adds multiplication by exponents of $\psi$, called pre-scale operation, compared with normal NTT, which transforms the polynomial to NTT domain for point-wise multiplication. The corresponding inverse negNTT is quite similar. It substitutes $\omega$ to its inverse $\omega^{-1} \bmod q$ and replaces the additional pre-scale operation to a post-scale by exponents of $\psi^{-1} \bmod q$. Finally, each coefficient is multiplied by the scalar $n^{-1} \bmod q$. Inverse negacyclic NTT is denoted as Eq. (2).

$$\begin{aligned}\text{Inverse negNTT}(a)_i =& n^{-1}\mathrm{PwM}_{\psi^{-1}}(\mathrm{NTT}^{\omega^{-1}}(a))_i \\ =& n^{-1}\psi^{-i}\sum_{j=0}^{n-1} a_j \omega^{-ij} \bmod q\end{aligned} \tag{2}$$

For polyvecs and polymatrices, the NTT algorithm executes polynomial-by-polynomial and performs in-place operation based on butterfly structure. The two widely-used butterflies are Cooley-Tukey (CT) butterfly and Gentlemen-Sande (GS) butterfly as shown in Fig. 1. The input of the butterfly unit consists of two polynomial coefficients $u, v$ and corresponding parameter $\omega_{uv}$. The output $u', v'$ are stored in the original address of $u, v$. CT and GS butterfly have the same time-efficiency and consist of the same elements. CT butterfly performs multiplication before addition and subtraction, while GS butterfly uses the multiplication result only in the subtraction path.

The overall workflow of NTT-based polyvec multiplication is shown in Fig. 2, The coefficients of the polynomial are obtained by sampling, the steps for processing polyvec multiplication $c = \mathbf{a}^T\mathbf{b}$ are as follows.

1. Calculate $\mathrm{PwM}_\psi$, i.e. pre-scale, which requires $n \times k$ modular multiplication operations for both polyvec $\mathbf{a}$ and $\mathbf{b}$.
2. Perform order-reverse operation for $\mathbf{a}$ and $\mathbf{b}$, which typically consumes $n \times k$ clock cycles.
3. Transform polyvec $\mathbf{a}$ and $\mathbf{b}$ to NTT domain by $\mathrm{NTT}^\omega(\mathrm{PwM}_\psi(\mathbf{a}))$ and $\mathrm{NTT}^\omega(\mathrm{PwM}_\psi(\mathbf{b}))$. Each of them costs $\frac{n}{2} \times k \times \log(n)$ modular multiplication operations, which overlap modular addition operations.

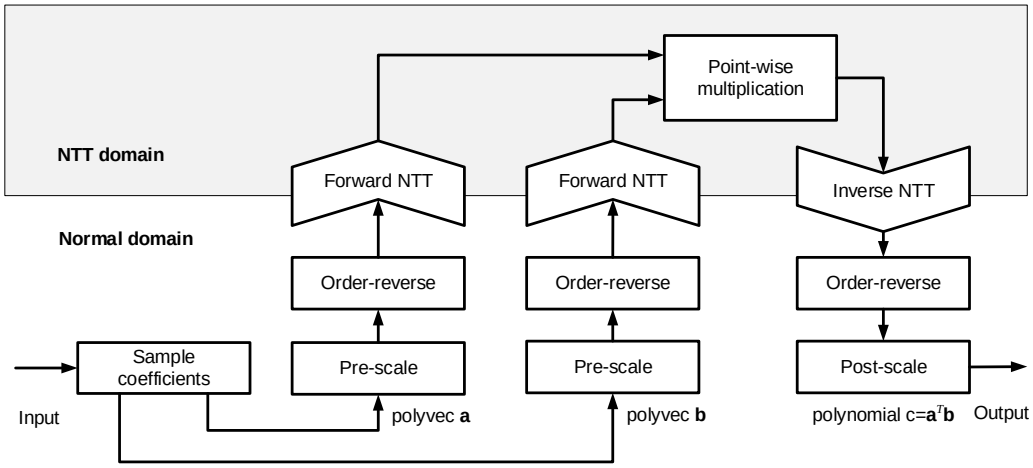



**Fig. 2.** Calculating polyvec multiplication with NTT.

4. Point-wise multiply the polymatrix $\mathbf{a}$ by the polyvec $\mathbf{b}$ in NTT domain. This costs $n \times k$ modular multiplication operations and $n \times (k-1)$ modular addition operations.
5. Perform $\text{NTT}^{\omega^{-1}}$ transformation on the point-wise multiplication result, which costs $\frac{n}{2} \times \log(n)$ modular multiplication operations.
6. Perform order-reverse operation, which typically consumes $n$ clock cycles.
7. Calculate $\text{PwM}_{\psi^{-1}}$, which costs $n$ modular multiplication operations.
8. Calculate $\text{PwM}_{n^{-1}}$ to obtain $\mathbf{a}^T\mathbf{b}$, which costs $n$ modular multiplication operations. Step 8 and Step 7 are called post-scale.

For a polynomial with n = 1024, the schoolbook polynomial multiplication requires 1048576 modular multiplication, while the butterfly-based NTT implementations (contains only 2 forward NTT operations, an inverse NTT and a point-wise multiplication) requires 16384 modular multiplications. In this case, NTT-based implementation consumes 64x fewer cycles, which explains that the state-of-the-art schoolbook implementation is still not as efficient as NTT implementation [31]. However, in-place NTT will cause the data to be stored in bit-reversed order, that is, from normal order to bit-reversal order (no $\rightarrow$ bo) or from bit-reversal order to normal order (bo $\rightarrow$ no). Therefore, order-reverse steps are introduced in the workflow to solve the problem.

### *2.4 Arithmetic optimization and meta operations*

In addition to the operations related to polyvec multiplication, Kyber algorithm also includes other operations such as polynomial addition and subtraction. An efficient optimization idea is to decompose and integrate the operations over polynomials, which can maximize the use of arithmetic logic and avoid data conflicts.

We observe that there are three types of computation-intensive formulas in Kyber. $\mathbf{t} = \mathbf{A}\mathbf{s} + \mathbf{e}$ in key generation algorithm and $\mathbf{u} = \mathbf{A}^T\mathbf{r} + \mathbf{e}_1$ in encryption algorithm consist of a polymatrix-polyvec multiplication and a polynomial addition, in which the elements of polymatrix $\mathbf{A}$ and $\mathbf{A}^T$ are sampled in NTT domain, and polymatrix-polyvec multiplication is equivalent to executing polyvec multiplication for $k$ times. Formula $v = \mathbf{t}^T\mathbf{r} + e_2 + \overline{m}$ includes a polyvec multiplication and 2 polynomial additions. Formula $m' = v - \mathbf{s}^T\mathbf{u}$ consists of a polyvec multiplication and a polynomial subtraction. Note that the transposition operation of the polyvec is included in the polynomial multiplication operation by default.

The ideal NTT-based polyvec multiplication only includes forward NTT, point-wise multiplication and inverse NTT. However, we observe some miscellaneous operations such as steps 1, 2, 6, 7, 8 described in subsection 2.3. In a previous work [29], GS butterfly is used for forward and inverse transformation, thus avoiding step 7. However, step 1 is performed at the time of data input, which increases the latency of loading data. In a software implementation [22], the scheme uses CT butterfly in forward NTT and GS butterfly in inverse NTT to avoid step 1 and step 7. The side effect of this strategy is that two sets of control and arithmetic units are required for the two butterfly units, which means that the area of the circuit will double. This work eliminates step 1 and step 7 and minimizes these adverse effects on control logic and arithmetic logic in subsections 3.1 and 4.1. Order-reverse operations in step 2 and step 6 can be eliminated with an addressing technique as shown in subsection 3.2. Unlike the point-wise multiplication for polynomials, the extra modular addition leads to more time and area consumption in the point-wise multiplication for polyvec. In step 4, modular multiplication operations can overlap addition operations with multiply-accumulate (MACC) command of DSP which directly accumulates the results of each multiplication into a built-in register, thus reducing the redundant latency for modular addition. In general, $n \times (k-1)$ logic operations are saved.

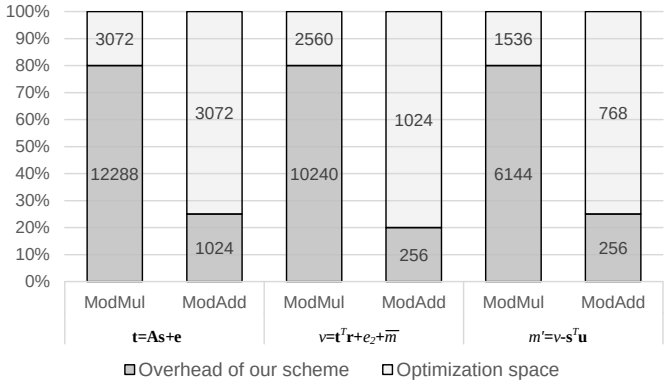
Through decomposing and merging operation steps, the 8 steps of typical polyvec multiplication $c = \mathbf{a}^T\mathbf{b}$ are re-

**Table 2**
Commands and usage in the formulas.

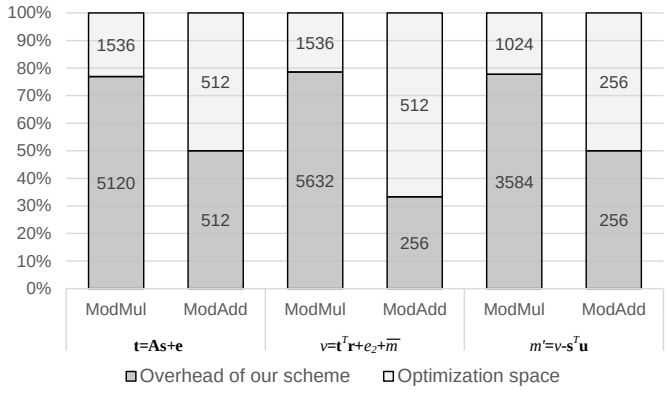
| Formula | PV_NTT | PV_INTT | PM_PWM | PV_PWM | PV_MADD | POLY_MSUB | POLY_MADD2 |
|---|---|---|---|---|---|---|---|
| | Polyvec forward NTT | Polyvec or polynomial inverse NTT | Polymatrix point-wise multiply polyvec | Polyvec point-wise multiply polyvec | Polyvec add polyvec | Polynomial subtract polynomial | Polynomial add polynomial twice |
| $\mathbf{t} = \mathbf{As} + \mathbf{e}$ | ✓ | ✓ | ✓ | | ✓ | | |
| $v = \mathbf{t}^T\mathbf{r} + e_2 + \overline{m}$ | ✓ | ✓ | | ✓ | | | ✓ |
| $m' = v - \mathbf{s}^T\mathbf{u}$ | ✓ | ✓ | | ✓ | | ✓ | |

constituted to 4 steps, i.e. forward NTT, point-wise multiplication, inverse NTT and post-scale. Similarly, miscellaneous operations in the mentioned computation-intensive formulas can also be optimized. Steps required for each formula are listed in Table 2. For $\mathbf{t} = \mathbf{As} + \mathbf{e}$ and $\mathbf{u} = \mathbf{A}^T\mathbf{r} + \mathbf{e}_1$, polyvec addition can be merged into post-scale by converting modular multiplication and addition into multiply-add (MADD) operation. For $v = \mathbf{t}^T\mathbf{r} + e_2 + \overline{m}$, polynomial addition twice can be merged into post-scale by multiply-add-add (MADD2) operation. For $m' = v - \mathbf{s}^T\mathbf{u}$, polynomial subtraction can be merged into post-scale by multiply-subtract (MSUB) operation. These critical operations over polynomial corresponding to commands for controlling the state machine and selection signals.

Benefit from integrating miscellaneous operations, the overall process of $\mathbf{t} = \mathbf{As} + \mathbf{e}$ requires only $(\log(n) + k) \times n \times k$ modular multiplications and $n \times k$ modular additions (in fact, MADD). That is, the proposed scheme saves 20% modular multiplications and 75% modular additions for Kyber1024, 23.1% modular multiplications and 50% modular additions for Kyber512. The optimization space for other formulas is also evaluated as shown in Fig. 3. In brief, we reduced the number of modular multiplication by more than 20% and that of modular addition by more than 50%.




(a) Kyber1024




(b) Kyber512

**Fig. 3.** Optimization of modular multiplication and modular addition.

# 3. Optimization of the NTT

This section introduces the optimization technology of NTT operation on hardware. First, the forward and inverse NTT reuse control logic to minimize the circuit area while avoiding miscellaneous operations. Then, we propose a polynomial-oriented data storage scheme.

### *3.1 NTT control logic*

The NTT algorithm proposed in [22] uses CT-based forward NTT to transform polynomials from normal order to bit-reversal order ($\text{NTT}^{CT,\omega}_{no\rightarrow bo}$). After point-wise multiplication, polynomials are transformed from bit-reverse order to normal order with GS-based forward NTT ($\text{NTT}^{GS,\omega^{-1}}_{bo\rightarrow no}$). NTT based on butterfly operation is a three-layer loop algorithm, and the loop control of each layer needs corresponding control logic. Because the signal flow of no $\rightarrow$ bo is different from that of bo $\rightarrow$ no, we need to double the control unit of the circuit. However, according to the research of [29], the control logic of NTT algorithm occupies even more hardware resources than the arithmetic logic, and the discrete control units for forward and inverse NTT will aggravate this disadvantage.

In this work, we propose an optimized NTT using no $\rightarrow$ bo signal flow in both forward and inverse transformations as shown in Algorithm 1. The overall signal flow is shown in Fig. 4. The three stages in Fig. 4 correspond to the

**Algorithm 1:** Optimized $\text{NTT}^{CT,\omega}_{no\to bo}$ and $\text{NTT}^{GS,\omega^{-1}}_{no\to bo}$

**Input:** Polynomial $a \in R_q$, store powers of $\psi$ in bit-reversed order in $psi[\,]$, and store $\psi^{-i}$ in normal order in $invpsi[\,]$.

**Output:** Polynomial $a \in R_q = \text{NTT}^{CT,\omega}_{no\to bo}(a)$ or $\text{NTT}^{GS,\omega^{-1}}_{no\to bo}(a)$.

```
1  t = n
2  for (m = 1; m < n; m = m × 2) do
3      t = t/2
4      for (i = 0; i < m; i++) do
5          jFirst = 2 × i × t
6          jLast = 2 × i × t + t
7          for (j = 0; j < t; j++) do
8              if NTT^{CT,ω}_{no→bo} then
9                  ω = psi[m + i]
10                 (u1, u2) = a[j + jFirst]
11                 (v1, v2) = a[j + jLast]
12                 a[j + jFirst] =
                     ((u1 + v1ω), (u2 + v2ω))   mod q
13                 a[j + jLast] =
                     ((u1 − v1ω), (u2 − v2ω))   mod q
14             else
15                 /* NTT^{GS,ω^{-1}}_{no→bo}          */
16                 ω^{-1} = invpsi[(2 × j + 1) × m]
17                 (u1, u2) =
                     a[BitReverse(j + jFirst)]
18                 (v1, v2) = a[BitReverse(j + jLast)]
19                 a[BitReverse(j)] =
                     (u1 + v1, u2 + v2)  mod q
20                 a[BitReverse(j + t)] =
                     ((u1 − v1)ω^{-1}, (u2 − v2)ω^{-1})
                     mod q
21             end
22         end
23     end
24 end
```

outermost cycle in Algorithm 1, and each cycle is taken as a stage. The idea is to reuse the outer loop control logic to minimize the area of the control unit. The difference of forward and inverse NTT is limited to the innermost loop, i.e. butterfly operation, the arithmetic logic of CT butterfly is shown in lines 10 to 13, while that of GS butterfly is shown in lines 16 to 20. The proposed optimization eliminates steps 1 and 7 in subsection 2.3 and does not require another loop control circuit. Thus, the optimized algorithm saves nearly half of NTT control logic. Since most of the parameters in the algorithm, such as $m$ and $t$, are powers of 2, multiplication can be replaced by shift operation in addressing.

### *3.2 Bit-reversed address accessing scheme*

Since the output of the forward NTT is in a bit-reversed order, there is still a gap to perform the inverse NTT that requires normal-order input. As a straightforward method, [26, 27] reverse the order of coefficients, which consumes about $n \times k$ clock cycles. In the state-of-the-art implementation [25], an order-reverse step can be completed in $n \times k/4$ clock cycles.

This work uses the underlying bit-reversed address accessing technique to avoid all the order-reverse operations. In the forward transformation, $\text{NTT}^{CT,\omega}_{no\to bo}$ reverses the normal-order polynomials to bit-reversed order. As for $\text{NTT}^{GS,\omega^{-1}}_{no\to bo}$, we use the inverted bit connection of address line as shown in Eq. (3). This multiplexer solves the gap between bit-reversed output and normal-order input.

$$\begin{aligned}\text{addr} =&(\text{command} == \text{PV\_INTT})?\\ &\text{addr\_line[0:mem\_width-1]} :\\ &\text{addr\_line[mem\_width-1:0]}\end{aligned} \tag{3}$$

For example, suppose memory $mem_{seq}$ stores an 8-dimensional polynomial $a$ in sequential order. That is, $a[0]$ to $a[7]$ are stored in addresses 0 to 7. Memory $mem_{rev}$ stores the same polynomials $a$ in bit-reversed order. That is, $a[0], a[4], a[2], a[6], a[1], a[5], a[3], a[7]$ are stored in addresses 0 to 7. For the memory with a 3-bit width address line, accessing address $(bit2, bit1, bit0)_2$ from memory $mem_{seq}$ would always be equivalent to access $(bit0, bit1, bit2)_2$ from memory $mem_{rev}$.

As shown in Fig. 4, the normal address is accessed in the forward NTT, and the bit-reversed address is accessed in the inverse NTT. The inverse transformation counteracts the order change caused by forward transformation. After $\text{NTT}^{GS,\omega^{-1}}_{no\to bo}$, the data in the memory is in sequential order. This technique estimates the step 2 and 6 in subsection 2.3, thus at least $\frac{n}{2} \times k$ cycles are saved.

### *3.3 Dual-column storage structure*

In-place NTT implementation helps to achieve efficient memory utilization. In this case, the butterfly structure needs to read 2 coefficients and write back 2 results, except the factor $\omega$ or $\omega^{-1}$. For block RAMs (BRAMs) on FPGAs, a read or write operation takes at least 1 clock cycle. Thus, to read a value and then write in the same address using one port needs 2 clock cycles. Even though the two ports of one BRAM can be configured as 2 read ports, 2 write ports or 1 read + 1 write ports, the coefficients can not be read and written in the same clock cycle if only one coefficient is stored in each address. The sequential storage of polynomials limits the number of read-write coefficients per clock cycle of BRAM. Affected by insufficient memory bandwidth, dataflow would get bubbles and pipeline stalls would occur in arithmetic logic. As a result, memory access becomes the bottleneck of efficient implementation on hardware. In [28], the implementation for

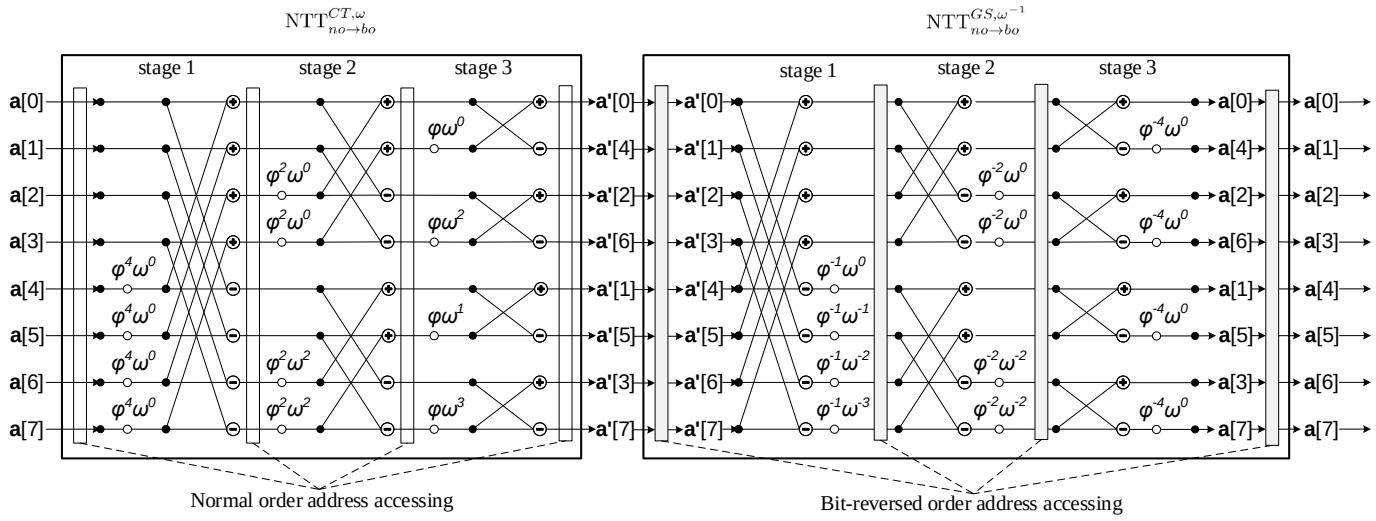



**Fig. 4.** $\mathrm{NTT}^{CT,\omega}_{no\to bo}$ and $\mathrm{NTT}^{GS,\omega^{-1}}_{no\to bo}$ signal flow graphs for 8-dimensional polynomials.

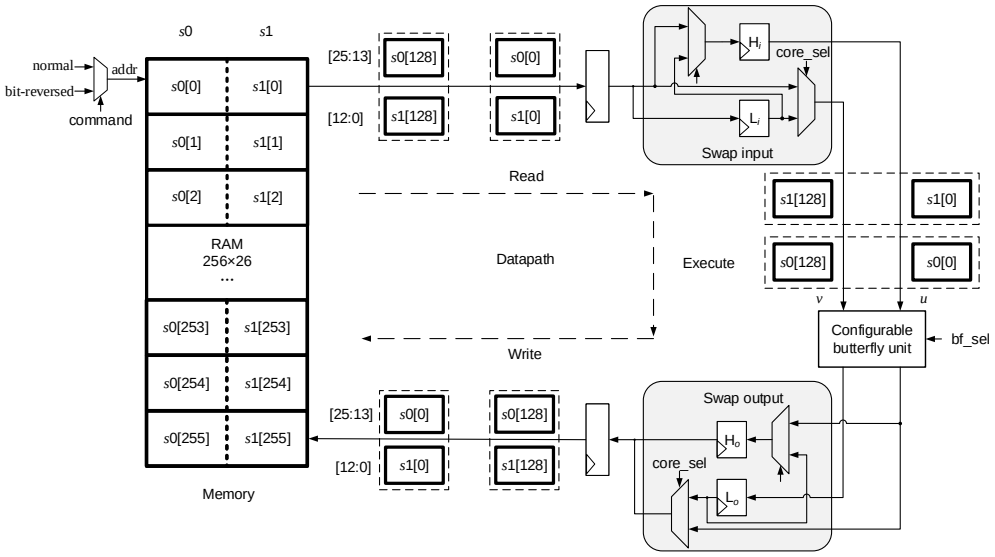



**Fig. 5.** Datapath of the proposed swap circuit.

NewHope-Simple takes 7 clock cycles to perform a butterfly operation, in which memory accessing occupies 2 extra cycles. In [18], 50% of the storage is vacant in about half of the running time, which leads to under-utilization. In [20], the coefficients are stored in two columns in a swapped order, but this requires redundant clock cycles to rearrange the coefficient pairs. According to the structural characteristics of polyvec in Kyber, we propose a dual-column sequential storage structure. The proposed structure enables a pipelined memory access scheme which reads and writes 2 coefficients in each clock cycle without redundant latency.

The memory is arranged quite concise as shown in Fig. 5. For polyvec $\mathbf{s} = \begin{bmatrix} s0, s1 \end{bmatrix}^T$, the polynomial coefficients in $s0$ and $s1$ are sequentially stored in pairs according to $(s0[0],\ s1[0])$, $(s0[1],\ s1[1])$, $\cdots$, $(s0[255],\ s1[255])$. Polynomial $s0$ is stored in the higher 13 columns and $s1$ is stored in the lower 13 columns. In the memory, coefficients with the same serial number are stored in the same address. High-dimensional polyvecs in Kyber1024 are also applicable by increasing the memory depth.

In this scheme, memory throughput is increased to read 2 coefficients and write back 2 coefficients in one clock cycle. Algorithm 1 describes a pair of coefficients belonging to different polynomials to be accessed in parallel and processed separately. To perform NTT polynomial-by-polynomial, the processor adopts a swapping circuit as shown in Fig. 5. The proposed circuit includes the swap input circuit and the swap output circuit. The function of the input circuit is to adjust the order of the coefficients to adapt to butterfly operation, and that of the output circuit is to store the coefficients in the original address. Taking the coefficient pairs $(s0[0], s1[0])$ and $(s0[128],\ s1[128])$

as an example, coefficient pair ($s0[0], s1[0]$) is read before ($s0[128]$, $s1[128]$). The swap input module swaps the order of the coefficients, and ($s0[0], s0[128]$) executes butterfly operation before ($s1[0], s1[128]$). Then, the renewed coefficient pair ($s0[0], s1[0]$) is written before ($s0[128]$, $s1[128]$) to there original address. The dataflow can be fully pipelined thus extra latency of memory access in [28] is saved. To avoid abnormal datapath, the selection signals of the swap in/out and butterfly operation unit are determined by the command currently executed by the processor. The datapath of swap and execute operations is described as the following steps.

1. Read the first coefficient pair ($s0[0], s1[0]$). Update the read address every clock cycle and the next address would be 128 for $n = 256$.
2. Read the second coefficient pair ($s0[128]$, $s1[128]$). Coefficients ($s0[0]$ and $s1[0]$) are stored in the $H_i$ and $L_i$ registers respectively.
3. Perform butterfly operation with the coefficients $s0[0]$ stored in $H_i$ and $s0[128]$. Coefficient $s1[0]$ shifts from $L_i$ to $H_i$, and $s1[128]$ is stored in $L_i$.
4. Perform butterfly operation with the coefficients $s1[0]$ stored in $H_i$ and $s1[128]$ stored in $L_i$. Outputs of butterfly $s0[0]$ and $s0[128]$ are stored in $H_o$ and $L_o$.
5. Data $s0[0]$ stored in $H_o$ and the output of butterfly $s1[0]$ are written into the address 0. Shift $s0[128]$ from $L_o$ to $H_o$ and $s1[128]$ is stored in $L_o$.
6. $s0[128]$ in $H_o$ and $s1[128]$ in $L_o$ are stored in address 128.

# 4. Optimization of the processing element

This section presents a DSP-based configurable processing element for various meta operations. A general-purpose modular reduction unit based on Barrett algorithm [32, 33] is used to optimize the critical arithmetic logic.

### *4.1 Efficient CBU*

Since the signal flow includes $\mathrm{NTT}^{CT,\omega}_{no\rightarrow bo}$ and $\mathrm{NTT}^{GS,\omega^{-1}}_{no\rightarrow bo}$, a straightforward architecture doubles the number of modular multipliers, modular adders and subtractors. The implementation in [24] reuses the modular multiplier in CT and GS butterfly. The state-of-the-art CBU [25] needs an additional modular subtractor. This work further proposes a more efficient CBU as shown in Fig. 6. Dataflow of the unit is controlled by bf_sel and dsp_sel signal. The CBU is configured to CT butterfly when bf_sel is 0, and GS butterfly when bf_sel is 1. The on-chip DSP slice performs multiply (MUL), subtract-multiply (SUBMUL), MADD and MACC.

The proposed CBU has the following three advantages. (1) The modular subtractor marked by the dotted line in Fig. 6 uses the subtractor in the DSP. This reduces the delay and area of LUT-based combinational circuits. (2) The circuit enables other arithmetic logic such as MADD2, when configuring bf_sel/dsp_sel to 0/MADD (output on o_bf_u port). (3) DSP slices have abundant sequential circuit resources, which support pipeline technology. This helps to improve the performance of the processor.

### *4.2 Modular arithmetic*

This subsection presents the hardware architecture for modular arithmetic for modulus 7681 of Kyber, as shown in Figure 8. In the modular multiplier, a Barrett algorithm module is used to modular reduce the output of DSP to $Z_{7681}$. Since all the coefficients of polynomials are less than 7681, the result of coefficient-wise multiplication is up to 26-bit. However, because our processing element includes MACC operation, the output width of DSP can be 28-bit for Kyber1024, and 27-bit for Kyber512.

The circuit in Fig. 7(a) implements the function of $r = x \bmod 7681$ for Kyber1024. It draws lessons from an efficient design [34], in which all multiplication operations with the determined values are replaced with shifting and adding. By inserting pipelines, the path delay caused by the adder tree can be reduced. The modular adder and modular subtractor in the processing elements are implemented as shown in Fig. 7(b) and Fig. 7(c).

# 5. Processor implementation and Pipeline

This section introduces the implementation techniques. The proposed fully pipelined processor can execute 7 operation commands. This work focuses on Kyber1024 with parameter sets (256, 4, 7681) and Kyber512 with parameter sets (256, 2, 7681).

### *5.1 Overall architecture*

The processor architecture shown in Fig. 8 consists of memory, arithmetic logic unit (ALU) and a finite state machine-based control unit(CU).

Since the polyvec coefficients need a mass of storage, it is advisable to store data in BRAMs. Each 36K-bit BRAMs slice in Xilinx 7 series FPGAs can be divided into two 18K-bit slices. The 18K-bit storage can be configured as $512 \times 36$ or $1024 \times 18$. Since the ceiling of each coefficient is $\lceil \log_2 7681 \rceil = 13$-bit width, each 2-dimensional polyvec can be stored in a block of $256 \times 26$ memory exactly in simple dual-port mode. The processor meets the maximum amount of storage consumption in polynomial ring processing, i.e. $\mathbf{As} + \mathbf{e}$.

### *5.2 Operation code*

The CU controls the circuit through the selection signal and memory addressing registers.

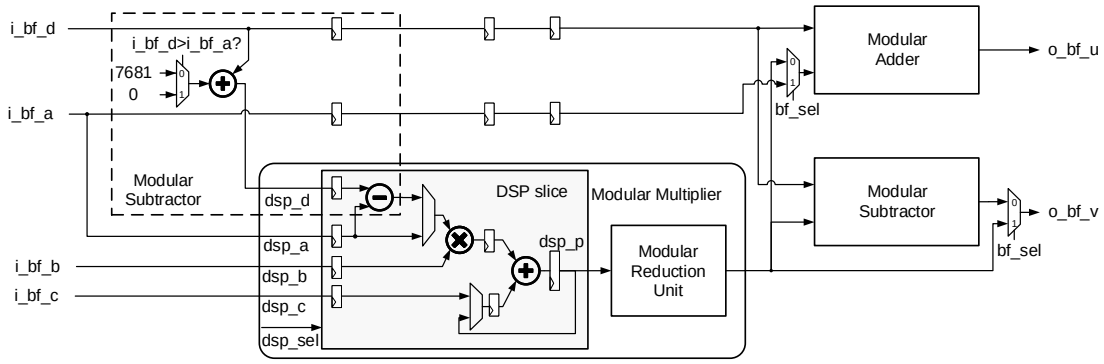
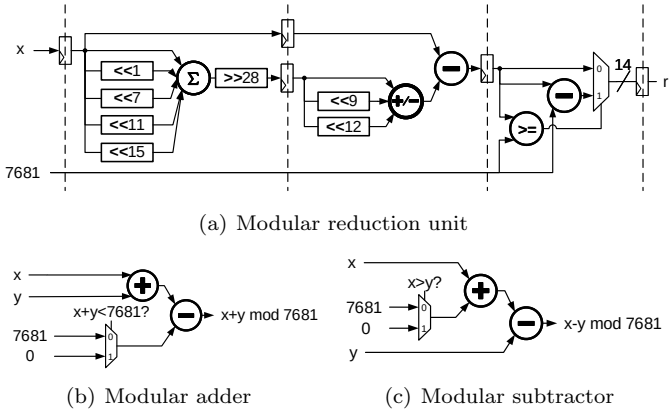



**Fig. 6.** CBU with a DSP slice on FPGAs.




(a) Modular reduction unit

(b) Modular adder

(c) Modular subtractor

**Fig. 7.** Modular arithmetic circuit for Kyber algorithm.

**Table 3**
Arithmetic commands and clock cycles.

| Command | opcode | bf_sel | dsp_sel | Cycle | |
|---|---|---|---|---|---|
| | | | | k=2 | k=4 |
| PM_PWM | 0000 | GS | MUL/MACC | 1036 | 4109 |
| PV_NTT | 0001 | CT | MUL | 2060 | 4109 |
| PV_INTT | 0010 | GS | SUBMUL | 2060 | 4109 |
| PV_PWM | 0011 | GS | MUL/MACC | 524 | 1037 |
| PV_MADD | 0100 | GS | MADD | 524 | 1037 |
| POLY_MSUB | 0110 | CT | MUL | 268 | 269 |
| POLY_MADD2 | 0111 | CT | MADD | 268 | 269 |
| IDLE | 1000 | NA | NA | NA | NA |

All the meta operations mentioned in subsection 2.4 are implemented in the proposed hardware architecture. An external selection signal controls the working stage of the processor. Meta operations are mapped to 4-bit opcodes as shown in Table 3. These opcodes can be decoded into CBU and DSP selection signals by the OP_decode module. The idea is to decouple the selection signals of arithmetic logic and the control signals of datapath with DSP_sel and bf_sel, respectively. In this way, the control unit has commendable extension capability.

*5.3 Pipeline techniques*

There are a lot of registers in the BRAMs and the DSP slice. The proposed architecture achieves high-performance with a relatively low consumption on flip-flops (FF). Since the modular multiplier is the most complex arithmetic logic, it becomes the critical path in the processor. In order to improve the maximum frequency, we inserted a bubble-free pipeline in datapath.

The whole operation is divided into 13/12 pipeline stages for the proposed Kyber1024/Kyber512 processor. Take the Kyber1024 processor as an example, operations in each stage are shown in Fig. 9 and explained as follows.

- **Stage 0 to 1.** Insert a 2-level read pipeline to reduce the path delay. Internal registers of BRAM are used for stage 0.
- **Stage 2 to 3.** The swap input module costs 2 clock cycles to swap the input coefficients as described in subsection 3.3.
- **Stage 4 to 10.** Insert a 7-level pipeline in the CBU. The DSP slice introduces 3 stages for each arithmetic, and the Barrett reduction unit is implemented with 4-level pipeline as described in subsection 4.1 (3-level pipeline for Kyber512).
- **Stage 11 to 12.** Insert a 2-level pipeline to swap the output order as described in subsection 3.3.
- **Stage 13.** Insert a pipeline to store the processed data into the memory.

# 6. Results and Comparison

We synthesize and implement the pipeline processor with Vivado 2018.2 edition on a Xilinx Artix-7 series FPGA. The XC7A200T FPGA platform has the advantages of low power consumption, and it provides middle end (-2) speed grade. The maximum frequency of the high-performance processor is 240MHz for Kyber1024 and 246MHz for Kyber512. The proposed processor can perform $\mathbf{t} = \mathbf{As} + \mathbf{e}$, $v = \mathbf{t}^T\mathbf{r} + e_2 + \overline{m}$ and $m' = v - \mathbf{s}^T\mathbf{u}$ in $55.5\mu s$, $48.1\mu s$ and $31.0\mu s$ for Kyber1024 and $23.1\mu s$, $28.3\mu s$, $20.0\mu s$ for Kyber512. The general rule is that a

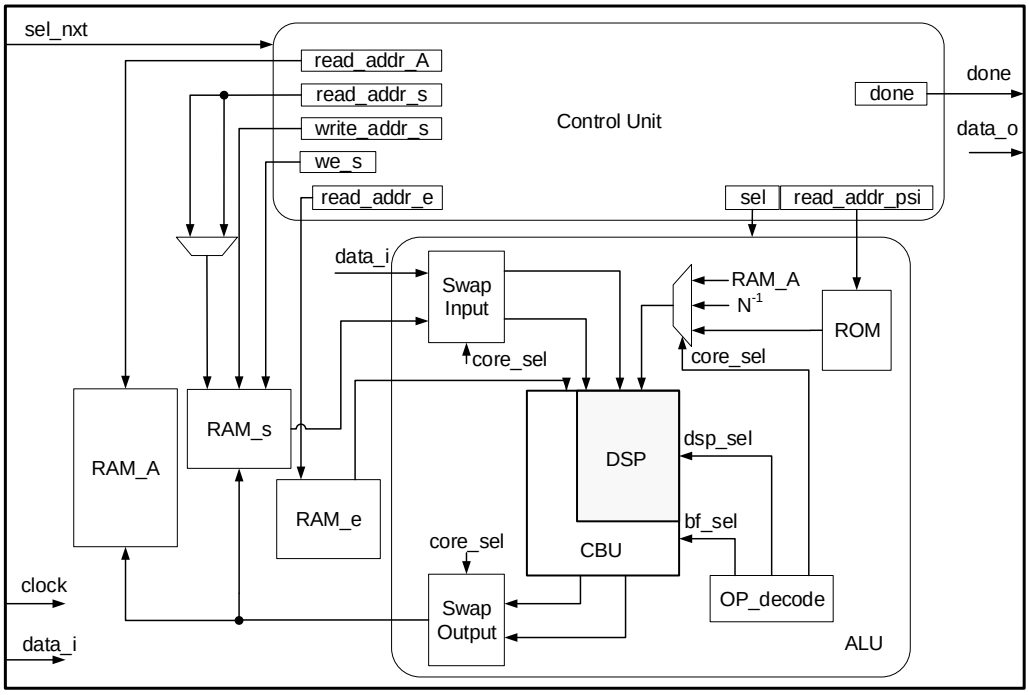



**Fig. 8.** Overall architecture of the polynomial ring processor.

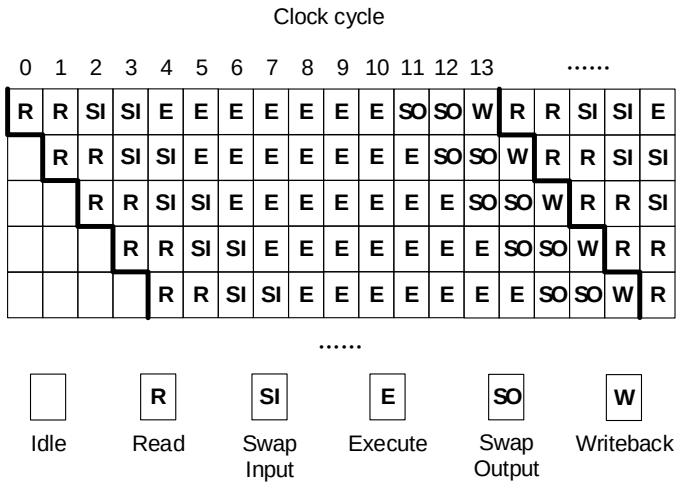



**Fig. 9.** Pipeline structure for the high-performance design.

higher security level cryptography requires more circuit area and greater operation latency.

### *6.1 Comparisons of NTT*

NTT is the most complex meta operation in many Ring-LWE based cryptography algorithms. This work is compared with NTT module of the state-of-the-art Kyber and NewHope processors. The post-place and route (Post-PAR) results are listed in Table 4. In some modules, the number of clock cycles of executing forward NTT and inverse NTT is different. We use the average value as a reference. For a fair comparison, the execution time has been normalized as the time cost for 1024-coefficient output. In Kyber algorithm, the NTT computation load of 4-dimension polyvec and 4 independent polynomials are the same. The efficiency is compared by throughput-area ratio, and the amounts of LUT and FF represent the occupation of combinational logic and sequential logic, respectively. As an advantage, our design has a balanced use of LUTs and FFs. For Kyber1024 and Kyber512, the ratio of LUT and FF is close to 1.

The processor in [29] has a balanced performance with a relatively low area by using GS butterfly for both forward and inverse NTT. However, there are two problems that need to be solved. Firstly, the control unit takes up nearly half of the area, which makes the reduction of logic in the ALU module less effective. Secondly, although its maximum frequency is reasonably high for a low-area implementation, it limits high-performance applications in many real-time scenarios. Compared with [29], we design a new command set for the selection signal and optimize the control logic. Although this work adds some additional command to fit the formulas in Kyber algorithm, the area of control unit for Kyber512 only costs 83% (209) LUTs and 102% (120) FFs of the former work. In order to improve the frequency, we add up to 13 levels pipeline to reduce the delay of the critical path. As a high-performance processor, this work improves the speed by 85.0% and achieves 1.65x LUT-efficiency for NTT in Kyber1024 algorithm.

Compared with the software implementation on STM32, our speed advantage is very obvious (about 13.83x). Note that this work only uses about 0.4% of the logic resources on the FPGA chip, which means that this core can also be implemented on a smaller FPGA. The

**Table 4**
Comparison between hardware implementations of NTT.

| Processor | Device | Parameter | Area | | | | Time | Throughput/Area | |
|---|---|---|---|---|---|---|---|---|---|
| | | (n × k × times) | LUT | FF | DSP | BRAM36 | ($\mu s$) | Kbps/LUT | Kbps/FF |
| **This Kyber1024** | Artix-7 | 256 × 4 × 1 | 533 | 514 | 1 | 3 | 17.1 | 1425 | 1477 |
| Kyber1024 [29] | Artix-7 | 256 × 4 × 1 | 442 | 237 | 1 | 3 | 31.6 | 864 | 1738 |
| **This Kyber512** | Artix-7 | 256 × 2 × 2 | 479 | 472 | 1 | 2 | **16.7** | **1620** | 1645 |
| Kyber512 [29] | Artix-7 | 256 × 2 × 2 | 477 | 237 | 1 | 2 | 30.2 | 973 | **1815** |
| Kyber software [35] | STM32F4 | 256 × 1 × 4 | - | - | - | - | 236.0 | - | - |
| NewHope[a][26] | Zynq-7000 | 1024 × 1 × 1 | 343 | 493 | 3 | 6 | 24.7 | **1651** | 1149 |
| NewHope-NIST[a][25] | Zynq-7000 | 1024 × 1 × 1 | 847 | 375 | 2 | 6 | 10.5 | 1574 | **3556** |
| NewHope-NIST[b][36] | Zynq-7000 | 1024 × 1 × 1 | 886 | 618 | 26 | 1 | 102.2 | 155 | 222 |
| NewHope-USENIX[c][27] | Zynq-7000 | 1024 × 1 × 1 | 2942 | 4881 | 8 | 0 | **8.4** | 569 | 343 |
| NewHope-USENIX [37] | Artix-7 | 1024 × 1 × 1 | 2382 | 1381 | 8 | 10 | 17.4 | 337 | 581 |
| NewHope-Simple [28] | Artix-7 | 1024 × 1 × 1 | 415 | 251 | 2 | 4 | 286.8 | 118 | 195 |

[a] The time for order-reverse is not included.
[b] The frequency is set the same as our Kyber1024 processor, i.e. 240MHz.
[c] The time for pre-scale and order-reverse are not included.

technology of this processor has the potential to be used in resource-constrained chips.

We observe that the state-of-the-art NewHope implementations [25–27] achieve high speed and efficiency. The Kyber1024 processor is to be compared with the counterparts of NewHope1024 $(1024, 1, 12289)$ with a similar security level. Compared with the counterparts of [25], their speed advantage is due to the use of two parallel butterfly structures, which also brings a burden on the area. In fact, their advantages also come from the following two aspects. Firstly, they implement the processor on an FPGA chip with smaller technology node and have higher speed grades (-3). This means that although we eventually achieved the same frequency, our processor will have better results after porting to high-end platforms. Secondly, they use more DSP slices and BRAMs to construct the processing element. Considering these dedicated resources, the efficiency of our processor is comparable.

The processor in [26] implemented a relatively compact pipelined NTT core on a high-end FPGA. The processor in our work takes up more circuits because it integrates multiple commands, while [26] does not include the time and area occupied by operations such as order-reverse. In addition, although our processor is implemented on a low-power FPGA, it shows an advantage in speed.

Compared with the HW/SW co-design in [36], our high-performance processor is 6.0x faster in speed. The disadvantage of the co-design is that the coefficient transmission between the RISC-V core and the memory consumes too many clock cycles.

The hardware module in [27] is the fastest core in the provided schemes. They used 8 DSPs to build 4 parallel butterfly cores, which greatly reduced the clock cycles of NTT. However, They did not propose the optimization for order-reverse operation. In order to reduce the latency, both participants of the key exchange could omit the order-reverse operation by reach an agreement in advance. The efficiency of [27] is relatively high considering that the design is free of BRAM block.

The implementation of NewHope in [37] tries to improve performance by adding up to 4 parallel paths. As the cost of high-speed, 8 DSP slices are occupied in NTT and reduction module. Although we use only one DSP in the CBU, we make full use of the register resource and configure it into multiple operation modes. The utilization of DSP in [37] is inferior to our implementation. Moreover, our processor achieves 4.2x LUT-efficiency and 2.6x FF-efficiency.

The implementation of NewHope-Simple in [28] uses one DSP for NTT and another DSP for multipurpose. Although they also use CT butterfly for forward NTT and GS butterfly for inverse NTT to avoid $O(n)$ cycles for order-reverse operations, the clock cycles for the time-consuming modular reduction and memory accessing are massive burdens. Their butterfly operation costs 1 clock cycle on multiplication and leaves 4 for modular reduction and 2 for memory access so that a total of $7 \times \frac{n}{2} \log(n)$ are used. Their modular reduction module is not further optimized and brings high latency. As an advantage, our processor reaches approximately 2x maximum frequency and achieves 16.8x speed. The LUTs and FFs results for [28] are obtained by re-synthesizing their open source code in Vivado 2018.2. Since the area of different modules can not be simply added up, we only show the area of independent forward NTT module. Since we use $no \rightarrow bo$ NTT flow in both CT and GS butterfly to reuse control logic, the area of our multi-purpose module is comparable to [28].

### *6.2 Comparisons of polynomial multipliers*

Polyvec multiplication is a computation-intensive task in Kyber algorithm, and the polynomial multiplication plays a similar role in Ring-LWE-based algorithms such as NewHope. The total latency of polynomial multiplication

**Table 5**
Comparison between hardware implementations of polynomial multiplication.

| Processor | Device | Parameter | Area | | | | Time | Throughput/Area | |
|---|---|---|---|---|---|---|---|---|---|
| | | (n × k × times) | LUT | FF | DSP | BRAM36/18 | ($\mu s$) | Kbps/LUT | Kbps/FF |
| **This Kyber1024** | Artix-7 | 256 × 4 × 1 | 533 | 514 | 1 | 3/0 | 55.7 | 438 | 454 |
| Kyber1024 [29] | Artix-7 | 256 × 4 × 1 | 477 | 237 | 1 | 3/0 | 118.5 | 230 | 463 |
| **This Kyber512** | Artix-7 | 256 × 2 × 2 | 479 | 472 | 1 | 2/0 | **54.5** | **498** | **505** |
| Kyber512 [29] | Artix-7 | 256 × 2 × 2 | 442 | 237 | 1 | 3/0 | 113.6 | 259 | 483 |
| NewHope [26] | Zynq-7000 | 1024 × 1 × 1 | 524[a] | 823[a] | 3[a] | 0/6[a] | 88.5 | **302** | **192** |
| NewHope-USENIX [27] | Zynq-7000 | 1024 × 1 × 1 | 2942 | 4881 | 8 | 0/0 | **54.4** | 88 | 53 |
| NewHope-Simple [28] | Artix-7 | 1024 × 1 × 1 | 432[b] | 278[b] | 2[b] | 4/0[b] | 1007.8 | 32 | 50 |
| Ring-LWE [21] | Spartan-6 | 256 × 1 × 4 | 1438 | 1123 | 1 | 0/2.5 | 91.4 | **129** | **166** |
| Ring-LWE Seq[c][38] | Zynq UltraScale+ | 256 × 1 × 4 | 2999 | 4281 | 10 | 2/0 | 145.5 | 32 | 22 |
| Ring-LWE SA[c][38] | Zynq UltraScale+ | 256 × 1 × 4 | 2372 | 2230 | 48 | 24/0 | 86.2 | 68 | 73 |
| Ring-LWE CONV[c][38] | Zynq UltraScale+ | 256 × 1 × 4 | 47332 | 38108 | 1282 | 2/0 | **21.3** | 14 | 17 |

[a] The area is the sum of polynomial multiply, polynomial bit-reversal and pipelined NTT module.
[b] The area is the sum of forward NTT and pointwise multiply module.
[c] The data width is regarded as 24-bit, which can adapt to the signature algorithms such as CRYSTALS-Dilithium.

is considered as the sum of 2 forward NTT, 1 point-wise multiply, 1 inverse NTT, as well as pre-scale and post-scale. For Kyber, the latency of the proposed processor is counted as the sum of 2 complete PV_NTT, 1 complete PV_INTT and 1 complete PV_PWM commands. Table 5 shows the comparison result among polynomial multipliers. Because some miscellaneous items are ignored in the comparisons of NTT operations, the following analysis shows the effect of this work from a more comprehensive perspective. Note that we execute the $\mathrm{PwM}_{\psi^{-i}}$ operation in the polyvec addition, and the polyvec multiplication disregards the consumption of these clock cycles for Kyber implementations. Similarly, the clock period of polynomial addition is not included for NewHope implementations.

The processor in [29] merges the $\mathrm{PwM}_{\psi^{-i}}$ operation into INTT computation. However, the $\mathrm{PwM}_{\psi^{i}}$ needs to be performed when loading coefficients, since GS butterfly is used in forward NTT. This work utilizes the CBU to further reduce clock cycles and ends up with a 2.1x performance improvement.

The multipurpose module in [26] provides the best LUT-efficiency in NTT comparisons. However, our proposed work shows advantages in this comparison. The first reason is that the comparisons of polynomial multipliers are more objective because it not only considers the area of NTT module. Another reason is that we avoid order-reverse operations, which saves 3×n×k clock cycles. Similarly, for the fastest module [27] in the NTT competition, the order-reverse operations almost offset its advantages. For the implementation omitting order-reverse operations in [27], the speed and efficiency can be increased to twice, but its circuit efficiency is still far lower than ours.

Compared with early Ring-LWE implementations [21], LUT-efficiency and FF-efficiency of this work are at least 3.4x and 2.8x higher, respectively. The actual advantage can be greater because we also integrate multiple commands over polynomial.

The work in [38] provides three architectures to adapt to various applications from resource-constrained devices to high-performance servers with the commercial high level synthesis (HLS) technology. [38] suggests that systolic array NTT-based (SA) polynomial multiplier is better than the sequential NTT-based (Seq) architecture considering speed and energy consumption. However, our implementation is still 6.5x more efficient and 1.55x faster in comparison to their best architecture. The convolution-based (CONV) polynomial multiplier achieves extremely high speed. However, the efficiency of CONV is only about 3.71% of this work.

# 7. Conclusion

Public key cryptography plays an important role in security and privacy. In this paper, a 240MHz polynomial ring processor is proposed for CRYSTALS-Kyber, a Module-LWE-based post-quantum cryptography. The processor executes optimized polynomial ring arithmetic, which saves over 20%/50% on the number of modular multiplication/addition. Moreover, optimized NTT using no $\rightarrow$ bo signal flow in both forward and inverse transformation minimizes the control circuit taking advantage of a novel CBU. To increase memory throughput, the dual-column sequential storage scheme is adopted in a simple dual-port block RAM. Performance evaluation on an Artix-7 FPGA shows the pipelined processor achieves 2.1x speedup compared with the state-of-the-art implementation.

Our implementation focuses on the hardware optimization and the trade-off of speed and efficiency. However, some research shows that NTT based polynomial multiplication can be the target of template attack [39]. Fortunately, some resistance schemes have been proposed [40], which can also be applied to our processors. We believe that practical countermeasures for Kyber against side-channel attacks are still an open question.

# References


[1] W. Diffie, M. E. Hellman, New directions in cryptography, IEEE Trans. Inf. Theory 22 (6) (1976) 644–654. https://doi.org/10.1109/TIT.1976.1055638.

[2] P. W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, SIAM J. Comput. 26 (5) (1997) 1484–1509. https://doi.org/10.1137/S0097539795293172.

[3] K. Basu, D. Soni, M. Nabeel, R. Karri, NIST post-quantum cryptography- A hardware evaluation study, IACR Cryptol. ePrint Arch. 2019 (2019) 47.

[4] H. Nejatollahi, N. D. Dutt, S. Ray, F. Regazzoni, I. Banerjee, R. Cammarota, Post-quantum lattice-based cryptography implementations: A survey, ACM Comput. Surv. 51 (6) (2019) 129:1–129:41. https://doi.org/10.1145/3292548.

[5] J. W. Bos, C. Costello, L. Ducas, I. Mironov, M. Naehrig, V. Nikolaenko, A. Raghunathan, D. Stebila, Frodo: Take off the ring! practical, quantum-secure key exchange from LWE, in: E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, S. Halevi (Eds.), Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016, ACM, 2016, pp. 1006–1018. https://doi.org/10.1145/2976749.2978425.

[6] O. Regev, On lattices, learning with errors, random linear codes, and cryptography, in: Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005, 2005, pp. 84–93. https://doi.org/10.1145/1060590.1060603.

[7] V. Lyubashevsky, C. Peikert, O. Regev, On ideal lattices and learning with errors over rings, in: Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings, 2010, pp. 1–23. https://doi.org/10.1007/978-3-642-13190-5_1.

[8] C. Peikert, How (not) to instantiate Ring-LWE, in: Security and Cryptography for Networks - 10th International Conference, SCN 2016, Amalfi, Italy, August 31 - September 2, 2016, Proceedings, 2016, pp. 411–430. https://doi.org/10.1007/978-3-319-44618-9_22.

[9] A. Langlois, D. Stehlé, Worst-case to average-case reductions for module lattices, Des. Codes Cryptography 75 (3) (2015) 565–599. https://doi.org/10.1007/s10623-014-9938-4.

[10] J. W. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, 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, 2018, pp. 353–367. https://doi.org/10.1109/EuroSP.2018.00032.

[11] E. Alkim, L. Ducas, T. Pöppelmann, P. Schwabe, Post-quantum key exchange - A new hope, 25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016. (2016) 327–343.

[12] F. Valencia, A. Khalid, E. O'Sullivan, F. Regazzoni, The design space of the number theoretic transform: A survey, in: 2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, SAMOS 2017, Pythagorion, Greece, July 17-20, 2017, 2017, pp. 273–277. https://doi.org/10.1109/SAMOS.2017.8344640.

[13] T. Pöppelmann, L. Ducas, T. Güneysu, Enhanced lattice-based signatures on reconfigurable hardware, in: Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings, 2014, pp. 353–370. https://doi.org/10.1007/978-3-662-44709-3_20.

[14] C. Du, G. Bai, 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, 2016, pp. 9–14. https://doi.org/10.1145/2902961.2902969.

[15] R. Agrawal, L. Bu, A. Ehret, M. A. Kinsy, Open-source FPGA implementation of post-quantum cryptographic hardware primitives, in: I. Sourdis, C. Bouganis, C. Álvarez, L. A. T. Díaz, P. Valero-Lara, X. Martorell (Eds.), 29th International Conference on Field Programmable Logic and Applications, FPL 2019, Barcelona, Spain, September 8-12, 2019, IEEE, 2019, pp. 211–217. https://doi.org/10.1109/FPL.2019.00040.

[16] H. Nejatollahi, R. Cammarota, N. D. Dutt, Flexible NTT accelerators for RLWE lattice-based cryptography, in: 37th IEEE International Conference on Computer Design, ICCD 2019, Abu Dhabi, United Arab Emirates, November 17-20, 2019, IEEE, 2019, pp. 329–332. https://doi.org/10.1109/ICCD46524.2019.00052.

[17] A. C. Mert, E. Öztürk, E. Savas, Design and implementation of a fast and scalable NTT-based polynomial multiplier architecture, in: 22nd Euromicro Conference on Digital System Design, DSD 2019, Kallithea, Greece, August 28-30, 2019, IEEE, 2019, pp. 253–260. https://doi.org/10.1109/DSD.2019.00045.

[18] A. Aysu, C. Patterson, P. Schaumont, Low-cost and area-efficient FPGA implementations of lattice-based cryptography, in: 2013 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2013, Austin, TX, USA, June 2-3, 2013, IEEE Computer Society, 2013, pp. 81–86. https://doi.org/10.1109/HST.2013.6581570.

[19] T. Pöppelmann, T. Güneysu, Towards practical lattice-based public-key encryption on reconfigurable hardware, in: Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers, 2013, pp. 68–85. https://doi.org/10.1007/978-3-662-43414-7_4.

[20] S. S. Roy, F. Vercauteren, N. Mentens, D. D. Chen, 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, 2014, pp. 371–391. https://doi.org/10.1007/978-3-662-44709-3_21.

[21] T. Pöppelmann, 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, 2012, pp. 139–158. https://doi.org/10.1007/978-3-642-33481-8_8.

[22] T. Pöppelmann, T. Oder, T. Güneysu, High-performance ideal lattice-based cryptography on 8-bit atxmega microcontrollers, in: K. E. Lauter, F. Rodríguez-Henríquez (Eds.), Progress in Cryptology - LATINCRYPT 2015 - 4th International Conference on Cryptology and Information Security in Latin America, Guadalajara, Mexico, August 23-26, 2015, Proceedings, Vol. 9230 of Lecture Notes in Computer Science, Springer, 2015, pp. 346–365. https://doi.org/10.1007/978-3-319-22174-8_19.

[23] P. Longa, 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, 2016, pp. 124–139. https://doi.org/10.1007/978-3-319-48965-0_8.

[24] U. Banerjee, T. S. Ukyab, A. P. Chandrakasan, Sapphire: A configurable crypto-processor for post-quantum lattice-based protocols, IACR Trans. Cryptogr. Hardw. Embed. Syst. 2019 (4) (2019) 17–61. https://doi.org/10.13154/tches.v2019.i4.17-61.

[25] N. Zhang, B. Yang, C. Chen, S. Yin, S. Wei, L. Liu, Highly efficient architecture of NewHope-NIST on FPGA using low-complexity NTT/INTT, IACR Trans. Cryptogr. Hardw. Embed. Syst. 2020 (2) (2020) 49–72. https://doi.org/10.13154/tches.v2020.i2.49-72.

[26] A. Jati, N. Gupta, A. Chattopadhyay, S. K. Sanadhya, Spqcop: Side-channel protected post-quantum cryptoprocessor, IACR Cryptol. ePrint Arch. 2019 (2019) 765.

[27] Y. Xing, S. Li, An efficient implementation of the NewHope key exchange on FPGAs, IEEE Trans. Circuits Syst. I Regul. Pap. 67-I (3) (2020) 866–878. https://doi.org/10.1109/TCSI.2019.2956651.

[28] T. Oder, T. Güneysu, Implementing the NewHope-Simple key exchange on low-cost fpgas, in: T. Lange, O. Dunkelman (Eds.), Progress in Cryptology - LATINCRYPT 2017 - 5th International Conference on Cryptology and Information Security in Latin America, Havana, Cuba, September 20-22, 2017, Revised Selected Papers, Vol. 11368 of Lecture Notes in Computer Science, Springer, 2017, pp. 128–142. https://doi.org/10.1007/978-3-030-25283-0_7.

[29] Z. Chen, Y. Ma, T. Chen, J. Lin, 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, IEEE, 2020, pp. 247–252. https://doi.org/10.1109/ASP-DAC47756.2020.9045459.

[30] N. Göttert, T. Feller, M. Schneider, J. A. Buchmann, S. A. Huss, On the design of hardware building blocks for modern lattice-based encryption schemes, in: E. Prouff, P. Schaumont (Eds.), Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings, Vol. 7428 of Lecture Notes in Computer Science, Springer, 2012, pp. 512–529. https://doi.org/10.1007/978-3-642-33027-8_30.

[31] W. Liu, S. Fan, A. Khalid, C. Rafferty, M. O'Neill, Optimized schoolbook polynomial multiplication for compact lattice-based cryptography on FPGA, IEEE Trans. Very Large Scale Integr. Syst. 27 (10) (2019) 2459–2463. https://doi.org/10.1109/TVLSI.2019.2922999.

[32] P. Barrett, Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor, in: A. M. Odlyzko (Ed.), Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings, Vol. 263 of Lecture Notes in Computer Science, Springer, 1986, pp. 311–323. https://doi.org/10.1007/3-540-47721-7_24.

[33] A. Bosselaers, R. Govaerts, J. Vandewalle, Comparison of three modular reduction functions, in: Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings, 1993, pp. 175–186. https://doi.org/10.1007/3-540-48329-2_16.

[34] Z. Liu, H. Seo, S. S. Roy, J. Großschädl, H. Kim, I. Verbauwhede, Efficient Ring-LWE encryption on 8-bit AVR processors, in: Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings, 2015, pp. 663–682. https://doi.org/10.1007/978-3-662-48324-4_33.

[35] L. Botros, M. J. Kannwischer, P. Schwabe, Memory-efficient high-speed implementation of Kyber on Cortex-M4, in: J. Buchmann, A. Nitaj, T. Rachidi (Eds.), Progress in Cryptology - AFRICACRYPT 2019 - 11th International Conference on Cryptology in Africa, Rabat, Morocco, July 9-11, 2019, Proceedings, Vol. 11627 of Lecture Notes in Computer Science, Springer, 2019, pp. 209–228. https://doi.org/10.1007/978-3-030-23696-0_11.

[36] T. Fritzmann, U. Sharif, D. Müller-Gritschneder, C. Reinbrecht, U. Schlichtmann, J. Sepúlveda, Towards reliable and secure post-quantum co-processors based on RISC-V, in: J. Teich, F. Fummi (Eds.), Design, Automation & Test in Europe Conference & Exhibition, DATE 2019, Florence, Italy, March 25-29, 2019, IEEE, 2019, pp. 1148–1153. https://doi.org/10.23919/DATE.2019.8715173.

[37] P. Kuo, W. Li, Y. Chen, Y. Hsu, B. Peng, C. Cheng, B. Yang, Post-quantum key exchange on fpgas, IACR Cryptol. ePrint Arch. 2017 (2017) 690.

[38] H. Nejatollahi, S. Shahhosseini, R. Cammarota, N. D. Dutt, Exploring energy efficient quantum-resistant signal processing using array processors, IACR Cryptol. ePrint Arch. 2019 (2019) 1297.

[39] R. Primas, P. Pessl, S. Mangard, Single-trace side-channel attacks on masked lattice-based encryption, in: W. Fischer, N. Homma (Eds.), Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings, Vol. 10529 of Lecture Notes in Computer Science, Springer, 2017, pp. 513–533. https://doi.org/10.1007/978-3-319-66787-4_25.

[40] T. Zijlstra, K. Bigou, A. Tisserand, FPGA implementation and comparison of protections against scas for RLWE, in: F. Hao, S. Ruj, S. S. Gupta (Eds.), Progress in Cryptology - INDOCRYPT 2019 - 20th International Conference on Cryptology in India, Hyderabad, India, December 15-18, 2019, Proceedings, Vol. 11898 of Lecture

Notes in Computer Science, Springer, 2019, pp. 535–555.
https://doi.org/10.1007/978-3-030-35423-7_27.