Area Efficient Polynomial Arithmetic Accelerator for Post-Quantum Digital Signatures and KEMs

Dina Kamel and François-Xavier Standaert

UCLouvain, ICTEAM, Crypto Group, Louvain-la-Neuve, Belgium
`{dina.kamel, fstandae}@uclouvain.be`

Abstract. Cryptographic schemes relying on Lattice-based hard learning problems are popular options for post-quantum signature and key encapsulation. This is for example witnessed by the selection of CRYSTALS-Dilithium and CRYSTALS-Kyber as new standards by the National Institute for Standards and Technology (NIST). Many other algorithms are currently being considered by the scientific community. All lattice-based algorithms rely on polynomial operations, among which the polynomial multiplication is generally one of the most expensive from the implementation viewpoint. As a result, the Number Theoretic Transform (NTT) is very frequently considered to speed up the implementations of these algorithms. For this purpose, we propose a semi-generic lightweight hardware architecture that supports polynomial operations for multiple lattice-based schemes, namely Dilithium, Hawk, Raccoon, Kyber and Polka. Implementation results on an Artix-7 FPGA show that our design features a relatively small footprint compared to state-of-the-art implementations. For example, our polynomial arithmetic core requires 2604 LUTs, 770 FFs and 4 DSPs for Dilithium and 1583 LUTs, 458 FFs and 2 DSPs for Kyber and can operate at 100 MHz. It computes NTT/INTT, point-wise-multiplication, multiply-accumulate and addition/subtraction in 519, 134, 135 and 131 clock cycles for Dilithium and in 455, 134, 135 and 131 clock cycles for Kyber, respectively.

Keywords: Kyber · Dilithium · HAWK · Raccoon · Polka · Number theoretic transform (NTT) · Polynomial arithmetic · Lightweight design · FPGAs

1 Introduction

Post-Quantum Cryptography (PQC) has gained a significant momentum in recent years to match the advancements on the development of quantum computers [24]. Indeed, implementing Shor's algorithm [37] on quantum computers can break current public key cryptosystems (e.g Rivest-Shamir-Adleman (RSA) and Elliptic Curve Cryptography (ECC)) that rely on the hardness of integer factorization and discrete logarithms [9]. In 2016, the National Institute for Standards and Technology (NIST) launched a call for standardization of new post-quantum public key algorithms, covering both public-key encryption and digital signatures. The lattice-based CRYSTALS-Dilithium [4] is one of three digital signature schemes selected for standardization in 2022. CRYSTALS-Kyber [2], which belongs to the same CRYSTALS family as Dilithium, was the only selected scheme for Key Encapsulation Mechanism (KEM). On top of that, the NIST posted a call for additional signature proposals to be considered in the PQC standardization process to diversify its post-quantum signatures portfolio. The lattice-based signature schemes Hawk [8] and Raccoon [35] were submitted to this call and Hawk has been accepted for the second round.

All these lattice-based schemes use operations in the polynomial ring $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n+1)$, where n is the degree of the polynomial. One of the most costly operations is the multiplication of high-degree polynomials. The Number Theoretic Transform (NTT) reduces the complexity of high-order polynomial multiplication from $\mathcal{O}(n^2)$ (in case of direct school-book multiplication) to $\mathcal{O}(n \log n)$. As a result, most lattice-based schemes choose their parameters' sets to enable using the NTT allowing fast and efficient polynomial arithmetic computation. This leads the core module that handles all polynomial arithmetic to be quite similar. Hence, it suggests that having a generic design of a polynomial arithmetic module that can be tailored to the above-mentioned lattice-based schemes would be quite useful. At the same time, developing efficient implementations that satisfy a wide spectrum of applications from high-performance through mid-range to light-weight for different platforms (software and hardware) is a growing research field. In this work, we focus on light-weight hardware applications.

Many efficient hardware implementations of the polynomial arithmetic unit in the literature explore the trade off between the hardware cost and performance. Here are some examples in the case of Dilithium. Beckwith et al. [5] proposed a polynomial arithmetic unit featuring a radix-4 NTT that calculates two layers of NTT/INTT at a time. This allowed to reduce the latency and the cost of memory access while reordering coefficients during these operations to optimize the BRAM utilization. Similarly, Wang et al. [38] employed a radix-4 NTT to implement the polynomial multiplication. However, they opted for a conflict-free memory mapping scheme applied to four-bank Block RAMs. In contrast, Land et al. [23] opted for a radix-2 NTT that takes advantage of readily available DSPs on low-end FPGA. Another approach by Zhao et al. [43] is to use a radix-2 multipath delay commutator (R2MDC) NTT architecture that has fewer memory accesses and a simpler control logic compared to in-place NTT architectures. Gupta et al. [17] on the other hand used two dual-port RAMs in their radix-2 NTT implementation to allow reading and writing their internal data in a ping-pong fashion. A different strategy was presented by Pham et al. [34] emphasizing effective hardware resource reuse and minimizing redundancies.

Similarly, there exists many works in the literature that target resource-constrained hardware applications for Kyber. For example, Ni et al. [33] presented a compact polynomial arithmetic module promoting a BRAM-free radix-2 NTT architecture where BRAM units are replaced with three smaller FIFOs. Both Nguyen et at. [31] and Xing et al. [39] adopted a similar approach. Zhang et al. [41] on the other hand implemented a ping-pong memory access scheme for their polynomial arithmetic module that uses a radix-2 NTT architecture.

Since Hawk is quite recently introduced, to our knowledge there has not been any hardware implementations for it yet in the literature. Besides the actual proposal, there has been one software implementation [15]. Nevertheless, as a lattice-based signature, the implementation of its polynomial arithmetic core is expected to be similar to that of Dilithium and Kyber.

Finally, and in order to cover a wider spectrum of algorithms, we also chose to implement Raccoon which is a lattice-based digital signature submitted to NIST in response to its call for additional digital signature schemes [35]. The reason behind our choice is the appealing argument of Raccoon being a masking-friendly scheme that should enable better resistance against side-channel analysis attacks [19]. This appears as a natural motivation given the significant cost of protecting Dilithium against leakage [3]. For a similar reason, we added Polka to our portfolio [18], which is a lattice-based encryption scheme developed to take side-channel leakage into account. The design is based, among others, on avoiding "leaky" functions such as the Fujisaki-Okamoto transform and adopting masking-friendly key-homomorphic computations.

Based on this state of the art, in this work, we therefore propose an efficient low-cost hardware design that is suitable to perform polynomial operations among which the complex NTT/INTT for any of the target lattice-based schemes mentioned above. Concretely, our main contributions are threefold:

- First, our polynomial arithmetic module design is semi-generic in a way that provides compile-time configurability for the scheme parameters allowing easy implementation of any of the aforementioned lattice-base standards. Depending on whether $\log n$ is even or odd and whether the choice of the prime modulus q allows a fully-splitting ring or not, only minor changes in the core implementation of the polynomial arithmetic unit will be required. Namely, the address generation of the polynomial coefficients and the twiddle factors in the address control logic are the target of such slight modifications. Besides, the modular reduction is not generic – hence the full design is only semi-generic. The reason behind this is the fact that customized optimizations are required to design the Barrett reduction module for each standard (as they obviously have different prime moduli) in order to minimize its resource utilization for our target low-cost applications.
- Second, our compact FPGA-based polynomial arithmetic architecture has a small area footprint for most implemented standards. On an Artix-7, our core uses 2604 Look-Up Tables (LUT)s, 770 Flip-Flops (FF)s and 4 Digital Signal Processor (DSPs) to implement Dilithium. For Hawk1024 and prime modulus p_2, our core occupies 4451 LUTs, 1139 FFs and 8 DSPs. In the case of Raccoon (using the prime modulus $q_2 = 33292289$), the resource utilization is 3458 LUTs, 998 FFs and 4 DSPs. For Kyber, our core uses 1583 LUTs, 458 FFs and 2 DSPs. Regarding Polka, it utilizes 2512 LUTs, 593 FFs and 2 DSPs. Our core operate at a maximum of 83 to 100 MHz depending on the standard.

– Third, we provide the Verilog code for our design at `https://git-crypto.elen.ucl.ac.be/dkamel/genericpolyarithunit` to support open source research, something currently lacking in the literature.

The rest of the paper is organized as follows. Section 2 introduces preliminaries. The proposed poly-arithmetic architecture and design details of its sub-blocks are presented in Section 3. Section 4 discusses the implementation results and compares with related work. Section 5 concludes the paper.

2 Preliminaries

2.1 Notations

We denote by $\mathbb{Z}_q$ the ring of integers modulo the prime q and by $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$ the polynomial ring in X modulo $X^n + 1$, with n the degree of the polynomial. We represent a polynomial with regular lowercase (e.g. a), a vector of polynomials with bold lowercase (e.g. $\boldsymbol{a}$) and a matrix of polynomials with bold uppercase (e.g. $\boldsymbol{A}$). The i-th coefficient of a polynomial is denoted by a_i. The $\cdot$ symbol denotes the multiplication operation whereas $\circ$ refers to the point-wise polynomial multiplication. Letters with a hat symbol correspond to their representation after NTT (e.g. $\hat{a}$).

2.2 Standards

Our design covers several lattice-based post-quantum standards for both digital signature and Key Encapsulation Mechanism (KEM).

CRYSTALS-Dilithium. Dilithium is a lattice-based digital signature recently standardized by NIST as ML-DSA in FIPS 204 [29] for secure Post-Quantum Cryptography (PQC) in 2024. Its hardness is based on the Module Learning With Errors (MLWE) and the Module Short Integer Solution problems. The signature scheme design is based on the “Fiat-Shamir with Aborts” paradigm proposed in [26, 27]. The initial proposal is described by Ducas et al. in [12] and later refined in the NIST PQC submission [4]. Its main characteristics are: randomness generation from a uniform distribution instead of a discrete Gaussian distribution which is difficult to implement securely and efficiently, [13], adhere the public key and signature sizes to a minimum, and easiness to vary the security level by changing the size of the module (dimensions of the matrices and vectors). Relying on an algebraic structured lattice (MLWE) problem rather than an ideal lattice (Ring-LWE) or completely unstructured lattice (LWE) problems was an optimal intermediate solution that moves further away from the weaknesses of ideal lattice problems while still profiting from their efficiency without the extra cost of using unstructured LWE [12]. As for the rational behind building the digital signature scheme using the “Fiat-Shamir with aborts” paradigm was to reduce the size of the mask randomness and thus the signature significantly [27]. An extra rejection sampling step is needed to perform the aborts. Dilithium uses a 23-bit prime modulus $q = 8380417$ and degree $n = 256$ for all security levels.

Hawk. Hawk is a lattice-based signature scheme whose hardness is based on the module Lattice Isomorphism Problem (LIP) that responded to the NIST PQC call for additional digital signature scheme. The scheme was first introduced in [13] and later optimized in [8]. The goal of this call was to diversify its post-quantum signature portfolio which is mostly based on structured lattices so far (namely; CRYSTALS-Dilithium and Falcon). Hawk has been recently selected to move forward to the second round of the standardization process. Its main features are: randomness generation is either from a centred binomial distribution (during key generation) or from uniform distribution (during signing), compact public key and signature sizes (even smaller than those of Dilithium), floating-point free arithmetic which enables its implementation on various (constrained) hardware devices, small memory footprint and no rejection-sampling. The basic idea was to combine the use of module lattice based on LIP and ideas from NTRUSign and Falcon in order to design a highly efficient signature scheme. Hawk uses two 31-bit primes $p_1 = 2147473409$ and $p_2 = 2147389441$. The degree n is either 512 or 1024 for security level I or IV, respectively.

Raccoon. As a response to the NIST PQC call for additional digital signature schemes Raccoon [35] was submitted, but was not selected in the second round. It is a masking-friendly lattice-based digital signature scheme based on the "Fiat-Shamir" paradigm. As Dilithium, its hardness is based on the MLWE problem. The main objective of Raccoon is to build a scheme that is inherently resistant against side-channel attacks by making its subroutines either masking friendly (with quasilinear overheads) or ones that do not need to be masked at all. This was motivated by the fact that even though the standardized lattice-based signatures Dilithium and Falcon and the hash-based signature SPHINCS are efficient and their black-box security is well-understood, they remain vulnerable against side-channel attacks if left unprotected [10, 16, 20, 21, 28] (as some examples). The cost of protecting these schemes using the masking countermeasure is extremely expensive. Indeed, lattice-based signatures contain subroutines (mainly the rejection loop and the hash functions) which when masked incur quadratic or worse than quadratic overheads, see for example [3]. Raccoon uses a 49-bit modulus $q = 549824583172097$ which is a composite number consisting of two primes: 24-bit $q_1 = 16515073$ and 25-bit $q_2 = 33292289$. The degree $n = 512$ for all security levels.

CRYSTALS-Kyber. Belonging to the same CRYSTALS family as Dilithium, Kyber is a lattice-based KEM recently standardized by NIST as ML-KEM in FIPS 203 [30] for post-quantum secure KEMs in 2022. It is based on the hardness of MLWE where a CPA-secure Public Key Encryption (PKE) scheme is used to create a CCA-Secure KEM by applying a variant of the Fujisaki-Okamoto (FO) transform. Its main characteristics are: secret and noise generation from a centred binomial distribution which is easily, efficiently, and securely sampled from, adopting an implicit rejection approach, using a compress function to discard some low-order bits in the ciphertext; thus reducing its size, and easiness to vary the security level by changing the size of the module (dimensions of the

matrices and vectors). Kyber uses a 12-bit modulus $q = 3329$ and a degree $n = 256$ for all security levels.
Polka. Polka [18] is a lattice-based encryption scheme that relies on the recently introduced Learning With Physical Rounding (LWPR) assumption [14]. As Raccoon, the main goal of Polka is to enable efficient side-channel protected implementations, but for encryption schemes. To do that, Polka leverages various features such as the rigidity property introduced by Bernstein and Persichetti [6] which allows avoiding the FO transform that proved to be a source of side-channel leakage. It also proposes to randomize the decryption process and adopts key-homomorphic computations that are easily masked with linear overheads. Polka uses a 16-bit modulus $q = 5939$ and a degree $n = 1024$ that satisfies security level I.

2.3 Number Theoretic Transform

All the before-mentioned standards (and others) share the fact that their main algebraic operation is high-order polynomial multiplication (whether on a matrix, vector or a single polynomial). The NTT is the most efficient method for multiplying two high-order polynomials where the complexity is reduced from $\mathcal{O}(n^2)$ in case of school-book multiplication for example to $\mathcal{O}(n \log n)$ in case of NTT. The NTT is the special case of Discrete Fourier Transform (DFT) over finite-field polynomials in $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$. This ring structure enables the implementation of the Negative Wrapped Convolution-based (NWC) NTT effectively. Accordingly, n is a power of two and the prime modulus q satisfies $q \equiv 1 \bmod 2n$, such that the primitive $2n$-th root of unity ζ in $\mathbb{Z}_q$ exists; thus allowing a “fully-splitting” of the NTT algorithm. The NTT transform can therefore be written as:

$$\hat{a}_j = \sum_{i=0}^{n-1} a_i \zeta^{(2j+1)i} \bmod q,\ j \in [0, n-1]$$

Conveniently, the inverse NTT (INTT) is also straightforward and can be written as:

$$a_i = n^{-1} \sum_{j=0}^{n-1} \hat{a}_j \zeta^{-(2i+1)j} \bmod q,\ i \in [0, n-1]$$

Now, one can compute polynomial multiplication efficiently using NTT as $a \cdot b = \mathrm{INTT}(\mathrm{NTT}(a) \circ \mathrm{NTT}(b))$. The radix-2 NTT is the simplest form of NTT where a polynomial of length n is split into two parts of length $n/2$ and this can go on recursively until the original polynomial is reduced to degree 0. One efficient algorithm is the Cooley-Tukey (CT) butterfly that takes advantage of the fact that $\zeta^n \equiv -1 \bmod q$. As a result it holds that $X^n + 1 = X^n - \zeta^n = (X^{\frac{n}{2}} - \zeta^{\frac{n}{2}}) \times (X^{\frac{n}{2}} + \zeta^{\frac{n}{2}}) \bmod q$. If this step is repeated $\log n$ times then the polynomial $X^n + 1$ can therefore be written as

$$X^n + 1 = \prod_{i=0}^{n-1} (X - \zeta^{2i+1}) = \prod_{i=0}^{n-1} (X - \zeta^{2brv_{\log n}(i)+1}),$$

where the $brv_{\log n}(i)$ is the bit reversal of the unsigned $\log n$-bit integer i. This essentially implies that if the coefficients of the polynomial are kept in natural

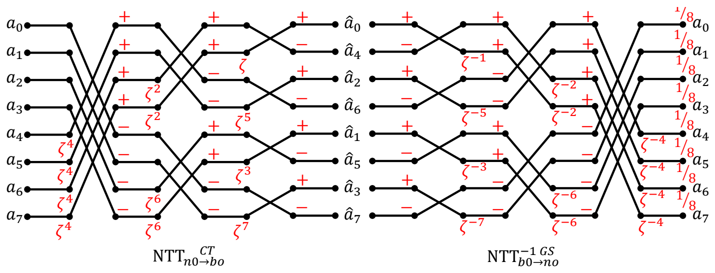

$\mathrm{NTT}_{no\rightarrow bo}^{CT}$
$\mathrm{NTT}_{bo\rightarrow no}^{-1\,GS}$

Fig. 1: Signal flow graph of radix-2 NTT/INTT for n = 8.

order (no), after the NTT operation they will be in bit-reversed order (bo). On the other hand, the INTT can be efficiently implemented using the Gentleman-Sande (GS) butterfly algorithm by inverting the mapping process. Keeping the inputs in bit-reversed order will result in naturally ordered coefficients after the INTT operation. This is quite convenient as it avoids the cost of reordering the polynomial coefficients [36]. Figure 1 shows the flow diagram for an 8-point radix-2 NTT and INTT using CT and GS algorithms, respectively.

2.4 NTT in Target Standards

The ring structure of Dilithium, Hawk, Raccoon and Polka were carefully chosen to enable the fully-splitting of the NWC NTT effectively without zero-padding. Accordingly, n is power of two and the prime modulus q satisfies $q \equiv 1 \bmod 2n$, such that the primitive $2n$-th root of unity ζ in $\mathbb{Z}_q$ exists. Therefore, the defining polynomial $X^n + 1$ of the ring $\mathcal{R}$ factors into n polynomials of degree 1 modulo q and the NTT of a polynomial $a \in \mathcal{R}_q$ is a vector of n polynomials of degree zero. Powers of ζ in the range $(0 : n-1)$ are referred to as twiddle factors. Polynomial multiplication can be efficiently computed using NTT as described in the previous section.

As for Kyber the prime modulus q is chosen to satisfy $q \equiv 1 \bmod n$ (instead of $q \equiv 1 \bmod 2n$), such that the primitive n-th root of unity (instead of $2n$-th root of unity) ζ in $\mathbb{Z}_q$ exists. The idea of decreasing the prime modulus was presented in [44] citing the main advantage as enabling the reduction of both the public key and the ciphertext sizes. As a result of this choice, the NTT algorithm cannot fully split. However, the defining polynomial $X^n + 1$ of the ring $\mathcal{R}$ factors into $n/2$ polynomials of degree 2 modulo q and the NTT of a polynomial $a \in \mathcal{R}_q$ is a vector of $n/2$ polynomials of degree one. This leads to powers of ζ being in the range $(0 : n/2 - 1)$. Polynomial multiplication can be computed using NTT as $a \cdot b = \mathrm{INTT}(\mathrm{NTT}(a) \circ \mathrm{NTT}(b))$. Nevertheless, $\mathrm{NTT}(a) \circ \mathrm{NTT}(b) = \hat{a} \circ \hat{b} = \hat{c}$ consists of the $n/2$ products in the form of $\hat{c}_{2i} + \hat{c}_{2i+1}X = (\hat{a}_{2i} + \hat{a}_{2i+1}X).(\hat{b}_{2i} + \hat{b}_{2i+1}X)\ mod(X^2 - \zeta^{2brv_{(\log n-1)}(i)+1})$, where i is the coefficient index and $brv_{(\log n-1)}$ is the bit reverse operation over $(\log n - 1)-$bits. As a result, an additional school-book multiplication is necessary to complete the polynomial multiplication.

For each of Dilithium, Kyber and Polka, the prime moduli used are all less than 31 bits which can easily fit on 32-bit platforms (mostly applicable to software). However in the case of Hawk and Raccoon, the moduli needed are larger than 32-bits. As a result, both schemes opted to apply the Chinese Remainder Theorem (CRT) and split the large modulus into two smaller primes to fit into 32-bit platforms, as explained in Section 2.2, then perform the necessary operations over these two primes.

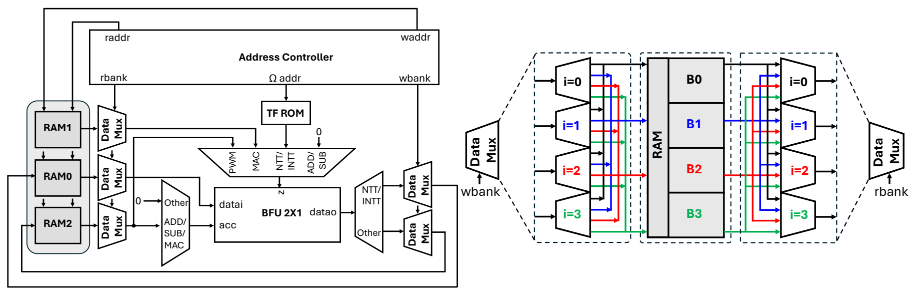
3 Proposed Poly-Arithmetic Architecture

In this section, we describe the architecture of our proposed semi-generic poly-arithmetic core which is responsible for the computation of all polynomial operations in the target standards.

3.1 Architecture overview

Figure 2a demonstrates the high-level architecture of the poly-arithmetic module. Our design consists of two butterfly units (BFUs) arranged in parallel (BFU 2X1), an address control unit with conflict-free memory access (thanks to an integrated address resolver block), a twiddle factor memory (TF ROM) as well as some data and control multiplexers. The proposed poly-arithmetic module is able to perform both radix-2 NTT and INTT as well as polynomial arithmetic operations such as pointwise multiplication (PWM), multiply and accumulate (MAC), addition (ADD) and subtraction (SUB) for all target standards. Additionally, the poly-arithmetic unit handles all interactions with the data RAMs where the polynomial coefficients are stored.

To meet the bandwidth requirement, the data RAMs are designed as a 4-bank memory block depicted in Figure 2b similar to [23,42]. The goal is to ensure that the four different polynomial coefficients accessed in parallel during NTT/INTT are always located in four different banks to guarantee a conflict-free memory access without having to shuffle and reorder the coefficients. Although a more efficient BRAM configuration that has a higher utilization of each memory row was proposed by [5], their final BRAM cost of a full Dilithium implementation at



(a) (b)

Fig. 2: (a) Proposed architecture and (b) internal RAM and data MUX structure.

security level 5 is slightly less than that of [23]. The banks are implemented with dual-port 36-kbit block RAMs capable of reading and writing in the same clock cycle. Each BRAM is configured as 1024×36 memory. This configuration can store up to 16 polynomials of degree $n = 256$, 8 polynomials of degree $n = 512$ or 4 polynomials of degree $n = 1024$. Although the utilization of each memory row is not maximized for all target standards (66%, 86%, 69% in case of Dilithium, Hawk and Raccoon, respectively and 33% and 44% in case of Kyber and Polka, respectively), for the sake of generality we decided to keep such configuration. For NTT, INTT a single data RAM is needed. The addition, subtraction and point-wise multiplication operations require two data RAMs where coefficients of two distinct polynomials are stored. As for the MAC operation, three data RAMs are employed. Two RAMs store the coefficients of two polynomials to be multiplied and the third RAM stores their product which is later accumulated as required by operations over vectors of polynomials. The data multiplexers/demultiplexers are internally divided into 4 blocks, each connected to a bank RAM and controlled by the address controller via the *rbank* and *wbank* signals for read and write operations, respectively. The TFs are stored in a distributed memory (TF ROM). Details of each block are provided in the following sections.

3.2 Dual Butterfly module

The butterflies arrangement in the BFU 2X1 block is illustrated in Figure 3. The two BFUs, each capable of performing both CT and GS butterfly operations as well as basic arithmetic operations such as multiplication, addition and subtraction, are placed in parallel. They process either 4 coefficients per clock cycle during NTT/INTT or 2 coefficients per clock cycle during all other operations when the pipeline is fulfilled. The multiplexers at the inputs and outputs provide the design with the flexibility to change between the operating modes depending on the 3-bit mode signal as explained in Table 1.

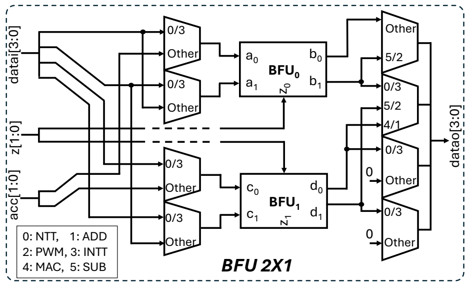


Fig. 3: Dual Butterfly unit.

3.3 Modular Reduction

The modular reduction is the main operation that has to be tailored to the prime modulus adopted by each standard. Different modular reduction techniques exist

Table 1: Dual butterfly unit in/out interconnections. 'X' represents a do not care state and '-' denotes an unconnected input.

	Operation	NTT	ADD	PWM	INTT	MAC	SUB
	mode[2:0]	0	1	2	3	4	5
In.	datai[0]	a_0	a_1	a_1	a_0	a_1	a_1
	datai[1]	a_1	c_1	c_1	a_1	c_1	c_1
	datai[2]	c_0	-	-	c_0	-	-
	datai[3]	c_1	-	-	c_1	-	-
	z[0]	z_0	X	z_0	z_0	z_0	X
	z[1]	z_1	X	z_1	z_1	z_1	X
	acc[0]	-	a_0	X	-	a_0	a_0
	acc[1]	-	c_0	X	-	c_0	c_0
Out.	datao[0]	b_0	b_0	b_1	b_0	b_0	b_1
	datao[1]	b_1	d_0	d_1	b_1	d_0	d_1
	datao[2]	d_0	0	0	d_0	0	0
	datao[3]	d_1	0	0	d_1	0	0

in the literature. The most common are the Montgomery and the Barrett reduction algorithms. However, both of these algorithms require additional multiplications, which are expensive in time and hardware resources. Beckwith et al [5] implemented Barrett reduction in hardware for Dilithium by only using shifts and additions. Nevertheless, their optimization method is highly customized to Dilithium's modulus using a complex Verilog code (available online [1]) and as a result difficult to reuse in case of other moduli. Another reduction technique recursively exploits the congruency relation within the prime modulus. For example in the case of Dilithium, $q = 2^{23} - 2^{13} + 1$, so by exploiting the relation $2^{23} \equiv 2^{13} - 1 \bmod q$ recursively as in [23]. Other reduction techniques, which are variants of Montgomery reduction, such as KRED, KRED-2X [25] and K^{2}RED [7] are also proposed in the literature. They require the modulus to be a Proth prime of the form $q = q_h 2^{\omega} + 1$, where $\omega > \log q/2$ which is not the case for all supported standards.

An optimized Barrett reduction implementation customized for a specific modulus using only addition, subtraction and shift operation was proposed in [22] and used for the Dilithium modulus in [34]. The basic principle of the original Barrett reduction is to subtract the multiplication result between the quotient $\lfloor U/q \rfloor$ and the modulus q from the input number U. To avoid the expensive division of U and q, $1/q$ can be replaced by $T/2^k$ which is just a right-shift operation and $T = \lfloor 2^k/q \rfloor$ such that:

$$D = (U \times T) >> k,$$
$$U \bmod q = U - D \times q,$$

Algorithm 1 explains the optimized version of the Barrett reduction as in [22]. The input U is split into two overlapping parts, where the upper value V and the lower value Y intersect in minimum two bits. Instead of multiplying the $2 \log q$-bit value U by T, the smaller upper value V, which is the most-significant bits (MSB)s of U replaces it then the product is scaled by $\approx 1/q$ $(1/2^{\lceil \log q \rceil + 1})$.

Algorithm 1 Optimized Barrett reduction [22, 34]

```
Require: $U$, $q$, $l = \lceil \log q \rceil$, $T = \lfloor 2^{2l}/q \rfloor$
Ensure: $Z = U \bmod q$
 1: $V = U >> (l - 1)$
 2: $W = (V \times T) >> (l + 1)$
 3: $X = (W \times q) \bmod 2^{l+1}$
 4: $Y = U \bmod 2^{l+1}$
 5: if $Y < X$ then
 6:     $Z = 2^{l+1} + Y - X$
 7: else
 8:     $Z = Y - X$
 9: end if
10: if $Z \geq q$ then
11:     $Z = Z - q$
12: end if
```

The scaled product is then multiplied by q as in the original Barrett reduction. Finally, the subtraction step is performed slightly different where the lower bits of this product is subtracted from the least-significant bits (LSBs) of the input Y. We follow the same path and optimize the modular reduction for each modulus of the target standards. Here we present the Barrett reduction of Dilithium and Kyber as examples due to space restrictions.

Dilithium Figure 4 illustrates the optimized Barrett reduction for Dilithium, i.e. for the specific prime $q = 8370417$, where the parameter $l = 23$ and the constant $T = 8396807$. The figure first shows a full multiplication between two 23-bit integers A and B which requires two DSPs. The DSP48 slices available on the target Artix7 FPGA allow multiplication between signed 25-bit and 18-bit values. As a result, to multiply two 23-bit values, two DSPs are required as well as a shift and an addition operation which are available within the DSP slice. The modular reduction then follows the steps in Algorithm 1. A highly optimized method to perform the multiplication by constants (T, q) using only addition, subtraction, xor and shift operations was detailed in [34] and presented here. The reduction operation is divided into two parts. The upper half multiplier (UH-MULT) sub-block handles the multiplication of the most-significant bits of the full multiplier $V = U[45:22]$ by the constant value T:

$$\begin{aligned}
V \times T &= V[23:0] \times (2^{23} + 2^{13} + 2^{2} + 2 + 1) \\
&= 2^{13}(2^{10}V[23:0] + V[23:0]) + 2(2V[23:0] + V[23:0]) + V[23:0] \\
&= 2^{13}(2^{10}V[23:0] + V[23:0]) + 2(2V[23:0] + V[23:0] + V[23:1]) \\
&\quad + V[0] \\
V_1[34:0] &= 2^{10}V[23:0] + V[23:0] \\
&= \mathit{concat}\{V[23:0] + V[23:10], V[9:0]\}
\end{aligned}$$

where, the concatenation from the MSB on the left to the LSB on the right is denoted by the *concat* function.

$$\begin{aligned}
V_2[25:0] &= 2V[23:0] + V[23:0] + V[23:1] \\
&= (V[23:0] << 1) + V[23:0] + V[23:1] \\
V \times T &= 2^{13} V_1[34:0] + 2V_2[25:0] + V[0] \\
&= 2(2^{12} V_1[34:0] + V_2[25:0]) + V[0] \\
&= concat\{V_1[34:0] + V_2[25:12], V_2[11:0], V[0]\} \\
V_{12} &= V_1[34:0] + V_2[25:12]
\end{aligned}$$

Now, the output of the UH-MULT is computed as:

$$\begin{aligned}
W[23:0] &= (V \times T) >> 24 \\
&= V_{12}[34:11]
\end{aligned}$$

Next, the lower half multiplier (LH-MULT) manages the multiplication of $W[23:0]$ by Dilithium prime modulus q.

$$\begin{aligned}
W \times q &= W[23:0] \times (2^{23} - 2^{13} + 1) \\
&= 2^{13}(2^{10} W[23:0] - W[23:0]) + W[23:0] \\
&= concat\{2^{10} W[23:0] - W[23:0] + W[23:10], W[12:0]\} \\
X[23:0] &= (W[23:0] \times q) \bmod 2^{24} \\
&= concat\{2^{10} W[0] - W[10:0] + W[23:13], W[12:0]\} \\
W_s[10:0] &= W[23:13] - W[10:0]
\end{aligned}$$

Then, the output of the LH-MULT is computed as:

$$\begin{aligned}
X[23:0] &= concat\{2^{10} W[0] + W_s[10:0], W[12:0]\} \\
&= concat\{W[0] \oplus W_s[10], W_s[9:0], W[12:0]\}
\end{aligned}$$

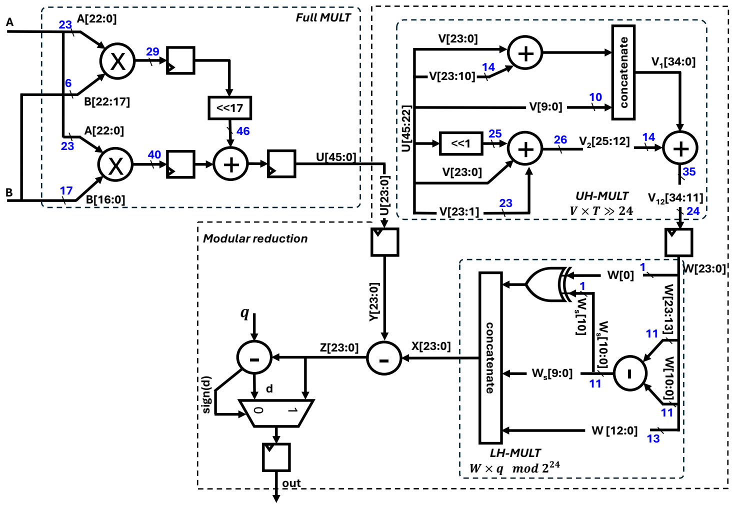


Fig. 4: Dilithium modular reduction module [34].

The X value in fact represents the multiples of q that is closest to the LSB of the input to the reduction module Y. Finally, X is subtracted from Y at the output of the LH-MULT sub-block and stored in Z. A multiplexer is needed in case $Z \geq q$. The design consists of 4 pipeline stages, the first two are deployed within the full multiplier (inside the DSPs). The third pipeline stage is located at the output of the UH-MULT sub-block. To respect the timing, a pipeline stage is added on the LSB of the input to the reduction module $U[23:0]$. The fourth and final pipeline stage is placed at the output of the modular reduction block.

Kyber Similar to the Barrett reduction for Dilithium, the one for Kyber is optimized for the specific prime $q = 3329$, where the parameter $l = 12$ and the constant $T = 5039$. Figure 5 first shows a full multiplication between two 12-bit integers A and B which takes place within a single DSP slice. The modular reduction then follows the steps in Algorithm 1. Using the same optimization techniques as in the Barrett reduction of Dilithium, multiplication by constants (T, q) needs only addition, subtraction, xor and shift operations.

The reduction operation is divided into two parts. The upper half multiplier (UH-MULT) sub-block handles the multiplication of the most-significant bits of the full multiplier $V = U[23:11]$ by the constant value T:

$$\begin{aligned}
V \times T &= V[12:0] \times (2^{12} + 2^9 + 2^8 + 2^7 + 2^5 + 2^3 + 2^2 + 2 + 1) \\
&= 2^5(2^7V[12:0] + V[12:0]) + 2^2(2^7V[12:0] + V[12:0]) \\
&\quad + 2(2^7V[12:0] + V[12:0]) + (2^7V[12:0] + V[12:0]) + 2^3V[12:0] \\
V_1[20:0] &= 2^7V[12:0] + V[12:0] \\
&= \mathit{concat}\{V[12:0] + V[12:7], V[6:0]\} \\
V \times T &= 2^5V_1[20:0] + 2^2V_1[20:0] + 2V_1[20:0] + V_1[20:0] + 2^3V[12:0] \\
&= 2^5V_1[20:0] + 2(2V_1[20:0] + V_1[20:0] + V_1[20:1]) + V_1[0] \\
&\quad + 2^3V[12:0] \\
V_2[22:0] &= 2V_1[20:0] + V_1[20:0] + V_1[20:1] \\
&= (V_1[20:0] << 1) + V_1[20:0] + V_1[20:1] \\
V \times T &= 2^5V_1[20:0] + 2V_2[22:0] + V_1[0] + 2^3V[12:0] \\
&= \mathit{concat}\{V_1[20:0] + V_2[22:4], V[12:2], V[1:0] + V_2[3:2], V_2[1:0], \\
&\quad V_1[0]\} \\
V_{12} &= V_1[20:0] + V_2[22:4] + V[12:2]
\end{aligned}$$

Now, the output of the UH-MULT is computed as:

$$\begin{aligned}
W[12:0] &= (V \times T) >> 13 \\
&= V_{12}[21:8]
\end{aligned}$$

Next, the lower half multiplier (LH-MULT) manages the multiplication of $W[12:0]$ by Kyber prime modulus q.

$$
\begin{aligned}
W \times q &= W[23:0] \times (2^{12} - 2^{9} - 2^{8} + 1) \\
&= 2^{8}(2^{4}W[12:0] - 2W[12:0] - W[12:0]) + W[12:0] \\
&= concat\{2^{4}W[12:0] - 2W[12:0] - W[12:0] + W[12:8], W[7:0]\} \\
X[12:0] &= (W[12:0] \times q) \bmod 2^{13} \\
&= concat\{2^{4}W[0] - 2W[3:0] - W[4:0] + W[12:8], W[7:0]\} \\
W_s[4:0] &= W[12:8] - 2W[3:0] - W[4:0]
\end{aligned}
$$

Then, the output of the LH-MULT is computed as:

$$
\begin{aligned}
X[12:0] &= concat\{2^{4}W[0] + W_s[4:0], W[7:0]\} \\
&= concat\{W[0] \oplus W_s[4], W_s[3:0], W[7:0]\}
\end{aligned}
$$

The X value in fact represents the multiples of q that is closest to the LSB of the input to the reduction module Y. Finally, X is subtracted from Y at the output of the LH-MULT sub-block and stored in Z. A multiplexer is needed in case $Z \geq q$. The design consists of 4 pipeline stages as in the Barrett reduction of Dilithium. In order to preserve timing between the different standards' implementations, two pipeline stages are used inside the full multiplier even though, one would have been enough in the case of Kyber. The third and fourth pipeline stages are placed similar to the Dilithium Barrett reduction module.

3.4 Butterfly Unit

A compact fully pipelined butterfly architecture that supports all polynomial operations leveraging resource sharing is demonstrated in Figure 6. It is inspired by the work of Pham et al [34] where one of the four deployed computational elements efficiently implements the CT and the GS butterfly operations as well as the remaining arithmetic operations, such as addition, subtraction, pointwise

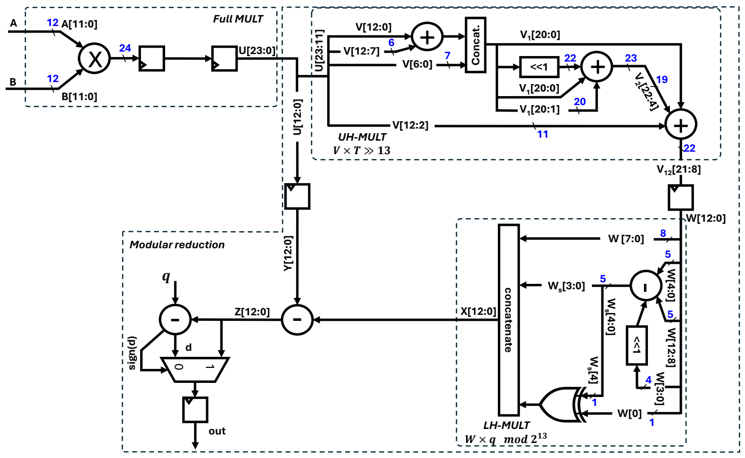


Fig. 5: Kyber modular reduction module.

multiplication and multiply-accumulate. The BFU unit is configured to operate in 4 modes (NTT/MAC, ADD/SUB, PWM and INTT) using only the 2 LSBs of the mode signal controlling the BFU 2X1 module. Since the MAC operation is basically the same as one NTT butterfly operation, but with outputs only from b_0 and not b_0 and b_1 so they both share mode[1:0] of 0. Similarly, the ADD and SUB operations are both done internally only outputed on different outputs and they have the same mode[1:0] = 1 within the BFU block. They only differ at the BFU 2X1 level where we need the full 3-bit mode to route the correct outputs. These different modes of operations are facilitated by the integration of 9 q-bit 2-MUXs. The butterfly module also comprises a modular reduction block after the multiplication is performed. It uses the optimized form of Barrett reduction as detailed in Section 3.3. Instead of multiplying with 256^{-1} in the last step of the INTT, a divide-by-2 operation is incorporated inside the BFU module. This division operation only requires low-cost shift, addition and multiplexer blocks as explained in [42]. For all target standards, the BFU block is instantiated in the same way, only the sizes of the signals are adapted to each standard's data width and the corresponding Barrett reduction block is called explicitly for each standard. The BFU block performs its operations as follows. For NTT, the $b_0 = a_0 + a_1 \cdot z_0 \bmod q$ and $b_1 = a_0 - a_1 \cdot z_0 \bmod q$, where a_0 and a_1 are the input coefficients and the twiddle factor (stored in a distributed memory in the order of its fetch request) is applied to the z_0 input. The INTT operation is carried out as $b_0 = (a_0 + a_1)/2 \bmod q$ and $b_1 = (a_0 - a_1) \cdot z_0^{-1}/2 \bmod q$. In this case to compute z_0^{-1}, the twiddle factor is loaded from the memory in reverse order (at the layer level) and subtracted from q, allowing the reuse of the twiddle factors of NTT. This is inline with the symmetry property of the twiddle factor $\zeta^{k+n} = -\zeta^k$ where k is an integer. For modular multiplication, the factors are the inputs a_1 and z_0, whereas input a_0 is only used to accumulate the product of previous multiplications in case of the MAC mode (used for matrix-vector and vector-vector polynomial multiplications). As for modular addition and subtraction, the polynomial coefficients are entered through the inputs a_0 and a_1 that have direct access to the adder/subtracter blocks through a few independent multiplexers and FFs. Similar to the NTT, INTT and multiplication

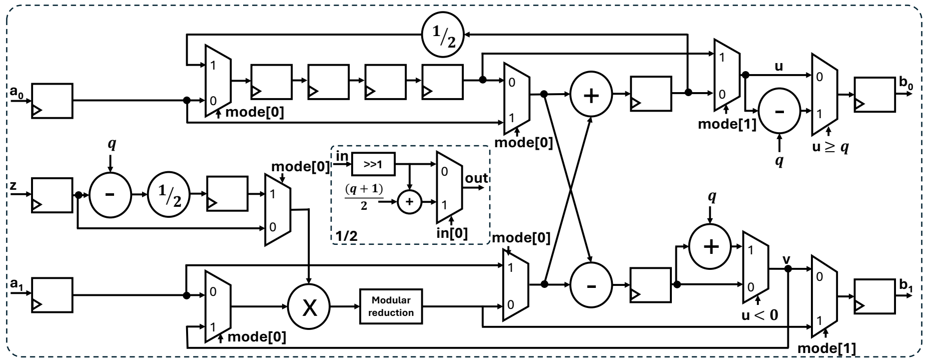


Fig. 6: Butterfly unit block diagram.

operations, a few extra registers are required to balance the pipeline latency. For NTT, INTT and MAC operations, the latency is 7 clock cycles. As for PWM, ADD and SUB operations, the latency is 6, 3 and 3 clock cycles, respectively. The cost of NTT/INTT operations is $n/4 \cdot \log n + c$ clock cycles whereas that of the other operations is $n/2 + c$, where c is the latency.

3.5 Address Controller

The address controller module is responsible for generating the read/write addresses of the polynomial coefficients for all six operations as well as the read addresses of the twiddle factors in the case of NTT/INTT. It also provides the select signals to the data and address multiplexers connected to the external RAM blocks. It supports all target standards by simple and inexpensive tweaks. In the case of NTT/INTT, two main factors affect how the address controller block generates the read addresses according to the required standard. The first is the number of NTT levels $\log n$ whether it is even or odd. The second factor is whether the NTT fully splits into degree 0 polynomials or not as in Kyber. Since the design consists of two parallel butterfly units, four polynomial coefficients are processed per clock cycle. For any standard, the read addresses of every four coefficients in each two successive levels are the same, only in different order. The first and fourth coefficients remain unchanged and only the second and third ones that switch places between even and odd stages. Figure 7 illustrates the read address call sequences of the twiddle factors for the NTT/INTT operation and the polynomial coefficients for all operations applicable to Dilithium as an example of a standard that has a fully splitting polynomial ring and the number of its NTT levels is even. Two brackets per line represent address(es) of

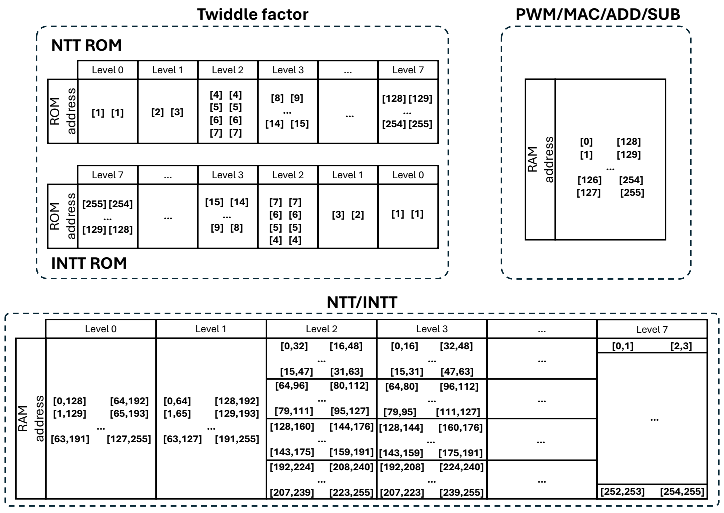


Fig. 7: Address generation of twiddle factors and polynomial coefficients for standards supporting fully-splitting NTT with even number of levels.

polynomial coefficients (or twiddle factors) required by the two BFUs. Regarding NTT/INTT, four addresses of different coefficients from the same polynomial and two twiddle factor addresses are generated. As for the remaining operations, a single address is produced per BFU since they operate on the same coefficient in two different polynomials placed in different memories.

If the number of NTT levels is odd, read address generation for all successive pairs of the NTT stages remain the same as explained before. However, the coding of addresses of the last stage which is odd is different. In a fully-splitting NTT, the last stage must process consecutively ordered coefficients. Finally for Kyber where the NTT does not fully split, the modification is quite trivial where the level count stops at $\log n-1$, but the read address generation remains unchanged.

Since the polynomial coefficients are placed in a data RAM designed as a four-bank memory, the read addresses generated so far need to be mapped to real RAM addresses as presented in [23, 42] using an integrated address resolver block within the address controller module. The bank address for selecting the memory banks and the new addresses of the coefficients are computed as follows:

$$BankAddr = \sum_{i=0}^{\lceil \frac{1}{2}\log_2(n)\rceil -1} RawAddr[2i+1:2i] \bmod 4$$

$$NewAddr = RawAdrr \;>>\; 2,$$

where i is the bit position. This guarantees a conflict-free memory access.

To generate the write addresses, one straightforward method is to use shift registers to propagate the read addresses for the necessary number of clock cycles according to the pipeline depth of each operation. However instead of delaying the read addresses, we opted to store them in a small ROM (distributed memory) of length that equals the maximum pipeline depth (which is that of the NTT/INTT operation). The address of this ROM is a simple 3-bit counter. The polynomial coefficient read addresses are written inside the ROM upon a read enable signal. Similarly, the write addresses are read from the ROM upon a write enable signal that is activated after the pipeline is fulfilled for each operation. Figure 8 shows the timing diagram of the write operation taking the pipeline depth of the NTT/INTT as an example.

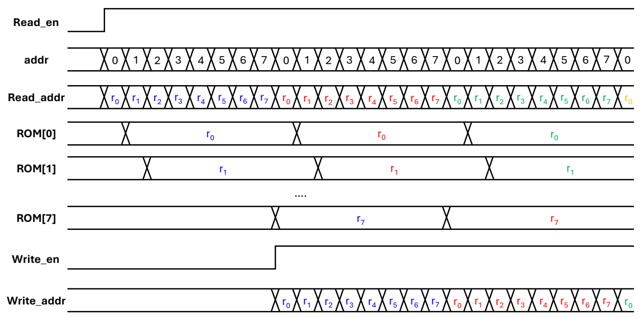


Fig. 8: Write address generation timing diagram for an NTT operation.

4 Implementation results

Our proposed architecture was implemented on an Xilinx XC7A100T Artix-7 FPGA using Vivado 2022.1. For each standard, post-place and route (PnR) simulations using the Vivado simulator were performed.Table 2 details the resource utilization of our butterfly and modular reduction (MR) units and compares them to the state-of-the-art. Our Dilithium BFU is similar to the first control element (CE0) of [34]. The nubmer of required LUTs is slightly higher as for the number of FFs, we report 48% more because we deploy registers at the input and output of the BFU whereas in [34] they do not show these registers in their FF count, but from their NTT latency it is evident that they also implement these registers, but most probably at a higher level. Compared to [33]'s lookup-table-based modular reduction, our Kyber MR unit needs 64% more LUTs and only 4 more FFs. In [11] the number of LUTs and FFs of the BFU are almost the same for both Dilithium and Kyber. This is because the authors unified the butterfly unit to support multiple lattice-based PQC schemes at run time. Our Dilithium and Kyber BFUs (that are tailored at design time) require 43% and 65% less LUTs and similar percentages less FFs, respectively. Table 3 shows the implementation results of the polynomial arithmetic unit and comparison with state-of-the-art.

For Dilithium, our proposed architecture fairly occupies overall less resources compared to other existing implementations. It requires 2604 LUTs, 770 FFs and 4 DSPs. Since our design targets low-cost area-constraint applications, we opted to optimize each sub-block individually while promoting resource sharing wherever possible. Using the same FPGA, Nguyen et al. [32] designed a configurable high-speed NTT accelerator suitable for both Dilithium and Kyber and supports both radix-2 and radix-4 MDC NTT operation. Compared to their radix-2 implementation, our design occupies 2.85× less LUTs and 6.85× less FFs, at the expense of an extra 4 DSPs. Land et al. [23] proposed a mid-range implementation that focused on optimizing the usage of LUTs and FFs by exploiting DSPs available in low-end FPGAs. Their design consists of three modules (NTT based on two radix-2 BFUs, Multiply-Accumulate and Matrix-Vector Mult.) that collectively perform the poly arithmetic functions. Our design needs 2× less LUTs, 1.6× less FFs and far less DSPs. A single BRAM is needed to store the twid-

Table 2: Detailed Resource utilization comparison with state-of-the-art implementations of the butterfly and the modular reduction units.

Work	Module	LUT	FF	DSP	Work	Module	LUT	FF	DSP
Dilithium					Kyber				
[34]	BFU	351	209	2	[33]	BFU			
	⌊ MR	69	71	0		⌊ MR	50	34	0
[11]	BFU	705	488	8	[11]	BFU	703	474	8
	⌊ MR					⌊ MR			
⋆	**BFU**	**400**	**310**	**2**	⋆	**BFU**	**241**	**167**	**1**
	⌊ MR	**114**	**72**	**0**		**⌊ MR**	**82**	**39**	**0**

Table 3: Resource utilization comparison with state-of-the-art implementations of the poly arithmetic unit.

Work	Plat.	n	$\lceil log_2(q) \rceil$	Resources				Freq.	Latency (CCs)		
				LUT	FF	DSP	BRAM	[MHz]	NTT	INTT	PWM
Dilithium											
[32]	A7	256	23	7451	5275	0	0	180	319	319	
[23]	A7	256	23	5676	1218	41	1	311	533	536	
[5]	VUS+	256	23	4509	3146	8	0				
[43]	Z7000	256	23	2812	1748	10	2		296	296	
[17]	ZUS+	256	23	2759	2037	4	7		606	614	147
[34]	ZUS+	256	23	2637	1071	8	1	385	268	268	
[38]	Z7000	256	23	2386	932	8	1	217	264		
[11]	A7	256	23	2119	1058	8	3	117	1052	1318	3688
⋆	**A7**	**256**	**23**	**2604**	**770**	**4**	**0**	**100**	**519**	**519**	**134**
Hawk											
⋆	**A7**	**512**	**31**	**3801**	**1135**	**8**	**0**	**83**	**1159**	**1159**	**262**
		512	**31**	**3968**	**1135**	**8**	**0**	**83**	**1159**	**1159**	**262**
		1024	**31**	**4287**	**1139**	**8**	**0**	**83**	**2567**	**2567**	**518**
		1024	**31**	**4451**	**1139**	**8**	**0**	**83**	**2567**	**2567**	**518**
Raccoon											
⋆	**A7**	**512**	**24**	**3194**	**912**	**4**	**0**	**83**	**1159**	**1159**	**262**
		512	**25**	**3458**	**998**	**4**	**0**	**83**	**1159**	**1159**	**262**
Kyber											
[32]	A7	256	12	4834	4683	0	1	250	247	247	
[11]	V7	256	12	2128	1144	8	3	174	922	1184	3812
[39]	A7	256	12	1579	1058	2	3		512	448	256
[31]	A7	256	12	1416	1074	2	1.5	227	448	448	256
[33]	A7	256	12	1154	1031	2	0	300	456	456	265
[40]	A7	256	12	948	352	1	2.5	190	904	904	3359
[41]	A7	256	12	609	640	2	4	257	490	490	
⋆	**A7**	**256**	**12**	**1583**	**458**	**2**	**0**	**100**	**455**	**455**	**134**
Polka											
⋆	**A7**	**1024**	**16**	**2512**	**593**	**2**	**0**	**100**	**2567**	**2567**	**518**

dle factors whereas we decided to use a distributed ROM instead. In [5] the authors targeted a high-performance implementation. They use a radix-4 2×2 NTT BFU arrangement to speed up the NTT and INTT operations. As a result, their design requires 1.7×, 4× and 2× more LUTs, FFs and DSPs than our design, respectively. Zhao et al. [43] proposed a compact and high-speed hardware design that employs four BFUs in a radix-2 R2MDC NTT architecture. Our design utilizes slightly less LUTs, 2.3× less FFs and 2.5× less DSPs. The two BRAMs are reportedly used to store the twiddle factors and to replace large shift registers (required by the R2MDC NTT). The work by Gupta et al. [17] reports a lightweight hardware implementation that invested in resource and control logic sharing as well as pre-computed LUTs among other optimization strategies. Their design requires two radix-2 BFUs and two 64×256 dual-port RAMs to transfer internal computations of NTT in a ping-pong fashion until all the layers have been processed. Our design requires slightly less LUTs, 2.6× less FFs and the same number of DSPs. As for the BRAM cost, their design uses 7 BRAMs, however, it is not clear how they are exploited. The polynomial arithmetic module introduced in [34] targets a lightweight hardware implementation even though it implements a radix-4 NTT. Our design employs almost the same number of LUTs and 1.4× less FFs, but half the DSPs. Wang et al. [38] targets a high-performance efficient design that uses a radix-4 NTT block. Among the reported state-of-the-art, Wang's implementation is the smallest even though it requires 4 BFUs. Yet, our design still uses less FFs (18%), slightly higher LUTs (9%) and half the DSPs. An interesting implementation by [11] offers both run-time and compile-time configurability to cover a wide base of parameter sets (n and q) and performance requirements of various platforms. Compared to their one BFU implementation, our design uses 23% more LUTs, but it requires 27% less FFs and half the DSPs. Our design can run at a maximum frequency of 100 MHz on Artix-7 FPGA. Indeed, this is less than the state-of-the-art reported frequencies. Nevertheless, they mostly used high-speed FPGAs and targeted efficient high-performance applications. In addition, our target is area-constrained applications, therefore the maximum frequency requirement can be relaxed. We also provide latency figures for all polynomial operations which admittedly lie mid-range the state-of-the-art spectrum.

For Kyber, our proposed architecture requires 1583 LUTs, 458 FFs and 2 DSPs. Compared to the configurable design of Nguyen et al. [32], our implementation needs 1.9× less LUTs, 7.9× less FFs at the expense of an extra 2 DSPs. Compared to the one BFU implementation of [11], our design uses 1.3× less LUTs, 1.88× less FFs and 4× less DSPs. Xing et al. [39] implemented a compact hardware design. Our design needs slightly more LUTs, but 2.3× less FFs. Works such as [31,33] opted to rearrange the order of the polynomial coefficients at all NTT/INTT stages instead of changing the data addresses to access the coefficients stored in the RAMs in a conflict-free memory access fashion. This requires the use of a reordering unit that also acts a temporary memory to hold coefficients after each stage. Their design eliminates the need for additional memory usage in the iterative NTT design claiming to simplify the control logic.

Our design again requires slightly more LUTs, but 2.3× less FFs. Yaman et al. [40] proposed three different hardware architectures (lightweight, balanced, high-performance), we compare our work to their lightweight implementation that employs one BFU. Indeed their design occupies 1.67× less LUTs, 23% less FFs and half the DSPs compared to ours. Zhang et al. [41] reported an efficient implementation that favors a ping-pong memory access to read/write the polynomial coefficients from/to the block RAMs. Their approach avoids read/write conflicts without the cost of reordering the coefficients. Indeed, our design occupies 2.3× more LUTs, but 1.4× less FFs which makes the overall resource utilization in favor of [41]. Also, the maximum frequency of most reported works is higher than ours which is 100 MHz even though all implementations are done on the same Artix-7 FPGA. As for the latency of the NTT and INTT operations, they are quite comparable to similar designs that use two butterflies. Regarding the point-wise multiplication we report nearly half the clock cycles compared to similar works. This is because the polynomial arithmetic unit does not compute the last step of polynomial multiplication needed after NTT. It is left to be implemented on the upper level. We also provide the resource utilization for Hawk, Raccoon and Polka, but comparison to the literature was not possible since there is no available hardware implementations that we know of.

A cautionary note, since the block RAMs used to store the coefficients are shared among other higher level modules, they are not considered part of the BRAM cost within the poly arithmetic unit in our design and in most state-of-the-art works.

5 Conclusion

This paper presents a lightweight polynomial arithmetic hardware architecture for post-quantum digital signatures and KEMs, suitable for low-cost and area-constrained applications. Our approach uses an in-place NTT without reordering of coefficients during the NTT and INTT operations, which reduces the complexity and the area cost of the control unit. In addition, storing the NTT twiddle factors in LUTs avoids occupying unnecessary BRAM footprint and reusing them during INTT avoids redundancy. The proposed architecture also establishes a straightforward address generation mechanism with simple conflict-free memory access which further facilitates the usage of lower resources. For most algorithms, our work utilizes fewer hardware resources than state-of-the-art lightweight implementations.

Acknowledgment

François-Xavier Standaert is a senior research associate of the Belgian Fund for Scientific Research (F.R.S.-FNRS). This work has been funded in part by the European Research Council (ERC) Advanced Grant BRIDGE (number 101096871). Views and opinions expressed are those of the authors and do not necessarily reflect those of the European Union or the ERC. Neither the European Union nor the granting authority can be held responsible for them.

References

1. CERG, dilithium. https://github.com/GMUCERG/Dilithium
2. Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS,-kyber algorithm specifications and supporting documentation (version 3.02). https://pq-crystals.org/kyber/data/kyber-specification-round3-20210804.pdf
3. Azouaoui, M., Bronchain, O., Cassiers, G., Hoffmann, C., Kuzovkova, Y., Renes, J., Schneider, T., Schönauer, M., Standaert, F., van Vredendaal, C.: Protecting dilithium against leakage revisited sensitivity analysis and improved implementations. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2023**(4), 58–79 (2023). https://doi.org/10.46586/TCHES.V2023.I4.58-79
4. Bai, S., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS,-dilithium algorithm specifications and supporting documentation (version 3.1). https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf
5. Beckwith, L., Nguyen, D.T., Gaj, K.: High-performance hardware implementation of crystals-dilithium. In: 2021 International Conference on Field-Programmable Technology (ICFPT). pp. 1–10 (2021). https://doi.org/10.1109/ICFPT52863.2021.9609917
6. Bernstein, D.J., Persichetti, E.: Towards KEM unification. Cryptology ePrint Archive, Paper 2018/526 (2018), https://eprint.iacr.org/2018/526
7. Bisheh-Niasar, M., Azarderakhsh, R., Kermani, M.M.: High-speed ntt-based polynomial multiplication accelerator for crystals-kyber post-quantum cryptography. IACR Cryptol. ePrint Arch. p. 563 (2021), https://eprint.iacr.org/2021/563
8. Bos, J.W., Bronchain, O., Ducas, L., Fehr, S., Huang, Y.H., Pornin, T., Postlethwaite, E.W., Prest, T., Pulles, L.N., van Woerden, W.: HAWK version 1.0.2 (september 26, 2024). https://hawk-sign.info/hawk-spec.pdf
9. Chen, L., Jordan, S.P., Liu, Y.K., Moody, D., Peralta, R.C., Perlner, R.A., Smith-Tone, D.C.: Report on post quantum cryptography. https://nvlpubs.nist.gov/nistpubs/ir/2016/NIST.IR.8105.pdf (Apr 2016), nathional Institute of Standards and Technology, Tech. Rep. NIST IR 8105
10. Chen, Z., Karabulut, E., Aysu, A., Ma, Y., Jing, J.: An efficient non-profiled side-channel attack on the crystals-dilithium post-quantum signature. In: 39th IEEE International Conference on Computer Design, ICCD 2021, Storrs, CT, USA, October 24-27, 2021. pp. 583–590. IEEE (2021). https://doi.org/10.1109/ICCD53106.2021.00094
11. Derya, K., Mert, A.C., Öztürk, E., Savaş, E.: CoHA-NTT: A configurable hardware accelerator for NTT-based polynomial multiplication. Cryptology ePrint Archive, Paper 2021/1527 (2021), https://eprint.iacr.org/2021/1527
12. Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: Crystals-dilithium: A lattice-based digital signature scheme. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(1), 238–268 (2018). https://doi.org/10.13154/TCHES.V2018.I1.238-268
13. Ducas, L., Postlethwaite, E.W., Pulles, L.N., van Woerden, W.P.J.: Hawk: Module LIP makes lattice signatures fast, compact and simple. In: Agrawal, S., Lin,

D. (eds.) Advances in Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part IV. Lecture Notes in Computer Science, vol. 13794, pp. 65–94. Springer (2022). https://doi.org/10.1007/978-3-031-22972-5_3

14. Duval, S., Méaux, P., Momin, C., Standaert, F.: Exploring crypto-physical dark matter and learning with physical rounding towards secure and efficient fresh rekeying. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(1), 373–401 (2021). https://doi.org/10.46586/TCHES.V2021.I1.373-401
15. Eum, S., Lee, M., Seo, H.: Optimizing hawk signature scheme performance on armv8. Applied Sciences **14**(19) (2024). https://doi.org/10.3390/app14198647
16. Guerreau, M., Martinelli, A., Ricosset, T., Rossi, M.: The hidden parallelepiped is back again: Power analysis attacks on falcon. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2022**(3), 141–164 (2022). https://doi.org/10.46586/TCHES.V2022.I3.141-164
17. Gupta, N., Jati, A., Chattopadhyay, A., Jha, G.: Lightweight hardware accelerator for post-quantum digital signature crystals-dilithium. IEEE Trans. Circuits Syst. I Regul. Pap. **70**(8), 3234–3243 (2023). https://doi.org/10.1109/TCSI.2023.3274599
18. Hoffmann, C., Libert, B., Momin, C., Peters, T., Standaert, F.: POLKA: towards leakage-resistant post-quantum cca-secure public key encryption. In: Boldyreva, A., Kolesnikov, V. (eds.) Public-Key Cryptography - PKC 2023 - 26th IACR International Conference on Practice and Theory of Public-Key Cryptography, Atlanta, GA, USA, May 7-10, 2023, Proceedings, Part I. Lecture Notes in Computer Science, vol. 13940, pp. 114–144. Springer (2023). https://doi.org/10.1007/978-3-031-31368-4_5
19. Kamel, D., Standaert, F., Bronchain, O.: Information theoretic evaluation of raccoon's side-channel leakage. IACR Commun. Cryptol. **1**(3), 44 (2024)
20. Kannwischer, M.J., Genêt, A., Butin, D., Krämer, J., Buchmann, J.: Differential power analysis of XMSS and SPHINCS. In: Fan, J., Gierlichs, B. (eds.) Constructive Side-Channel Analysis and Secure Design - 9th International Workshop, COSADE 2018, Singapore, April 23-24, 2018, Proceedings. Lecture Notes in Computer Science, vol. 10815, pp. 168–188. Springer (2018). https://doi.org/10.1007/978-3-319-89641-0_10
21. Karabulut, E., Aysu, A.: FALCON down: Breaking FALCON post-quantum signature scheme through side-channel attacks. In: 58th ACM/IEEE Design Automation Conference, DAC 2021, San Francisco, CA, USA, December 5-9, 2021. pp. 691–696. IEEE (2021). https://doi.org/10.1109/DAC18074.2021.9586131
22. Kim, S., Lee, K., Cho, W., Cheon, J.H., Rutenbar, R.A.: Fpga-based accelerators of fully pipelined modular multipliers for homomorphic encryption. In: Andrews, D., Cumplido, R., Feregrino, C., Platzner, M. (eds.) 2019 International Conference on ReConFigurable Computing and FPGAs, ReConFig 2019, Cancun, Mexico, December 9-11, 2019. pp. 1–8. IEEE (2019). https://doi.org/10.1109/RECONFIG48160.2019.8994793
23. Land, G., Sasdrich, P., Güneysu, T.: A hard crystal - implementing dilithium on reconfigurable hardware. In: Grosso, V., Pöppelmann, T. (eds.) Smart Card Research and Advanced Applications - 20th International Conference, CARDIS 2021, Lübeck, Germany, November 11-12, 2021, Revised Selected Papers. Lecture Notes in Computer Science, vol. 13173, pp. 210–230. Springer (2021). https://doi.org/10.1007/978-3-030-97348-3_12

24. Liu, Y.K., Moody, D.: Post-quantum cryptography, and the quantum future of cybersecurity. https://doi.org/10.1103/PhysRevApplied.21.040501 (Apr 2024), nathional Institute of Standards and Technology,Physical Review Applied
25. Longa, P., Naehrig, M.: Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In: Foresti, S., Persiano, G. (eds.) Cryptology and Network Security - 15th International Conference, CANS 2016, Milan, Italy, November 14-16, 2016, Proceedings. Lecture Notes in Computer Science, vol. 10052, pp. 124–139 (2016). https://doi.org/10.1007/978-3-319-48965-0_8
26. Lyubashevsky, V.: Fiat-shamir with aborts: Applications to lattice and factoring-based signatures. In: Matsui, M. (ed.) Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5912, pp. 598–616. Springer (2009). https://doi.org/10.1007/978-3-642-10366-7_35
27. Lyubashevsky, V.: Lattice signatures without trapdoors. In: Pointcheval, D., Johansson, T. (eds.) Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7237, pp. 738–755. Springer (2012). https://doi.org/10.1007/978-3-642-29011-4_43
28. Marzougui, S., Ulitzsch, V., Tibouchi, M., Seifert, J.: Profiling side-channel attacks on dilithium: A small bit-fiddling leak breaks it all. IACR Cryptol. ePrint Arch. p. 106 (2022), https://eprint.iacr.org/2022/106
29. National Institute of Standards and Technology: Module-lattice-based digital signature standard. Tech. Rep. Federal Information Processing Standards Publications (FIPS PUBS) 204, U.S. Department of Commerce, Washington, D.C. (2024). https://doi.org/10.6028/NIST.FIPS.204
30. National Institute of Standards and Technology: Module-lattice-based key-encapsulation mechanism standard. Tech. Rep. Federal Information Processing Standards Publications (FIPS PUBS) 203, U.S. Department of Commerce, Washington, D.C. (2024). https://doi.org/10.6028/NIST.FIPS.203
31. Nguyen, T.H., Dam, D.T., Duong, P.P., Kieu-Do-Nguyen, B., Pham, C.K., Hoang, T.T.: Efficient hardware implementation of the lightweight crystals-kyber. IEEE Transactions on Circuits and Systems I: Regular Papers pp. 1–13 (2024). https://doi.org/10.1109/TCSI.2024.3443238
32. Nguyen, T., Kieu-Do-Nguyen, B., Pham, C., Hoang, T.: High-speed NTT accelerator for crystal-kyber and crystal-dilithium. IEEE Access **12**, 34918–34930 (2024). https://doi.org/10.1109/ACCESS.2024.3371581
33. Ni, Z., Khalid, A., Liu, W., O'Neill, M.: Towards a lightweight crystals-kyber in fpgas: an ultra-lightweight bram-free NTT core. In: IEEE International Symposium on Circuits and Systems, ISCAS 2023, Monterey, CA, USA, May 21-25, 2023. pp. 1–5. IEEE (2023). https://doi.org/10.1109/ISCAS46773.2023.10181340
34. Pham, T.X., Duong-Ngoc, P., Lee, H.: An efficient unified polynomial arithmetic unit for crystals-dilithium. IEEE Transactions on Circuits and Systems I: Regular Papers **70**(12), 4854–4864 (2023). https://doi.org/10.1109/TCSI.2023.3316393
35. del Pino, R., Prest, T., Rossi, M., Saarinen, M.O.: High-order masking of lattice signatures in quasilinear time. In: 44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023. pp. 1168–1185. IEEE (2023). https://doi.org/10.1109/SP46215.2023.10179342

36. Pöppelmann, T., Oder, T., Güneysu, T.: High-performance ideal lattice-based cryptography on 8-bit atxmega microcontrollers. In: Lauter, K.E., Rodríguez-Henríquez, F. (eds.) Progress in Cryptology - LATINCRYPT 2015 - 4th International Conference on Cryptology and Information Security in Latin America, Guadalajara, Mexico, August 23-26, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9230, pp. 346–365. Springer (2015). https://doi.org/10.1007/978-3-319-22174-8_19
37. Shor, P.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th Annual Symposium on Foundations of Computer Science. pp. 124–134 (1994). https://doi.org/10.1109/SFCS.1994.365700
38. Wang, T., Zhang, C., Cao, P., Gu, D.: Efficient implementation of dilithium signature scheme on fpga soc platform. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **30**(9), 1158–1171 (2022). https://doi.org/10.1109/TVLSI.2022.3179459
39. Xing, Y., Li, S.: A compact hardware implementation of cca-secure key exchange mechanism CRYSTALS-KYBER on FPGA. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(2), 328–356 (2021). https://doi.org/10.46586/TCHES.V2021.I2.328-356
40. Yaman, F., Mert, A.C., Öztürk, E., Savas, E.: A hardware accelerator for polynomial multiplication operation of CRYSTALS-KYBER PQC scheme. In: Design, Automation & Test in Europe Conference & Exhibition, DATE 2021, Grenoble, France, February 1-5, 2021. pp. 1020–1025. IEEE (2021). https://doi.org/10.23919/DATE51398.2021.9474139
41. Zhang, C., Liu, D., Liu, X., Zou, X., Niu, G., Liu, B., Jiang, Q.: Towards efficient hardware implementation of ntt for kyber on fpgas. In: 2021 IEEE International Symposium on Circuits and Systems (ISCAS). pp. 1–5 (2021). https://doi.org/10.1109/ISCAS51556.2021.9401170
42. Zhang, N., Yang, B., Chen, C., Yin, S., Wei, S., Liu, L.: Highly efficient architecture of newhope-nist on FPGA using low-complexity NTT/INTT. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(2), 49–72 (2020). https://doi.org/10.13154/TCHES.V2020.I2.49-72
43. Zhao, C., Zhang, N., Wang, H., Yang, B., Zhu, W., Li, Z., Zhu, M., Yin, S., Wei, S., Liu, L.: A compact and high-performance hardware architecture for crystals-dilithium. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2022**(1), 270–295 (2022). https://doi.org/10.46586/TCHES.V2022.I1.270-295
44. Zhou, S., Xue, H., Zhang, D., Wang, K., Lu, X., Li, B., He, J.: Preprocess-then-ntt technique and its applications to kyber and newhope. In: Guo, F., Huang, X., Yung, M. (eds.) Information Security and Cryptology - 14th International Conference, Inscrypt 2018, Fuzhou, China, December 14-17, 2018, Revised Selected Papers. Lecture Notes in Computer Science, vol. 11449, pp. 117–137. Springer (2018). https://doi.org/10.1007/978-3-030-14234-6_7