Efficient Pipelining Exploration for a High-performance CRYSTALS-Kyber Accelerator

Ziying Ni[1,2], Ayesha Khalid[1], Dur-e-Shahwar Kundi[1], Máire O'Neill[1] and Weiqiang Liu[2]

[1] The Centre for Secure Information Technologies (CSIT), Queens University Belfast, UK
{zni03,a.khalid,d.kundi}@qub.ac.uk,m.oneill@ecit.qub.ac.uk

[2] College of Electronic and Information Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing, China liuweiqiang@nuaa.edu.cn

Abstract. This work explores several architectural optimizations to report a fast and area-time (AT) product efficient hardware accelerator for a lattice based Key Encapsulation Mechanism (KEM) scheme called the CRYSTALS-KYBER. Kyber was recently chosen as the first quantum resistant KEM scheme for standardisation, after three rounds of the National Institute of Standards and Technology (NIST) initiated NIST PQC competition for the search of the best quantum resistant KEMs and digital signatures (started in 2016). Kyber is based on Module-Learning with Errors (M-LWE) class of Lattice-based Cryptography, that is known to manifest efficiently on FPGAs. The architectural optimizations include inter-module and intra-module pipelining, that is designed and balanced via FIFO based buffering to ensure maximum parallelisation. The implementation results show that compared to the state-of-the-art, the proposed architecture delivers 23.8-43.8% speedups at three different security levels on Artix-7 and Zynq UltraScale+ devices, 50-75% reduction in DSPs and no BRAM resources at comparable security level. Consequently, the AT product efficiency is reported to be 45.8-51.9% higher in comparison with the state-of-the-art designs.

Keywords: Post-quantum Cryptography (PQC), Lattice-based Cryptography (LBC), Module-Learning with Errors (M-LWE), CRYSTALS-KYBER, Hardware accelerator.

1 Introduction

The advent of quantum computers threatens the security of all existing cryptosystems. A Quantum algorithm, called Shor's algorithm [Sho99], is capable of completely breaking all currently deployed Public-key Cryptography (PKC), including the RSA [Riv78] and Elliptic Curve Cryptography (ECC) [Mil85]. In addition, Grover's search quantum algorithm [Gro96] reduces the complexity of the search space of a brute force attack on symmetric-key encryption schemes (e.g., AES [RD01]) and hashing (e.g., SHA-3 [Dwo15]) to half. The National Institute of Standards and Technology (NIST) announced a formal global call in 2016, to standardized a new Post-quantum Cryptography (PQC) based Public-key Encryption (PKE) and digital signature schemes [Moo16]. In 2017, 69 proposals were selected in the Round 1 of the NIST PQC, whereas four candidates and five alternatives were shortlisted in July 2020 Round 3, for the final selection of PQC based PKC candidates. Three out of these four candidate algorithms were lattice-based cryptography (LBC) schemes, namely CRYSTALS-KYBER [ABD+20], SABER [ZZY+21], and NTRU [HRSS17]. NIST intended to standardize no more than one of these lattice-based Public-key Encryption and Key-establishment algorithm schemes and on July 5, 2022, NIST announced the first group of winners from its six-year competition. CRYSTALS-Kyber was announced as the first PQC algorithm to be standardized as a Key-encapsulation Mechanism (KEM) [GDD+22].

CRYSTALS-KYBER (hereafter called Kyber) [ABD+20] KEM is based on the module learning with errors (M-LWE) problem, which is a lattice based hard problem. M-LWE is an "algebraic" LWE with a tight formal mathematical security reduction of the ring-LWE (R-LWE) problem [PP19]. Schemes based on the M-LWE problem have a more elaborate algebraic structure and consequently, higher security than R-LWE schemes while achieving higher performance than LWE schemes. In the M-LWE scheme, the parameter k is introduced to restrict the dimensions of the public-key matrix $\boldsymbol{A}$, but all elements of the matrix are on the ring $\mathbb{Z}_q[x]/(x^n+1)$. Unlike SABER [ZZY+21], the polynomial operations in Kyber can be computed using Number Theoretic Transform (NTT), which allows Kyber to gain higher throughput performance. For the round 2 NIST PQC submission, the Kyber team adopted a technique to reduce the parameter q of Kyber from $7,681$ to $3,329$, further reducing the complexity of modular reduction and area resources.

This work presents a high-speed and Area-Time (AT) product efficient hardware accelerator for the IND-CCA secure Kyber KEM scheme. The accelerator comprises of Key Generation, Encryption and Decryption modules for the three NIST specific security levels 1/3/5. The major contributions of this work are summarized as follows:

- Our Kyber accelerator *aggressively exploits architectural parallelisation* via optimal inter-module and intra module pipelines. For balancing the pipeline, buffering is provided at the interface of several modules. The computation order of the modules is re-arranged to facilitate a fuller pipeline.

- A fully pipilined *Radix-2-Multipath Delay Commutator (MDC)-NTT* core is presented that multiplexes the resources for both NTT and inverse-NTT (INTT) computations. By using different delay units, the bit reversal operation in the NTT/INTT calculation is completely eliminated. Due to pipelining, a single NTT/INTT computation requires only 128 clock cycles, once the pipeline is full.

- *Resource utilization is reduced* in terms of DSPs and BRAMs by several strategies. The hardware for NTT/INTT is shared. For balancing pipelining, buffering is done via FIFOs (restricting first in first out data access) instead of BRAMs that allow more flexible access but are more expensive in resource consumption. The data in bit reversal order is fed directly into the INTT, and the INTT gets the result in normal order. Thus the use of BRAM is eliminated from our architecture.

- Our proposed Kyber accelerator easily surpasses all comparable previously reported FPGA implementations of Kyber implementations for comparable security levels in terms of throughput performance and design efficiency (i.e., Area-Time (AT) product). Compared to the state-of-the-art design, the proposed architecture has a speedup of 1.238-1.438× at the three security levels for Artix-7 devices and 1.254-1.516× for Zynq-UltraSale+, respectively. In terms of hardware efficiency, the proposed architecture improved the AT efficiency by 51.9%, 50.0%, and 45.8% at three different security levels.

This paper is organized as follows. Section II introduces the Kyber protocol and NTT, and Section III presents the proposed overall architecture including various modules. A fast pipelined scheduling scheme and storage methods for the Kyber hardware architecture is presented in Section IV. Implementation results and comparison are provided in Section V, and Section VI concludes our work.

2 Preliminaries

In this section, Round 3 of the Kyber KEM is explained, describing the main NTT construct in Kyber.

2.1 Kyber.v3 (NIST PQC Round 3)

Kyber KEM is the first lattice based PQC algorithm chosen by NIST for standardisation. The relative balance between performance and security can be directly adjusted by tweaking the size of the matrix k; the choice of k varies to 2, 3, or 4 for security levels 1 (Kyber512), 3 (Kyber768) and 5 (Kyber1024), respectively. The noise parameter η is adjusted according to the security level. The IND-CCA secure Kyber KEM, submitted to NIST PQC Round 3 (so called Kyber.CCA here) consists of three main steps: key generation (Kyber.CCA.KeyGen), key encapsulation (Kyber.CCA.Enc), and key decapsulation (Kyber.CCA.Dec). The prime of Kyber p is changed from 7681 to 3329, which enables that the polynomial multiplication in Kyber can be accelerated using NTT. Kyber.CCA implementation is built on top of the Kyber.CPA, using the Fujisaki-Okamoto transform [FO99]. Kyber.CPA comprises of three components: key generation, encryption, and decryption. In each function, uniform and binomial distributed samplers generate polynomial data, NTT and Inverse NTT (INTT) are used to compute polynomial multiplications. The NTT/INTT results are compressed and transmitted. For more details of Kyber, please refer to [ABD+20].

2.2 NTT in Kyber

The NTT algorithm is derived from the Fast Fourier Transform (FFT) algorithm. Compared to standard schoolbook polynomial multiplication, the complexity of the NTT algorithm is reduced from $O(n^2)$ to $O(nlogn)$. The choice of modulus in the construction of Kyber satisfies the modulus restriction for the NTT, and polynomial multiplication calculations in Kyber can be accelerated using NTT.

For NTT transformations, polynomials are expressed in terms of a vector of coefficients, e.g., the polynomial $a(x) = a_0 + a_1x + ... + a_{n-1}x^{n-1} + a_nx^n$ is represented as a set of n points $a(x_i) = \{(x_0, y_0), (x_1, y_1), ..., \}$. The NTT computation can be substantially improved when using n special points, i.e. n powers of the rotation factor w.

Defining polynomial with n elements, k is an integer ranging from 0 to $n-1$, w is the square of ψ, where ψ is the primitive root of unity in NTT. The NTT and INTT transformation are shown as follows:

$$\hat{a}_m = \sum_{k=0}^{n-1} a_k \psi^{(2m+1)k} = \sum_{k=0}^{n-1} (a_k \psi^k) w^{mk} mod\, q \tag{1}$$

$$a_k = \frac{1}{n} \sum_{m=0}^{n-1} \hat{a}_m \psi^{-(2m+1)k} = \psi^{-k} \cdot \frac{1}{n} \sum_{m=0}^{n-1} \hat{a}_m w^{-mk} mod\, q \tag{2}$$

There are some differences between the NTT defined in Kyber [ABD+20]. In Kyber, the base field $\mathbb{Z}_q$ contains the primitive 256^{th} root of unity but not the primitive 512^{th} root of unity. Thus, the polynomial $x^{256} + 1$ can be defined as a polynomial of 128 degrees of 2. Let $\zeta = 17$ be the first primitive 256^{th} root of unity modulo q. The polynomial $x^{256} + 1$ can be written as:

$$f(x) = \sum_{i=0}^{255} f_i x^i \tag{3}$$

Therefore, any polynomial $f(x)$ can be divided into two polynomials according to the parity term after NTT calculation, as shown in the formula Eq.(4)-(5), where $\mathbf{br}_7$ means bit reversal of the unsigned 7-bit integer i. In addition, when performing NTT calculations, these two parity polynomials are calculated independently.

$$\hat{f}_{2i} = \sum_{j=0}^{127} f_{2i}\zeta^{(2\mathbf{br}_7 i+1)j} \tag{4}$$

$$\hat{f}_{2i+1} = \sum_{j=0}^{127} f_{2i+1}\zeta^{(2\mathbf{br}_7 i+1)j} \tag{5}$$

The polynomial multiplication $\mathrm{NTT}(f) \circ \mathrm{NTT}(g) = \hat{h}\ mod\ x^2 - \zeta^{2\mathbf{br}_7 i+1}$, where $\hat{f}$, $\hat{g}$, and $\hat{h}$ are the polynomial in NTT representation, can be expressed as:

$$\begin{aligned}\hat{h}_{2i} + \hat{h}_{2i+1}x = \hat{f}_{2i}\hat{g}_{2i} + \zeta^{2\mathbf{br}_7 i+1}\hat{f}_{2i+1}\hat{g}_{2i+1} \\ + x(\hat{f}_{2i}\hat{f}_{2i+1} + \hat{g}_{2i}\hat{g}_{2i+1})\end{aligned} \tag{6}$$

2.3 Related Work

Hardware based Kyber KEM accelerator designs for Kyber have primarily and rightly focused on the optimizations of its most computationally intensive constituent component, i.e., polynomial multiplication module (implemented via the NTT module) [GL21, ZLL+21, YM21, TCW+21, BNAMK21a]. Several recent work on Kyber focus on the improvement of the NTT memory access methods and modulo multiplication units. In 2020, Chen *et al.* proposed a pipelined processor for the vector of polynomials using two-column sequential storage and bit-inverted addressing access for the Round 1 prime of Kyber ($p = 7,681$) [CMC+20]. Zhang *et al.* in 2021 proposed an effective NTT structure for the prime in Round 2 ($p = 3,329$) [ZLL+21]. In the same year, Mojtaba *et al.* proposed to apply the prime $p = 3,329$ reduction module of the K2-RED algorithm without register delays to four parallel-computing NTT butterfly units and achieve a high-speed polynomial multiplication accelerator [BNAMK21a]. Ferhat *et al.* implemented different architectures of multiplication units by increasing the number of butterfly units in the NTT based on a unified butterfly structure [YM21], with lightweight, balanced, and high-performance hardware architectures, using 1, 4, and 16 parallel butterfly units, respectively.

Several hardware-only complete Kyber implementations are reported in literature [HHLW20, XL21, BNAMK21a, BNAMK21b, DMG21]. The first complete pure hardware implementation of Kyber was presented by Huang *et al.* in 2020; the module reuse technique was undertaken as an optimization to achieve 129× speedup compared to Cortex-M4 processor, at the cost of high resource consumption [HHLW20]. In 2021, Xing *et al.* proposed a low-cost, high-performance Kyber processor on the Artix-7 platform [XL21]. This architecture utilized a predefined control order table with short control codes for the scheduling of various processes of NTT and uses different sized first-in-first-out (FIFO) for data transmission/ reception to achieve good throughput performance with limited computational resources. In the same year, Bisheh-Niasar *et al.* proposed a polynomial multiplier for Kyber, using a 2×2 reconfigurable butterfly cell with pure combinational logic (so called K2-RED modulo subtraction) cells [BNAMK21a], requiring only 798 LUTs and 715 FFs on the Artix-7 device operating at 200MHz. The same reconfigurable butterfly module was used in an instruction set processor for Kyber [BNAMK21b], whose overall operating frequency was limited. In addition, Dang *et al.* implemented three different lattice-based PQCs of NIST Round 3 in hardware [DMG21] and a 52.5%, 65.7%, 76.2% improvement in speed of Kyber at three different security levels compared to earlier high performance Kyber implementations [XL21].

3 The Proposed Kyber Hardware Accelerator

The proposed Kyber accelerator comprises of the server and client side implementations. The server-side accelerator includes the key generation and decapsulation functions while the client-side implementation is a sub-set of components of server, performing only key encapsulation function. An overall design of server-side Kyber accelerator is shown in Fig. 1. It comprises of a controller, computational units and storage units. The storage unit consists of multiple FIFOs and ROMs (in the NTT/INTT and polynomial multiplication unit). In Fig. 1, the black, green, and purple colors indicate the data flow for Kyber.CPA.KeyGen, Kyber.CPA.Dec and Kyber.CPA.Enc functions, respectively. The dashed lines indicate the data flow through INTT, which is computed after the solid line of the same color. To enable a pipelined execution, all modules use independent resources except NTT/INTT modules that have shared resources. In our design, four parallel data streams (totalling 48-bits for 12-bit streams) are simultaneously processed. Therefore, two Point-wise Multiplication (PWM) units and four groups of parallel adders (ADDs_1) are used after NTT computation. A finite state machine based controller controls the execution of the hash module, till enough random samples are generated. Then the Kyber accelerator enters an pipelined state for computational units until the hash function computation is needed again. The controller assembles the padding of hash module, based on the hash function needed for the current state and feeds it into the hash module. In addition, the controller includes four 256-bit registers for storing and distributing the results generated by the hash module. There are several differences in the accelerator's execution for the three different security levels of Kyber. For Kyber512, the centered-3 binomial distribution (CBD_3) sampling module is added to the overall architecture. For Kyber1024, the compress/ decompress modules differ from the other two security levels. Various modules are shared between key generation and decapsulation function on the server-side for area minimization.

In the rest of this section, we present the main modules used in the Kyber accelerator, i.e., the hash module, the sampling module, the NTT/INTT module, the PWM module, and the compression and encoding modules. All descriptions are based on the Kyber768 implementation (e.g., FIFO sizes, $k = 3$ etc.), while differences for Kyber512 and Kyber1024 are mentioned.

3.1 SHA-3 Based Hash Module

The hash module generates the random distribution samples to the sampling modules, to provide the coefficients of the noise polynomial and consequently can become the potential computational bottle-neck of the design. Hence consideration is given to match the Hash module throughput with other modules. Our design uses one Keccak core that is serially for different SHA-3 functions. The Keccak core needs significant hardware resources, e.g., 54.2% of the total LUTs used in the Kyber design [XL21].

Kyber uses four different SHA-3 functions, i.e., SHA3-256, SHA3-512, SHAKE128, and SHAKE256. While the Keccak core computations remain identical for these functions, the padding method differs for all of them. The maximum size of data output in a single computation is also different, i.e., 1,344, 1,088 and 576 bits for SHA3-128, SHA3-256, and SHA3-512, respectively.

The hash module receives up to 1,344-bits of data, in 64-bit chunks (in 21 cycles). This data is fed to the Keccak core, after the controller has added the SHA-3 function appropriate padding, e.g., SHA3-512 function takes 64-bit data inputs in 9 cycles and in the subsequent 12 cycles 64-bit '0' are padded to get a 1,344-bits of data. The Hash module defaults to the squeeze stage from the second round of computation and automatically feeds the results of the first round into the Kaccak core for further 24-round of computations; However the Hash module also contains an absorb signal for cases when the amount of

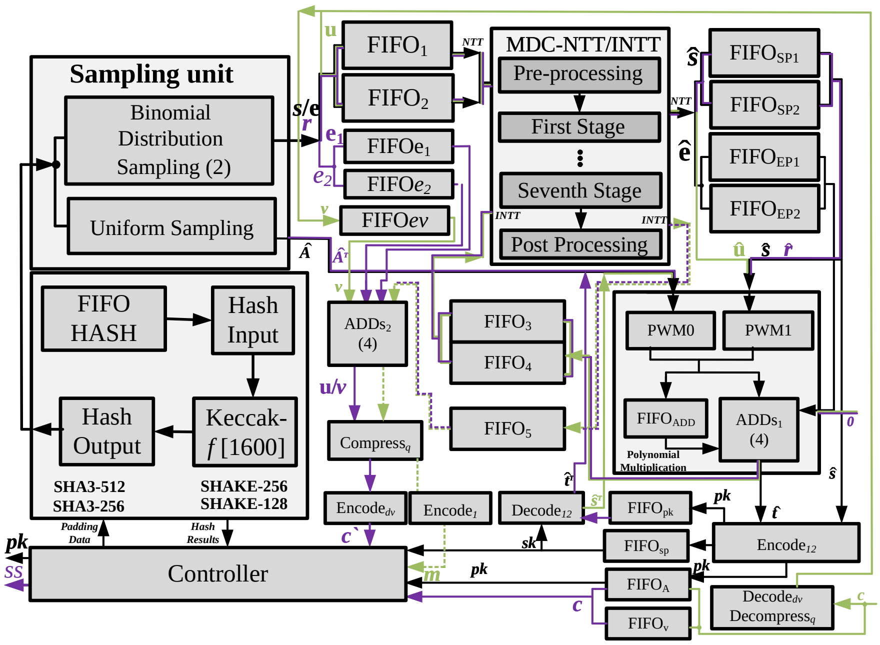

Figure 1: The overall server-side architecture diagram of Kyber768. The black, green, and purple colored arrows indicate the data flow for Kyber.CPA.KeyGen, Kyber.CPA.Dec and Kyber.CPA.Enc functions, respectively. The dashed line in the figure is the data flow after the solid line of the same color is complete.

data is greater than the maximum amount of data that can be carried in a single round.

The architecture of the Hash module comprises of the input/output stages and a Keccak core, as shown in Fig. 2. These three stages are independently buffered and operate in a fully pipelined manner ensuring substantial acceleration. It has a 64×64 FIFO to cache-in large volume of input data. The FIFO output is fed as 64-bit words into the 1,344-bit shift register in a big-endian format (in 21 cycles). The Keccak core takes the data and applies 24 rounds of iterative operations on it. The internal state of the Keccak core is of 1,600-bits, out of which the 1,344 most significant bits (MSBs) are taken for data output or squeezing after 24 round calculations. Out of the output stage of the Hash module, the data is filled in the 256-bit registers in the controller as 64-bits per cycle. Although 36-bit or 48-bit inputs are generally used in the sampling module in the proposed architecture, going from high-bit-width data to low-bit-width data, i.e., 48/32-bit will not cause discontinuities for input to the sampling module. For absorbing stage, in the proposed architecture, the 1344 MSBs of output from the Hash module are selected to be XOR with the subsequent input.

3.2 Sampling of Noise

Kyber uses two types of sampling, namely uniform distribution sampling ($Parse$) and central binomial distribution sampling ($CBD_\eta, \eta = 2, 3$). All three security levels of Kyber need $Parse$ to get polyvector matrix $\hat{\boldsymbol{A}}$ (as well as $\hat{\boldsymbol{A}}^{\boldsymbol{T}}$). For Kyber512, two different CBD modules (CBD_2, CBD_3) are used, the other two security levels use only CBD_2. Fig. 3(a)-(b) represent the $Parse$ and CBD_η modules, respectively, where Fig. 3(c) shows the state of the FIFO when a 64-bit data and 48-bit data conversion is performed.

While the hash module takes 21 clock cycles to output 1344-bits data (64-bits per cycle) and a single computation of Keccak core takes 24 cycles, there is always meant to

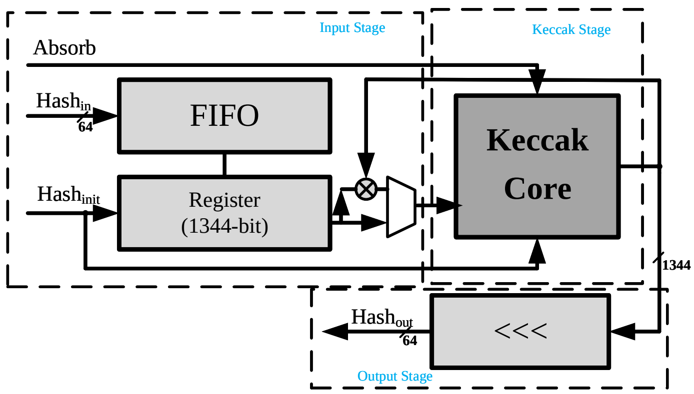

Figure 2: Hash module comprising of input/ output stages and a Keccak core

be a waiting interval even if pipelining is employed. The sampled data is stored in 64 × 32 FIFO called FIFO_{GETA}, converted to 48 bits, and fed into the sampling module. For the 64To48 module, as shown in Fig.3(c), the 64-bit data is fed three times every four cycles. The data is internally registered to enable a consecutive stream of 48-bits outputs in 4 consecutive cycles. The results of $Parse$ do not need to be stored and can be multiplied directly with the NTT results at output. However, the output of matrix $\hat{\boldsymbol{A}}$ requires 48 × 256 FIFO, called $\text{FIFO}_{AMatrix}$, to filter out the $Parse$ data. Results of $Parse$ greater than 3329 are discarded. For each set of matrix $\hat{\boldsymbol{A}}$ data, it is packaged into 64 48-bit data outputs (64 × 4) for subsequent polynomial multiplication.

The CBD_2 and CBD_3 modules require 32 and 48 bits of data, respectively, to generate eight output results. Data output from the Hash module is stored in a 64 × 32 FIFO called FIFO_{HASHO}. The CBD_3 requires a 64To48 module before sampling while CBD_2 does not need this conversion. The FIFO_{HASHO} pops out 64-bits of data once in alternate clock cycles. The upper and lower halves of this data (32 bits each) are processed in two consecutive cycles.

3.3 NTT/INTT Module

The polynomial multiplication is both a resource hungry and a computational bottle neck in a lattice based cryptography design. The Kyber parameters were tweaked to be more 'NTT-friendly' in 3rd round submission of NIST PQC and the use of NTT/INTT for polynomial multiplication is part of Kyber specifications [ABD+20]. For polynomial multiplication via NTT, first the NTT of the multiplicand and multiplier polynomials is computed, a multiplication of the two vectors is carried out and then the inverse NTT (INTT) is computed to complete the polynomial multiplication. Since NTT/INTT calculations are not simultaneously performed, the same architecture is reused to minimize resource consumption. To balance area and speed and match the throughput of data in the subsequent modules, the Radix-2 multipath delay commutator (MDC)-based architecture turns out to be the optimal choice.

A Switch-MDC-NTT (S-NTT) architecture is shown in Fig. 4. The S-NTT consists of a pre-processing unit, seven general processing units, and one post-processing unit. All of the first six general processing units contain a radix-2 butterfly unit (BF2), a modulo multiplier, delay unit (D), and a two-channel commutator (C2), while the 7^{th} general processing unit only contains a BF2 unit and a delay unit. The MDC architecture employs pipelining, significantly reducing the computation time by enabling simultaneous execution of multiple consecutive NTT/INTT calculations. The BF2 unit performs addition and

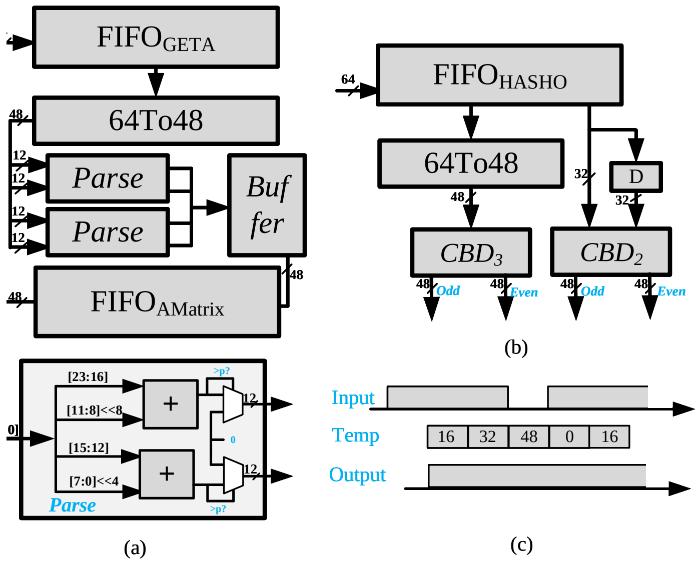

Figure 3: Sampling modules: (a) *Parse* module, (b) CBD_η module, and (c) The output stage of FIFO$_{GETA}$ and FIFO$_{HASHO}$.

subtraction calculations for each cross of input data pair. The result of the subtraction in BF2 is fed into C2 after the modulo reduction while the addition result does not need to be reduced and hence fed into C2 after buffering to match the pipeline. The internal architecture of the modulo multipliers follow Algorithm 4 in [XL21], which uses an optimized Barrett reduction algorithm with multiple partitioning and addition operations. The modulo multiplier is pipelined and requires six cycles for completion, the first two perform the 12-bit multiplication and the last four perform the reduction operation.

The S-NTT contains both pre-processing and post-processing blocks, and both use two modulo multiplication units for computation; However these two units do not execute simultaneously. To reduce resource consumption, the modulo multiplication unit is reused in these blocks. The pre-processing is performed at the start of the NTT, before entering the first stage of NTT. For INTT, the data enters the first stage directly and the post-processing multiplication is performed on the output of the seventh stage of the calculation. Note that reusing the multiplier in pre-processing and post-processing causes the NTT and INTT not to be pipelined when switching, but the delay in this case is negligible considering the difficulty of merging the pipeline between the NTT and INTT itself in the encapsulation and decapsulation functions in Kyber.

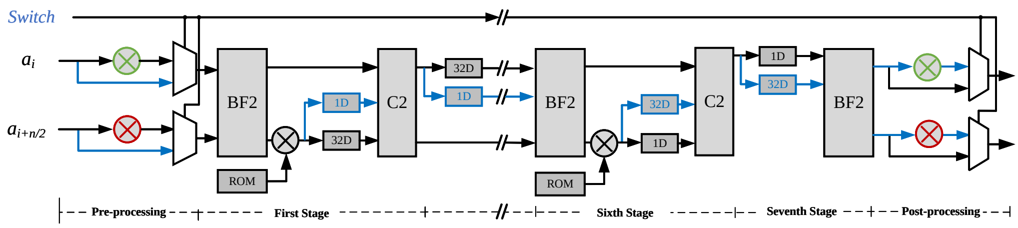

Figure 4: The switch Radix-2 multipath delay commutator (MDC) NTT/INTT pipelined architecture with seven stages.

For NTT computation, the Cooley-Tukey butterfly structure is undertaken at all stages. While the data flow of NTT/INTT is the same in the proposed architecture. In order to ensure that the output order of INTT is the normal order, the INTT in S-NTT is input in a bit-reversed order. Thus the result obtained from the polynomial multiplier can be directly input into the INTT in order. The calculation units in S-NTT, such as BF2 and modular multiplier, are all reused, but the delay unit (D) in each stage cannot be reused due to the different input order. The blue line and block diagram in Fig. 4 represent the alternative delay unit when calculating INTT. The result of INTT is sequential data separated by parity and even, no BRAM unit is used to store data in the input and output of S-NTT. The start-up time for the very first NTT and INTT calculation (when the pipeline is not full) is 119 cycles. The cycle time for a single 256-point-wise calculation after full pipelining is 128. Hence the first NTT/INTT computation requires 119+128 cycles but the subsequent NTT/INTT computations require only 128 cycles.

3.4 Point-wise Multiplications (PWMs) and ADDs

The point-wise multiplication (PWM) in Kyber is not as straightforward as required in Ring-LWE. It requires different computations for even and odd values of the point-wise multiplication product. For Kyber, assume that polynomials $\hat{f}$ and $\hat{g}$ are multiplied, and the result is $\hat{h}$, root of unity ζ, then an optimized PWM calculation formula based on the karatsuba algorithm from Eq.(6) was proposed in 2020 [XL21] as follows:

$$\hat{h}_{2i} = \hat{f}_{2i}\hat{g}_{2i} + \hat{f}_{2i+1}\hat{g}_{2i+1} \cdot \zeta^{2br(i)+1} \tag{7}$$

$$\hat{h}_{2i+1} = (\hat{f}_{2i} + \hat{f}_{2i+1})(\hat{g}_{2i} + \hat{g}_{2i+1}) - (\hat{f}_{2i}\hat{g}_{2i} + \hat{f}_{2i+1}\hat{g}_{2i+1}) \tag{8}$$

Thus, the result of h_{2i} comes from the sum of $\hat{f}_{2i}\hat{g}_{2i}$ and $\hat{f}_{2i+1}\hat{g}_{2i+1} \cdot \zeta^{2br(i)+1}$ as well as $\hat{f}_{2i+1}\hat{g}_{2i+1} \cdot \zeta^{2br(i)+1}$ comes from multiplying three 12-bit data ($\zeta^{2br(i)+1}$ can be stored in ROM by precomputation). If $\hat{f}_{2i+1}\hat{g}_{2i+1}$ do not perform reduction in time, it will cause the bit width of the addition to increase. Also, note that the results of $\hat{f}_{2i}\hat{g}_{2i}$ and $\hat{f}_{2i+1}\hat{g}_{2i+1}$ are directly usable in the calculation of $\hat{h}_{2i+1}$, only one additional multiplication is needed when calculating $\hat{h}_{2i+1}$. Since the pipeline length has minimal impact on the overall design calculation time, the proposed PWM design matches the frequency of the other modules. As shown in Fig. 5, the PWM module requires eleven cycles to provide a full pipeline operation. In the first two cycles two 12-bit multiplications and one 13×12 bit multiplication is carried out. sum_1 and sum_2 in Fig. 5 are derived from $\hat{f}_{2i} + \hat{f}_{2i+1}$ and $\hat{g}_{2i} + \hat{g}_{2i+1}$, respectively; However sum_1 is reduced to modulo p to ensure that $sum_1 \times sum_2$ does not exceed 25 bits first. During the calculation of h_{2i+1} if the result becomes negative,

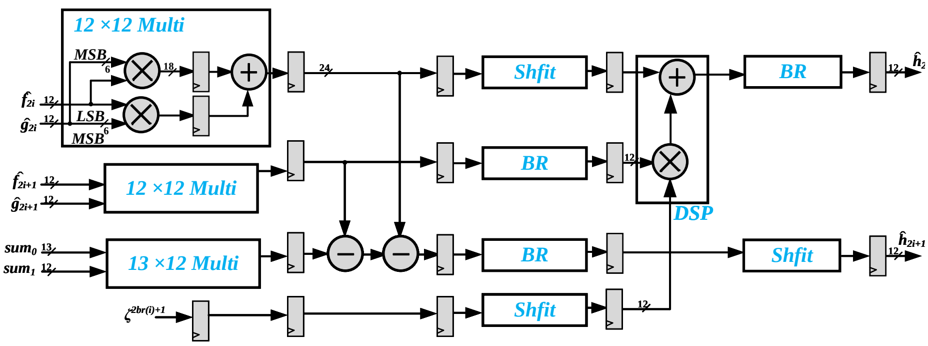

Figure 5: The piplined point-wise multiplication modules.

$24'hd01000$ is added to ensure that the data in the input of Barrett reduction (BR) is positive. In the 3^{rd} cycle, the result of $\hat{h}_{2i+1}$ is reduced to 12-bit and using the Barrett reduction (BR) module, which is the same as the BR used in the NTT module. Thereafter, $\hat{h}_{2i+1}$ passes through a shift register until $\hat{h}_{2i}$ is output at the same time. After $\hat{f}_{2i+1}\hat{g}_{2i+1}$ output from BR module, the 12-bit result is calculated with the value of ζ, added to $\hat{f}_{2i}\hat{g}_{2i}$ and the final result is reducing in one pass in the BR module.

Two PWM units are used in the overall design, allows simultaneous calculation of the four polynomial coefficients. A 24×64 distributed ROM is pre-computed to provide ζ values to two channels of PWMs according ($\zeta[23:12]$ and $\zeta[11:0]$). Four 12-bit adder with three input (ADDs_1) units are used in the design for the adder. This unit will performs the final addition for the 256 point-wise polynomial multiplication and the addition operation of the noisy polynomial multiplication e_2.

3.5 Storage: FIFOs and ROMs

The polynomials in lattice based cryptography schemes are large in dimension with low bit-width components, e.g., a single polynomial of Kyber has 256 elements, each of 12-bit size. High speed simultaneous access to several low bit-width data elements is critical to ensure high throughput performance but also dictates the amount of on-chip resources. The FPGA storage structures include Look-up-table RAM (LUTRAM) and Block RAMs (BRAMs). While BRAMs are dual ported, allowing fast and dual read/write access, they can be under-utilized due to their limited width-depth configurations, resulting in waste of resources. FIFOs instead are much more resource efficient in comparison and can be custom sized as needed. However, they only also access of data in the order that is pushed in the FIFO. All pre-computed data in the design uses distributed ROM, e.g., rotation factors in NTT/INTT and PWMs. Our Kyber accelerator has two types of FIFOs. One for buffering large amounts of data that cannot be computed on-the-fly. The depth in this type of FIFO does not need to match the total amount of data but just enough to meet the current pipeline speed. Another type of FIFO is used for the complete storage of data. Since the FIFO restricts first in first out access only, for NTT computations, the order of input data is adjusted before pushing into the FIFOs. In a fully pipelined architecture, the data is continuously pushed forward. Therefore, it can be consider that in many cases the computation modules are also treated as a kind of storage module, which saves a lot of storage units. Benefiting from the ability of INTT to directly use bit-reversal sequential inputs and to output data in normal order, the proposed architecture does not use BRAM to store data at all.

3.6 Decode Modules

In the kyber KEM, data in Kyber.CPA function is required to be compressed before encoding. Compression is relatively easy to implement, with the quotient being output by shifting the polynomial addition and reducing. Decompression does not require a reduction module, but all the data is multiplied by $\lfloor 3329/2 \rfloor$, using a constant multiplier composed of LUTs to reduce DSP consumption. The encode module takes data input from the INTT output. In a single inverse accessed address using BRAM, a pair of 12-bit data is available. Various encoding modules are obtained by shifting the input data and performing arithmetic operations. The bit width of the encoded output depends on the need of subsequent calculations. For example, the encoded output of pk is 64 bits one cycle, and the m output in Kyber.CPA.Dec generates 8 bits one cycle.

The Decode module is more complex, with different methods for the v vector and $\mathbf{u}$ polyvector for different security levels in Decode$_{dv}$. The same Decode method is used in Kyber512 and Kyber768, and it takes only eight cycles to complete a round of computation for v and $\mathbf{u}$ at 64 bits input. Fig. 6(a) shows the computation flow for the $\mathbf{u}$ decoding under

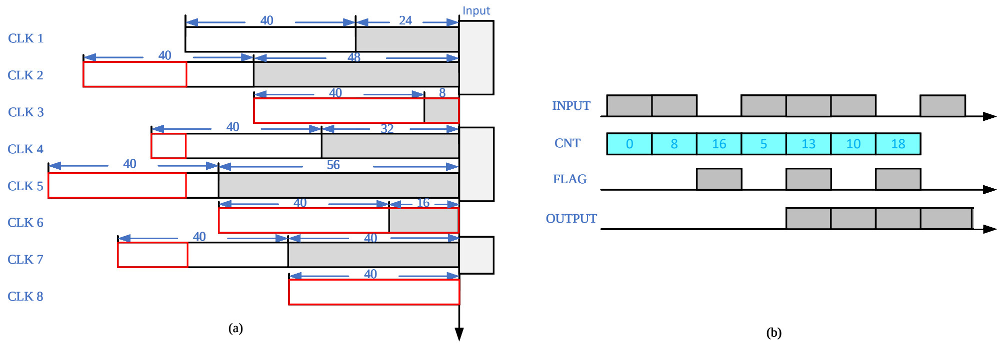

Figure 6: Decode$_{dv}$ in different security levels. (a) Decode$_{dv}$ of Kyber512 and Kyber768. (b) Decode$_{dv}$ of Kyber1024.

Kyber512 and Kyber768. For the **u** polyvector, a single cycle produces 48 bits of data after processing 40 bits (4 groups) of data, and the 8 data parity generated in two consecutive cycles are separated and fed into the FIFO in the same way as the sampled data. The left-side of Fig. 6(a) represents the data processed in the current cycle, where the white block is the 40-bit output data, the grey block is the data that cannot be output in the current cycle and needs to be processed in the next cycle, and the red block represents the data from the previous cycle. In addition, the grey block of the Input on the right side represents FIFO which storage ct should be fetched in the current cycle. Thus, five 64-bit data are fetched in eight consecutive cycles, generating 32 **u** polyvector data. Decoding v in Kyber512 and Kyber768 is simpler, with a single round of 4 sets requiring 16 bits of input and two times of input data taken from the FIFO every eight-cycle interval.

The decoding in Kyber1024 is more complicated than the other two security levels due to the irregularity of individual data (11 bits) in the decoding of the **u** polyvector. If the shifting method in Kyber768 is followed in Kyber1024, the generated single-round states reach more than 25. Therefore, an adaptive feedback (AF) scheme is proposed using the output control of the FIFO by the Decode module. In the AF scheme, the output of the FIFO is controlled by the decode module, whose internal counter is constantly updated with the current amount of unprocessed data to guide the output data and the FIFO read data. Each time the decode module receives 64 bits of data, the internal counter is added by 8. As shown in Fig. 6(b), in the case of the decode **u** vector, when the counter is greater than 11 (88 bits), the feedback FIFO pauses to take out the data and uses the 88 bits of data to generate eight **u** coefficients in the subsequent two cycles. For decoding the v vector coefficients, the feedback is performed and calculated when the counter is greater than 5.

4 Data Flow in the Proposed Kyber Accelerator

In order to ensure simultaneous execution of various modules of the Kyber accelerator, its modules are designed to support pipelining. The modules in kyber vary much in terms of their input/output word sizes and speeds and need to be thoughtfully scheduled so that data flow does not halt. This is ensured by using storage units whenever needed. For example, in CBD_2, a single 36 bit input results in eight 12-bit polynomial coefficients, while in the NTT module, only 24 bits of data are input and 24 bits are output in a single clock cycle. This section walks through the primary data flow for the main modules of Kyber.CPA.Enc (Kyber768) as an example, as shown in Fig. 7. The primary data

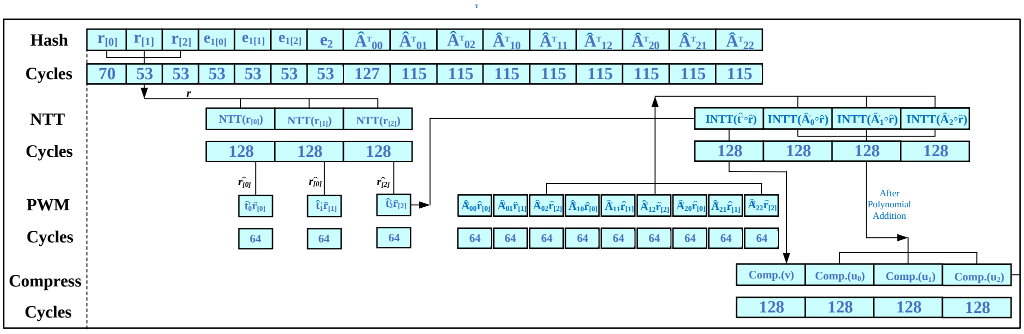

Figure 7: Execution order and clock cycles for main modules in Kyber.CPA.Enc of Kyber768.

flow consists of the Hash module, NTT/INTT module, PWM modules, and the data compression module. All cycles represent the total cycles of the current module calculation, except for the NTT/INTT, which represent the cycles needed to write the output data.

The Hash function computation time is critical to the data sampling time. For example, SHA3-256 takes 21 cycles to receive input data, 24 cycles for internal Keccak core computations, and up to 17 cycles (64-bit data each) can be output in one squeeze. Adding the time for data to pass through modules, a total of 70 cycles are needed for one hash calculation. However, the overhead of input and out can be made negligible by pipelining. In the Hash module, cycles for $r_{[0]}$ are computed from the start of data input, and the 70 cycles include the output stage of the previous parallel computation of the Hash module. The input data from $r_{[1]}$ is fed into the Hash module simultaneously with $r_{[0]}$ generating the output. Thus the computation cycles of the single Hash module is reduced to 53 cycles for CBD_2 sampling. To ensure that the matrix $\hat{\boldsymbol{A}}^{\boldsymbol{T}}$ gets the 256 data points from sampling module, the Hash module calculates SHAKE-128 by squeezing four times. Due to pipelining, the single calculation cycle is only 115 times.

The NTT/INTT computation is also pipelined, generating a complete a set of 256 coefficients every 128 cycles. During NTT computation, the first 64 cycles of data for each output set are delayed by 64 cycles of output using shift registers to meet $\hat{\boldsymbol{t}}^{\boldsymbol{T}}$ starting at the NTT output stage. The compress module takes the INTT output and generates a pair of data in a single pass, taking 512 cycles. The data flow of Fig. 7 consumes 1.9k cycles.

4.1 Primary Data Flow in Kyber768

The primary data flow starts with the sampling modules. A portion of the data generated by the Hash module is stored in four 256-bit registers for re-entry into the Hash module, while rest is sampled data is fed into the uniform *Parse* and binomial sampling $CBD_{2/3}$. A 64-bit data is fed into the *Parse* sampler every cycle, but the polyvector matrix $\hat{\boldsymbol{A}}^{\boldsymbol{T}}$ samples are not immediately generated. The 64-bit data is first fed to a 64 × 32 FIFO_{GETA} for caching before sampling. To match the 48-bit input per cycle of *Parse* sampling, the FIFO_{GETA} outputs 64 bits of data three times in four consecutive cycles. The 64To48 module reads 192 bits of data every four cycles and splits and reassembles to 48 bits per cycle output. Each sampling cycle of *Parse* produces four results, but not all are valid. This uncertainty in the generation of the matrix $\hat{\boldsymbol{A}}^{\boldsymbol{T}}$ can cause difficulties in matching data in subsequent polynomial multiplications. The 12 bits of valid results by *Parse* are stitched together as 48 bits and fed into a 48 × 256 $\text{FIFO}_{AMatrix}$. Due to the late timing of the output of $\hat{r}$ after the NTT calculation, the $\text{FIFO}_{AMatrix}$ waits for the complete output of $\hat{r}$ from the NTT module before generating the output data to ensure that the polynomial multiplication of $\hat{\boldsymbol{t^T}}$ does not conflict with matrix $\hat{\boldsymbol{A^T}}$ in the Kyber.CPA.Enc calculation.

Binomial distribution sampler comprises of CBD_2 and CBD_3 that take 32-bit and 48-bit inputs every cycle, respectively, to generate 8 results. The NTT module that takes CBD_2 output as odd/ even samples of 256 data points separately, and only a pair of data input is taken in a cycle. Hence the data generated by a single CBD_2 cycle (8 $\times$ 12 bits) is much more than what the NTT module can accommodate. Therefore, the data at the odd positions from $CBD_{2/3}$ is shifted back by 16 cycles, after which two FIFOs (FIFO_1 and FIFO_2, 48 $\times$ 128 FIFO) are utilized to store the sampling data.

As shown in Fig. 8, four consecutive even position data in order {0, 2, 4, 6} are combined and written to FIFO_1 in the first cycle. During the first 16 writing cycles, only FIFO_1 is used to store the even positioned data. Starting from the 17^{th} cycle of CBD_2 output, the FIFO_1 data is replaced with the data output from the shift register, i.e., the $CBD_{2/3}$ odd results {1, 3, 5, 7}, and the even position data from the $CBD_{2/3}$ {128, 130, 132, 134} is stored in FIFO_2, simultaneously. Therefore, eight data locations stored at the lowest part of these two FIFOs are {0, 128, 2, 130, 4, 132, 6, 134}. After that, an output signal (The gradient area in Fig. 8) is generated every four cycles to ensure that the FIFO data is read in order ({(0, 128), (2, 130), (4, 132), (6, 134)}). FIFO_1 and FIFO_2 continue to take in the results of the CBD while generating output data till the FIFO_1 and FIFO_2 signal to be empty. The data storage mechanism does not change for higher security level but the depth of FIFO_1 and FIFO_2 is increased to match the higher data size to be cached.

The polynomial multiplication module multiplies the data in different dimensions, i.e., matrix $\hat{\boldsymbol{A^T}}$ and vector $\hat{\boldsymbol{t^T}}$, by the data in different dimensions, i.e., vector $\hat{r}$, generated by NTT. The operation carried out is $\hat{A^T_{ji}} = \hat{A^T}[j][i] \cdot \hat{r}[i]$ or $\hat{t^T_i} = \hat{t^T}[i] \cdot \hat{r}[i]$ $(i, j \in 0, 1, 2$ in Kyber768). The polynomial $\hat{\boldsymbol{t^T}}$ is the output of key generation function and is stored in the 48 $\times$ 256 FIFO_{PK} and fed into the PWM modules simultaneously as the NTT produces its output. The $\hat{r}[i]$ generated by NTT is fed directly into the PWM module and also stored in two FIFOs (FIFO_{SP1} and FIFO_{SP2} as well as FIFO_{EP1} and FIFO_{EP2} for Kyber.CPA.KeyGen, 24 $\times$ 256 FIFO). The order of the NTT outputs in continuous data separated by parity {0, 2, 4, 6, ...1, 3, 5, 7, ...}. Fig. 9 shows the relationship between NTT results and FIFOs during the polynomial multiplication stage. A differential approach is used for FIFO_{SP1} and FIFO_{SP2} input to ensure that the order of the PWM module input is {0, 1, 2, 3...}. In the first 64 cycles of NTT generating results, the two 12-bit results of NTT will be spliced and input into FIFO_{SP1}. Starting from the 65^{th} cycle,

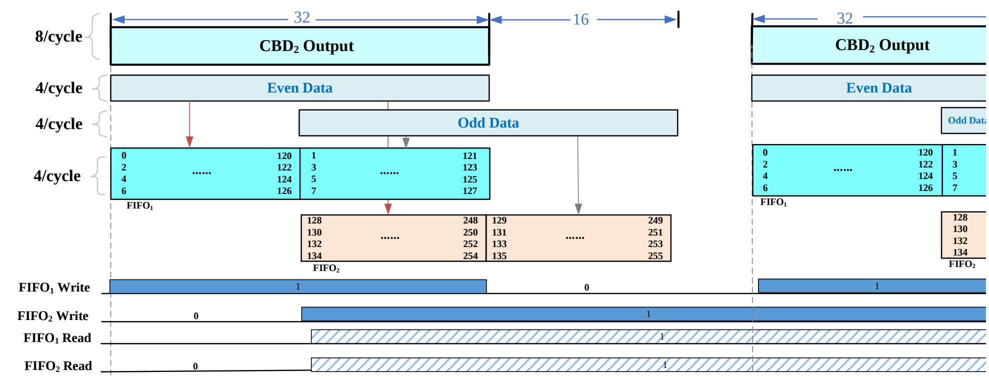

Figure 8: The access process of CBD_2 and FIFOs (FIFO_1 and FIFO_2). Eight data per cycle in CBD output and 4 data per cycle input in FIFO_1 and FIFO_2.

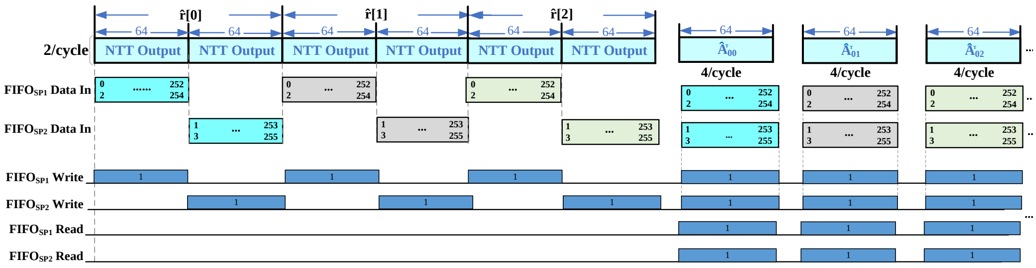

Figure 9: The access process of NTT results.

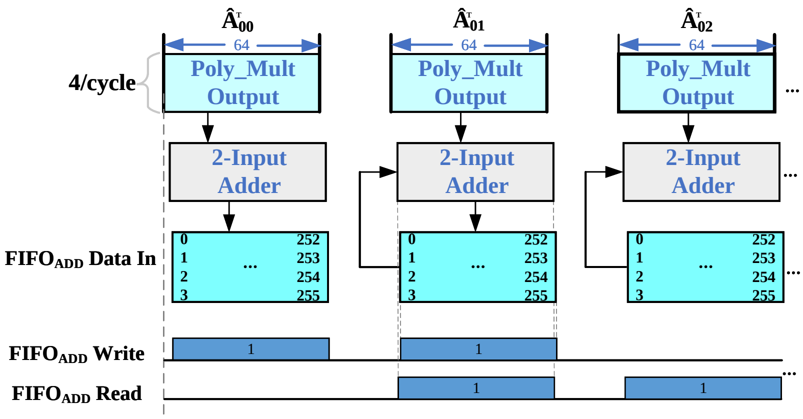

Figure 10: The access process of final ADDs (ADDs_1) in polynomial multiplication.

the results of NTT will be input into FIFO_{SP2}, thus ensuring that the order of lowest part data in these two FIFOs is {0, 1, 2, 3}. As the security level increases, more data volumes can be matched simply by increasing the depth of FIFOs. The data in FIFO_{SP1} and FIFO_{SP2} will be used multiple times. Therefore, the data output by current FIFOs will be stored in the same FIFOs again to reduce the use of storage resources. Since $\hat{r}[0]$, $\hat{r}[1]$, ..., $\hat{r}[n]$ are computed sequentially, the output data is restored at the top of the FIFOs and does not affect the polynomial multiplication computation of the current stage. In Kyber.CCA.KeyGen, all CBD sampling results are required for NTT calculation. Therefore, e from CBD is also stored in FIFO_1 and FIFO_2, waiting to be fed into the NTT module as s and r.

Final step of polynomial multiplication calculation should add up different dimensional data ($\{\hat{A}^T_{00}$,...,$\hat{A}^T_{22}\}$) to the same dimension, in the Kyber768 case, i.e. to compute $\hat{A}'^T[j] = \hat{A}^T_{j0} + \hat{A}^T_{j1} + \hat{A}^T_{j2}$ $(j \in 0, 1, 2)$ and in the Kyber.CCA.Enc and Kyber.CCA.Dec also should calculate $\hat{t'^T} = \hat{t^T_0} + \hat{t^T_1} + \hat{t^T_2}$. The output of PWM is spaced, and the number of cycles between data in $\hat{t^T_i}$ and $\hat{A}^T_{ji}$ is not the same, although it is possible to combine $\hat{A}^T_{j0}$, $\hat{A}^T_{j1}$, $\hat{A}^T_{j2}$ using shift registers to perform additions, the area will increases dramatically as the security level increases with the addition of shift registers. Therefore, using a FIFO to access sequential results of ADDs (ADDs_1) is still the best option. FIFO_{ADD} (48×64 FIFO) is set up to write and read data from the ADDs module. Fig. 10 shows the data flow of $\hat{A}^T_{00} + \hat{A}^T_{01} + \hat{A}^T_{02}$ computation and states of FIFO_{ADD} at Kyber768. There are three states in FIFO_{ADD}: store-only, store-read, and read-only. For Kyber512, only store-only and read-only states are included, while for Kyber768 and Kyber1024, three states are

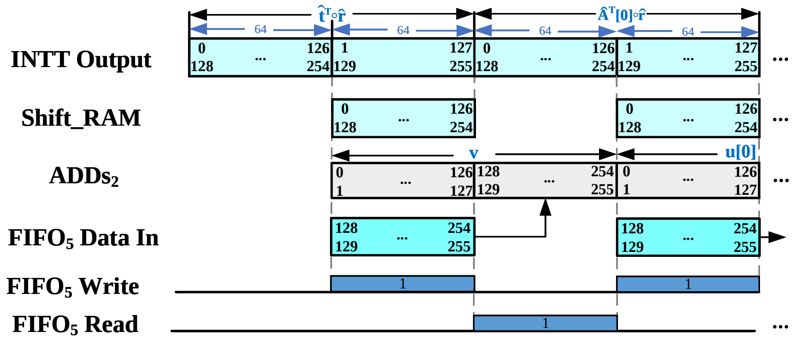

Figure 11: The access process of INTT results and $ADDs_2$.

all included, and two times the store-read state will be performed in Kyber1024. In the store-only state, the $FIFO_{ADD}$ is used to collect the result of the ADDs sequentially; in the store-read state, the data stored in the $FIFO_{ADD}$ is output sequentially first, and when the ADDs produces the result, the calculation result is stored in the $FIFO_{ADD}$ at the same time; in the read-only state, the $FIFO_{ADD}$ no longer accepts input, and all the data currently stored in the FIFO is output.

The $ADDs_1$ in the polynomial multiplication module generates four 12-bit data per cycle, which needs to be separated and buffered in odd and even positions before INTT calculation. Therefore, two FIFOs of the same size (24×256, $FIFO_3$ and $FIFO_4$) store the data in odd and even positions, respectively. The S-NTT module reads the data stored in $FIFO_3$ $FIFO_4$ directly in sequence and calculates them. As shown in Fig 11, the order of INTT results is close to the sequential order {(0, 128), (2, 130), ... (1, 129), ...}. Firstly using a shift register to delay INTT results in 64 cycles, then the output of the shift register is combined with the output of the current INTT results into four 12-bit data in the order {(0, 1, 128, 129), (2, 3, 130, 131), ...}. Two data with order {0, 1} of these four are fed directly into $ADDs_2$ to complete the polynomial adder calculation as well as the last two data with order {128, 129} are stored in a FIFO (24×64, $FIFO_5$). When S-NTT starts to output the second set of INTT results, the input of $ADDs_2$ is from $FIFO_5$, and the second set of INTT results will be shifted by 64 cycles.

4.2 Additional Data Flow in Key Decapsulation

Kyber.CCA.Dec contains both single Kyber.CPA.Enc and Kyber.CPA.Dec functions, while Kyber.CPA.Dec is more like a reduced Kyber main data flow. Fig. 12 shows the data flow in a single Kyber.CPA.Dec contains mainly NTT/INTT, polynomial multiplication, and encode modules. In Kyber.CPA.Dec, when the server-side receives the cipher-text ct from the client-side, it uses the decompress and decode module to process the data first, e.g., in Kyber768, it takes 64×4 cycles to decompress and decode all the data to polyvector $\mathbf{u}$ and vector v. The $\mathbf{u}$ and v output from decompress and decode module perform different operations when fed into the computation unit. Polyvector $\mathbf{u}$ will be fed into $FIFO_1$ and $FIFO_1$, and since every two cycles form 8 sets of data, the NTT starts the calculation after 32 cycles. v store in the FIFO (48×64, $FIFO_{ev}$) and then output when $ADDs_2$ is running. When the NTT calculation is complete, results from the S-NTT module are only multiplied with the secret key sk. Thus the NTT results do not need to be stored in FIFOs again. In this case, the NTT result is shifted back 64 bits using a shift register, enabling a succession of coefficients in the order {(0, 1, 2, 3),...} to be calculated simultaneously by the PWM unit for all four sets of data.

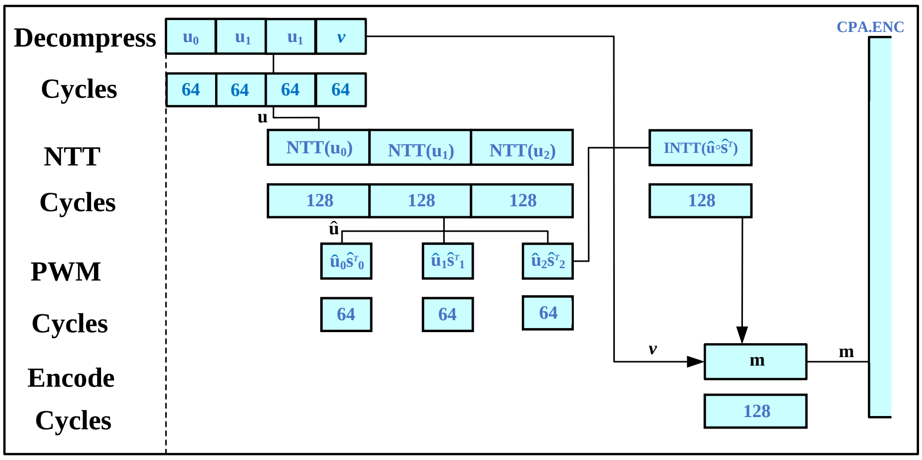

Figure 12: Cycles and order in Kyber.CPA.Dec (Kyber768).

In addition, when the service-side architecture accepts the ct value, the ct value is stored in FIFO_A (64×128 FIFO) and FIFO_v (48×256 FIFO) according to the difference between the **u** vector and v vector, respectively. When Kyber.CPA.Enc generates **u** vector and v vector, read FIFO_A and FIFO_v respectively to compare whether the decapsulation results are correct.

5 Implementation and Results

The proposed Kyber hardware architecture has been synthesized and implemented using Xilinx Vivado 2020.1 suite targeting two different devices, e.g., Artix XC7A200 and Zynq UltraScale+ XCZU7EV. Designs in this work have passed the post-place & route (post-PAR) simulation and functional verification. The main modules of the Kyber accelerator are the same under the three different security levels, except for the increase in FIFO depth due to the need of higher amounts of data.

Table 1 shows the speed and area of our Kyber accelerator architecture, compared against the state-of-the-art architectures, for the three different security levels it offers. Since the server side needs more computational processing, it consumes more resources than the client-side. Since all the constituent client side modules are included in the server side, we specify only the resources for the server side in Table 1, as done earlier in [XL21].

The DSPs are mainly used in the PWM module. In the S-INTT module, one pipeline cycle is added to each stage to perform the 12-bit multiplication using LUT. The BRAM block comes from the data storage prior to the INTT computation. Therefore, the proposed architecture uses only one DSP in each of the two PWM units, and only two DSPs are used for all security levels of the design. In addition, all storage units including distributed FIFO and distributed ROM using LUT resources as well as no BRAM resources are used in the proposed architecture. The accelerators at the three different security levels run at almost the same frequency as the critical paths are the same.

Comparison with related work focuses only on the hardware implementation of Kyber Round 3. *Key*, *Enc*, and *Dec* in Table 1 represent the Kyber.CCA.KeyGen, Kyber.CCA.Enc, and Kyber.CCA.Dec, respectively. Exploiting a fully pipelined implementation enables simultaneous execution of several modules, resulting in a 50.0% higher throughput performance for Kyber512 compared to [DMG21] (Artix-7). Compared

Table 1: Post-PAR Implementation Results for our Kyber Accelerator (Area and Timing) and Comparison with The-state-of-the-art Designs

Design	LUT	FF	DSP	BRAM	Slices	ENS*	F. [MHz]	Key/Enc./Dec./Total** [K Cycles]	Key/Enc./Dec./Total [us]	Improv. (Total Time)	AT*** (ENS×s)	FPGA Device
Kyber-512 NIST PQC Security level 1												
[BNAMK21b]	18,000	5,000	6	15	5,000	8,540	115	4.0/7.0/10.0	34.8/60.9/86.9/182.6	**87.6%**	1.56	Artix-7 XC7A100
[XL21]	6,785/7,412	3,981/4,644	3/3	2/2	2,126	2,818	161/167	3.8/5.1/6.7	23.4/30.5/41.3/95.2	**75.9%**	0.27	Artix-7 XC7A12
[BNAMK21a]	10,502	9,859	8	13	3,549	6,897	200	1.9/2.4/3.7	9.4/12.0/18.8/40.2	**43.8%**	0.28	Artix-7 XC7A100
[DMG21]	9,457	8,543	4	4.5	-	-	220	2.2/3.2/4.5	10.0/14.7/20.5/45.2	**50.0%**	-	Artix-7 XC7A200
Ours	**14,375/15,676**	**12,986/13,368**	**2/2**	**0/0**	**5,446**	**5,646**	**208**	**1.1/1.5/2.1**	**5.3/7.2/10.1/22.6**	-	**0.13**	**Artix-7 XC7A200**
[DMG21]	9504	8957	4	4.5	-	-	450	2.2/3.2/4.5	4.9/7.2/10.0/22.1	**51.6%**	-	Zynq-UltraScale+ XCZU7EV
Ours	**14,142/15,436**	**13,003/13,323**	**2/2**	**0/0**	-	-	**435**	**1.1/1.5/2.1**	**2.5/3.4/4.8/10.7**	-	-	**Zynq-UltraScale+ XCZU7EV**
Kyber-768 NIST PQC Security level 3												
[BNAMK21b]	16,000	6,000	9	16	4,000	8,036	115	7.0/10.0/14.0	60.9/86.9/121.7/269.5	**87.3%**	2.2	Artix-7 XC7A100
[XL21]	6,785/7,412	3,981/4,644	3/3	2/2	2,126	2,818	161/167	6.3/7.9/10.0	39.2/47.6/62.3/149.1	**77.1%**	0.42	Artix-7 XC7A12
[BNAMK21a]	11,783	10,424	12	14	3,952	7,896	200	2.7/3.2/4.8	13.3/16.3/24.0/53.6	**36.4%**	0.42	Artix-7 XC7A100
[DMG21]	10,530	9,837	6	6.5	-	-	220	2.6/3.7/4.9	12.0/17.0/22.2/51.2	**33.4%**	-	Artix-7 XC7A200
Ours	**15,636/16,926**	**12,976/13,526**	**2/2**	**0/0**	**5,864**	**6,064**	**208**	**1.7/2.4/3.0**	**8.2/11.5/14.4/34.1**	-	**0.21**	**Artix-7 XC7A200**
[DMG21]	10,458	10,458	6	6.5	-	-	450	2.6/3.7/4.9	5.9/8.3/10.9/25.1	**35.1%**	-	Zynq-UltraScale+ XCZU7EV
Ours	**15,455/17,280**	**12,927/13,476**	**2/2**	**0/0**	-	-	**435**	**1.7/2.4/3.0**	**3.9/5.5/6.9/16.3**	-	-	**Zynq-UltraScale+ XCZU7EV**
Kyber-1024 NIST PQC Security level 5												
[BNAMK21b]	16,000	6,000	12	17	5,000	9,532	112	10.0/14.0/18.0	86.9/121.7/156.5/365.1	**86.6%**	3.48	Artix-7 XC7A100
[XL21]	6,785/7,412	3,981/4,644	3/3	2/2	2,126	2,818	161/167	9.4/11.3/13.9	58.2/67.9/86.2/212.3	**76.9%**	0.59	Artix-7 XC7A12
[BNAMK21a]	13,347	11,639	16	16	4,585	9,321	185	3.5/4.1/6.2	17.3/20.6/31.3/69.2	**29.2%**	0.65	Artix-7 XC7A100
[DMG21]	11,623	11,131	8	8.5	-	-	220	3.6/4.8/5.8	16.2/21.7/26.4/64.3	**23.8%**	-	Artix-7 XC7A200
Ours	**16,088/17,975**	**12,954/13,748**	**2/2**	**0/0**	**6,263**	**6,463**	**208**	**2.7/3.4/4.1**	**13.0/16.3/19.7/49.0**	-	**0.32**	**Artix-7 XC7A200**
[DMG21]	11,676	11,959	8	8.5	-	-	450	3.6/4.8/5.8	7.9/10.6/12.9/31.4	**25.4%**	-	Zynq-UltraScale+ XCZU7EV
Ours	**15,965/18,405**	**12,902/13,760**	**2/2**	**0/0**	-	-	**435**	**2.7/3.4/4.1**	**6.2/7.8/9.4/23.4**	-	-	**Zynq-UltraScale+ XCZU7EV**

*ENS (equivalent number of slices) = $DSP \times 100 + BRAM \times 196 + Slices$[KZW+22] ** $Time(Total) = Time$(Key+Enc+Dec) *** Area and time production (AT) = ENS × Time ($Total$)

with [BNAMK21a], which also uses the high-speed NTT architecture, the speedup on Kyber512 is 47.0% for Key, 51.0% for Enc, and 50.7% for Dec (Artix-7). With faster devices (Zynq-UltraScale+) with larger resources, the speedup is 51.6% compared with [DMG21] in Kyber512. As the security level increases, the speedup is reduced as the proposed architecture uses the same computational architecture. Compared with the state-of-the-art architecture [DMG21], the total time of Key, Enc, and Dec are reduced by 33.4% and 23.8% under Kyber768 and Kyber1024, respectively (Artix-7). For Kyber1024, using the faster hardware architecture (Zynq-UltraScale+), it takes 6.2, 7.8, and 9.4 us for Key, Enc, and Dec, respectively. Compared with the designs in [XL21] and [BNAMK21b], the proposed architecture achieves a speedup of 4-7.5x for all three different security levels (Artix-7).

In terms of area, the proposed design is higher in LUT than the previous design due to the use of more FIFO cells. However, for other on-chip resources, the proposed design uses significantly less. Under Kyber512, compared with [BNAMK21b], [BNAMK21a], and [DMG21], the DSP is reduced by 4, 6, and 2 blocks, and the BRAM is reduced by 15, 13, and 4.5 blocks, respectively. For a fairer comparison of resource consumption, we estimate the equivalent number of slices (ENS) column, as undertaken in [KZW+22]. One DSP is taken as equivalent to 100 Slices, a 36K BRAM is equivalent to 196 Slices, resulting in ENS computation $ENS = DSP \times 100 + BRAM \times 196 + Slices$. Compared with [BNAMK21b] and [BNAMK21a], the proposed architecture reduces ENS by 24.5-33.8% and 18.1-30.7% for the three different security levels, respectively. Compared with [XL21], which implements a lightweight design, the proposed architecture uses more resources, but the speed increase outweighs the resource consumption. Compared with [DMG21], the proposed architecture uses more LUT resources but less DSP and BRAM resources. The number of DSPs is reduced by 50.0%, 66.7%, and 75.0% for the three different security levels, respectively. To better balance the advantages of area and speed, we introduce the AT (area and time product) metric, where AT = ENS × Time($Total$). As can be seen from Table 1, the proposed architecture reduces 51.9%, 50.0%, and 45.8% at three different security levels compared with [XL21] and [BNAMK21a], respectively. Therefore, the proposed Kyber accelerator significantly improved speed and hardware efficiency compared to the state-of-the-art at all three different security levels.

6 Conclusion

This work presents a high-performance Kyber accelerator that undertakes an optimally designed pipeline for parallel execution of various modules in the design. The accelerator uses a pipelined MDC-NTT to speed up operations and adds cycles to the pipeline to reduce DSP usage. Multiple FIFOs are used to buffer data for pipeline balancing. We performed a hardware implementation of the proposed architecture using two different devices, the Artix-7 and the Zynq-UltraScale+. The results show that the proposed Kyber accelerator on the Artix-7 is 1.438×, 1.334×, and 1.238× faster at security level 1/3/5, respectively. In terms of equivalent slice count, the proposed architecture reduces 45.8-51.9% in terms of AT (area and time product) for three different security levels. On Zynq-UltraScale+, the proposed architecture achieves a speedup of 1.516-1.254× compared to the state-of-the-art design. Thus, the proposed architecture achieves significant hardware speed and efficiency improvements.

7 Acknowledgments

This work is supported by grants from the National Natural Science Foundation of China (62022041), the Fundamental Research Funds for the Central Universities (NP2022103) and the Engineering and Physical Sciences Research Council (EPSRC) Quantum Communications Hub (EP/T001011/1).

References

[ABD+20] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Kyber algorithm specifications and supporting documentation. *NIST PQC Round 3*, 2020.

[BNAMK21a] Mojtaba Bisheh-Niasar, Reza Azarderakhsh, and Mehran Mozaffari-Kermani. High-speed NTT-based polynomial multiplication accelerator for post-quantum cryptography. In *2021 IEEE 28th Symposium on Computer Arithmetic (ARITH)*, pages 94–101, 2021.

[BNAMK21b] Mojtaba Bisheh-Niasar, Reza Azarderakhsh, and Mehran Mozaffari-Kermani. Instruction-set accelerated implementation of CRYSTALS-Kyber. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 68(11):4648–4659, 2021.

[CMC+20] Z. Chen, Y. Ma, T. Chen, J. Lin, and J. Jing. Towards Efficient Kyber on FPGAs: A Processor for Vector of Polynomials. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 247–252, 2020.

[DMG21] Viet Ba Dang, Kamyar Mohajerani, and Kris Gaj. High-speed hardware architectures and FPGA benchmarking of CRYSTALS-kyber, NTRU, and saber. *Cryptology ePrint Archive*, 2021.

[Dwo15] Morris J Dworkin. SHA-3 standard: Permutation-based hash and extendable-output functions. Technical report, 2015.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Annual international cryptology conference*, pages 537–554. Springer, 1999.

[GDD+22] Alagic Gorjan, Apon Daniel, Cooper David, Dang Quynh, Dang Thinh, Kelsey John, Lichtinger Jacob, Miller Carl, Moody Dustin, Peralta Rene, Perlner Ray, Robinson Angela, Smith-Tone Daniel, and Liu Yi-Kai. Status report on the third round of the NIST post-quantum cryptography standardization process. *https://csrc.nist.gov/publications/detail/nistir/8413/final*, 2022.

[GL21] Wenbo Guo and Shuguo Li. Area-efficient modular reduction structure and memory access scheme for NTT. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2021.

[Gro96] L. K. Grover. A fast quantum mechanical algorithm for database search. In *28th ACM Symposium on Theory of Computing*, pages 212–219, 1996.

[HHLW20] Yiming Huang, Miaoqing Huang, Zhongkui Lei, and Jiaxuan Wu. A pure hardware implementation of CRYSTALS-Kyber pqc algorithm through resource reuse. *IEICE Electronics Express*, pages 17–20200234, 2020.

[HRSS17] Andreas Hülsing, Joost Rijneveld, John M. Schanck, and Peter Schwabe. NTRU-KEM-HRSS17: Algorithm specification and supporting documentation. Submission to the NIST Post-Quantum Cryptography Standardization Project, 2017.

[KZW+22] Dur-e-Shahwar Kundi, Yuqing Zhang, Chenghua Wang, Ayesha Khalid, Maire O'Neill, and Weiqiang Liu. Ultra high-speed polynomial multiplications for lattice-based cryptography on FPGAs. *IEEE Transactions on Emerging Topics in Computing*, pages 1–1, 2022.

[Mil85] Victor S Miller. Use of elliptic curves in cryptography. In *Conference on the theory and application of cryptographic techniques*, pages 417–426. Springer, 1985.

[Moo16] D. Moody. Post-quantum cryptography: NIST's plan for the future. In *Talk given at PQCrypto'16 Conference*, 2016.

[PP19] Chris Peikert and Zachary Pepin. Algebraically structured LWE, revisited. In *Theory of Cryptography Conference*, pages 1–23. Springer, 2019.

[RD01] Vincent Rijmen and Joan Daemen. Advanced Encryption Standard. *Proceedings of Federal Information Processing Standards Publications, National Institute of Standards and Technology*, pages 19–22, 2001.

[Riv78] R. L. Rivest. A method of obtaining digital signatures and public key cryptosystems. *Communications of the ACM*, 120, 1978.

[Sho99] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. volume 41, pages 303–332, 1999.

[TCW+21] Weihang Tan, Benjamin M. Case, Antian Wang, Shuhong Gao, and Yingjie Lao. High-speed modular multiplier for lattice-based cryptosystems. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 68(8):2927–2931, 2021.

[XL21] Yufei Xing and Shuguo Li. A compact hardware implementation of CCA-secure key exchange mechanism CRYSTALS-Kyber on FPGA. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 328–356, 2021.

[YM21] Ferhat Yarman, Ahmet Can Mert, Erdinç Öztürk, and Erkay Savaş. A hardware accelerator for polynomial multiplication operation of CRYSTALS-Kyber PQC scheme. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1020–1025, 2021.

[ZLL+21] Cong Zhang, Dongsheng Liu, Xingjie Liu, Xuecheng Zou, Guangda Niu, Bo Liu, and Quming Jiang. Towards efficient hardware implementation of NTT for Kyber on FPGAs. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2021.

[ZZY+21] Yihong Zhu, Min Zhu, Bohan Yang, Wenping Zhu, Chenchen Deng, Chen Chen, Shaojun Wei, and Leibo Liu. LWRpro: An energy-efficient configurable crypto-processor for module-LWR. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 68(3):1146–1159, 2021.