IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 22 May 2025, accepted 10 June 2025, date of publication 13 June 2025, date of current version 20 June 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3579379

RESEARCH ARTICLE

An Area-Time Efficient Hardware Architecture for ML-KEM Post-Quantum Cryptography Standard

TRONG-HUNG NGUYEN, (Graduate Student Member, IEEE), TUAN-KIET DANG, (Graduate Student Member, IEEE), DUC-THUAN DAM, (Graduate Student Member, IEEE), KHAI-DUY NGUYEN, (Graduate Student Member, IEEE), PHUC-PHAN DUONG, (Graduate Student Member, IEEE), CONG-KHA PHAM, (Senior Member, IEEE), AND TRONG-THUC HOANG, (Member, IEEE)

Department of Computer and Network Engineering, The University of Electro-Communications (UEC), Tokyo 182-8585, Japan

Corresponding author: Trong-Thuc Hoang (hoangtt@uec.ac.jp)

ABSTRACT To facilitate the integration of the NIST-standardized post-quantum cryptographic (PQC) algorithm, Module Lattice-based Key Encapsulation Mechanism (ML-KEM), into quantum-resistant devices and cryptosystems, this study introduces an area-time efficient hardware implementation. We propose several key optimizations: a dedicated SHA3 design merging hashing and sampling to eliminate hash output storage, a high-performance polynomial arithmetic unit with a novel reuse technique that reduces twiddle factor storage by threefold, and a partitioned storage strategy to minimize memory footprint. By tightly coupling pipelining and parallelism, we further enhance execution speed and avoid memory access bottlenecks. Our FPGA-based ML-KEM implementation achieves a superior area-time product (ATP), outperforming state-of-the-art results by up to 6.7× across NIST security levels 1, 3, and 5. A 180 nm CMOS fabrication of our design yields a die area of $1{,}940 \times 1.789\ \mu m^2$. Measured performance shows the lowest SRAM requirement (7.37 kB) and minimal energy consumption (6.4 μJ, 10.3 μJ, and 14.4 μJ for security levels 1, 3, and 5), leading to ATP improvements ranging from 1.5× to 115.0× compared to existing ASIC solutions.

INDEX TERMS Post-quantum cryptography (PQC), module lattice-based key encapsulation mechanism (ML-KEM), FIPS 203, SHA3, CRYSTALS-Kyber, hardware architecture, FPGA, ASIC.

I. INTRODUCTION

Following extensive evaluation, the cryptographic community and NIST released the first three post-quantum encryption standards in 2024 [1]. Among these, the Module-Lattice-Based Key Encapsulation Mechanism (ML-KEM), standardized as FIPS 203 [2], stands out as the primary algorithm for general encryption. ML-KEM facilitates the creation of shared secrets for symmetric-key cryptography and offers three parameter sets – ML-KEM-512, ML-KEM-768, and ML-KEM-1024 – to suit various application scenarios. The release of ML-KEM signifies a major advancement in securing data against the looming threat of quantum computers, necessitating its prompt adoption and integration into existing systems.

ML-KEM's foundation lies in the CRYSTALS-Kyber algorithm [3], sharing its core operations but with differences in input and output handling. For clarity, we will use ML-KEM notation for general discussions and ML-KEM/CRYSTALS-Kyber when referencing prior work. The optimal implementation of ML-KEM is application-dependent. While software implementations of CRYSTALS-Kyber are prevalent for near-term deployment [4], [5], [6], [7], [8], [9], [10], they inherently suffer from performance bottlenecks and security risks associated with software layer vulnerabilities. Hardware-software (HW/SW) co-design offers a balance of performance and flexibility [11], [12], [13], [14], [15], [16] by accelerating

The associate editor coordinating the review of this manuscript and approving it for publication was Mario Donato Marino.

critical operations in hardware. However, both software and HW/SW approaches experience significant performance degradation when countermeasures against side-channel attacks (SCAs) are implemented [17].

Hardware-based designs, in contrast, excel in performance [14], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29], [30] by implementing the entire cryptographic algorithm as dedicated hardware IP cores on FPGA or ASIC platforms. This hardware isolation provides a strong security foundation for sensitive data. Furthermore, hardware offers inherent resistance to SCAs, notably timing attacks due to fixed execution times, and allows for efficient implementation of other SCA countermeasures [31], [32], [33], [34]. Therefore, for robust and reliable security, a hardware-centric approach to ML-KEM implementation is the preferred strategy.

The fundamental computational tasks within ML-KEM involve generating randomness polynomials and performing polynomial arithmetic. Randomness generation relies on uniform sampling or centered binomial distribution, driven by pseudorandom bytes derived from the Secure Hash Algorithm 3 (SHA3) standard, specifically the Keccak-f[1600] permutation [35]. Polynomial arithmetic is efficiently executed using the Number Theoretic Transform (NTT), which reduces the complexity of polynomial multiplication to a quasi-linear $O(n \times \log n)$, where n is the polynomial degree. Consequently, efficient hardware implementations of both SHA3 and NTT are crucial for optimizing the overall performance of ML-KEM.

A. RELATED WORKS

Over the past few years, numerous hardware implementations of ML-KEM/CRYSTALS-Kyber have emerged on both FPGA [18], [19], [20], [21], [23], [24], [25], [26], [27], [28], [29], [30] and ASIC [11], [12], [13], [14], [20], [22] platforms. FPGA-based efforts include Xing et al.'s compact design with a dual-butterfly unit (BU) NTT accelerator and Karatsuba-based pointwise multiplication reduction, along with memory sharing between SHA3 and NTT [18]. Bisheh et al. focused on a high-speed NTT with a 2×2 BU configuration tailored to security levels [19], later extending it to a customizable instruction set architecture, albeit with high cost and slow execution [20]. Dang et al. further optimized Xing's NTT design, implementing k accelerators for each security level, achieving improved performance [21]. Ni et al. introduced a memory-efficient multi-path delay commutator NTT with enhanced speed but increased resource usage [23]. Li et al. demonstrated the advantages of a split-radix NTT for high-speed implementations, though with inefficient data storage [24]. Guo et al. explored mixed radix-2/4 NTT with conflict-free memory access [25] and subsequently a split-radix NTT-based design [26]. Kim et al. presented a configurable ML-KEM design supporting all security levels at high speeds but with significant hardware overhead [27]. Ni et al. also proposed a flexible architecture with FIFO-based storage and layered computation for faster parallel processing and lower cost [28]. Nguyen et al. presented a design aimed at resource-constrained devices [29]. This design achieved the smallest footprint to date by proposing a compact, dedicated SHA3 and a non-memory-based iterative (NMI) NTT. Recently, Cui et al. proposed an instruction-based design featuring high performance and flexibility [30]. The functional modules were optimized and controlled via instructions, significantly enhancing operating frequency and hardware efficiency.

ASIC implementations include Banerjee et al.'s HW/SW co-design with a RISC-V controlled crypto-processor, outperforming other co-design approaches [11]. Fritzmann et al. designed tightly coupled accelerators with a RISC-V processor and specialized instructions, but suffered from slow execution and high SRAM usage [12]. Xin et al. proposed a vector processor for NTT and sampling, extending the RISC-V instruction set for fast execution but with a large area footprint [13]. Bisheh et al.'s ASIC implementation also exhibited high SRAM requirements [20]. Zhao et al. developed a RISC-V controlled hardware crypto-core with a 2D systolic array for fast NTT computation [14]. Kali et al. presented a larger area design combining CRYSTALS-Kyber and CRYSTALS-Dilithium [22].

Our literature review reveals a predominant focus on optimizing the low-level polynomial arithmetic operations in prior ML-KEM/CRYSTALS-Kyber hardware implementations. These operations include integer multiplication and modular reduction for modular arithmetic in butterfly and point-wise operations [36], [37], [38]. Subsequently, NTT designs have been extensively investigated regarding coefficient memory costs and the execution time of NTT, INTT, and point-wise multiplication (PWM). The efficient design of these operations significantly enhances overall performance. However, these studies have not fully explored optimizing designs at the algorithmic level, resulting in limited hardware efficiency.

Specifically, the SHA3 module, despite its substantial hardware cost, has received comparatively less attention. Most existing works adopt a standard SHA3 design that requires significant hardware for input, output data buffers, and sampling [12], [14], [20], [21], [25]. Hung et al. introduced a compact SHA3 based on a configurable multi-mode state A design for input/output data loading and executing the Keccak-f[1600] permutation [29]. However, its limitation is the sequential execution of different modes, which slows down hashing and sampling. Besides that, storing all hash data during sampling incurs a memory cost. Reference [30] also proposes a low-cost SHA3 design with equivalent hashing and sampling speed to [29] due to the need to sequentially update state A with external data and the internal permutation. However, they utilized an output FIFO buffer for sampling, resulting in lower memory cost than [29]. The common limitation of both SHA3 designs is that they are inefficient for high-performance applications.

Moreover, executing data flow between computational modules is a critical issue. A widely used approach involves independently storing data for each module, enabling direct data exchange and simplified control. However, this approach trades off memory cost and may lead to stall time due to inter-module data dependencies. Some references have reported high memory usage, which increases significantly at higher security levels [19], [20], [21], [24], [27]. In some ASIC implementations, reported SRAM usage is significantly larger than the necessary storage for primary cryptographic data (e.g., seeds, keys, and ciphertext) [11], [12], [14], [20], [22].

Furthermore, several references have proposed optimizing data flow through buffer memory reuse [18], [23], [28], [29], [30]. However, these designs still require initializing and storing intermediate data before use. For instance, the design in [29] requires separate storage for hash data and polynomial sampling results. The design in [30] also requires that polynomials be pre-sampled and stored before computation. Thus, these memory buffers must be sufficiently large to support intermediate data storage across all security levels.

A comprehensive approach that thoroughly exploits and tightly integrates optimizations for SHA3, NTT, and data flow is essential to address the aforementioned issues. Therefore, the main objective of this paper is to focus on hardware optimization at the algorithmic level to achieve superior area-time efficiency. The primary innovations of this work are:

- An area-optimized SHA3 architecture that tightly integrates the hashing and sampling stages. This is achieved by directly feeding the pseudorandom byte stream output from the hashing process back into the input FIFO. Consequently, sampling operations can proceed in parallel with subsequent hashing computations, effectively eliminating the need for dedicated SHA3 output memory while maintaining the throughput of both operations.
- The design of two high-performance and memory-efficient NTT accelerators, one for the even and one for the odd sub-polynomials of ML-KEM. Each accelerator incorporates two butterfly units to enhance processing speed. Furthermore, we propose a novel method to derive the required twiddle factors for the inverse-NTT ζ^{-i}, and pointwise multiplication $\zeta^{2\text{BitRev}_7(i)+1}$ directly from the twiddle factors used in the NTT operation ζ^i. This approach achieves a threefold reduction in the storage requirements for twiddle factors.
- We implement a partitioned storage scheme to enhance memory utilization, propose intermediate data overwriting to reduce memory cost, and adopt a tightly coupled pipelined-parallel strategy to accelerate the overall execution time substantially.
- The integration of the aforementioned optimizations into a complete ML-KEM design that supports all three NIST security levels. Evaluation on an FPGA platform demonstrates a 6.7× improvement in area-time product (ATP) compared to existing FPGA implementations, with a resource footprint of 7.6k LUTs, 5.7k FFs, 2.3k Slices, 4 DSPs, and 3 BRAMs.
- The fabrication and measurement of the proposed ML-KEM design in 180 nm CMOS technology. The resulting ASIC occupies an area of $1{,}940 \times 1.789\mu\text{m}^2$ (equivalent to 160 kGE and utilizing 7.37 kB of SRAM). Measurement results at a 1.8 V supply voltage and 93 MHz operating frequency show energy consumption of 6.4 μJ, 10.3 μJ, and 14.4 μJ for ML-KEM-512, ML-KEM-768, and ML-KEM-1024, respectively. This leads to an area-time product that is 1.5× to 115.0× better than previously reported ASIC implementations.

The subsequent sections of this paper are organized to guide the reader through our work. Section II provides the necessary background information on the post-quantum cryptography standard, ML-KEM. Section III then delves into the specifics of our main contributions. The hardware architecture of our ML-KEM design for the ASIC platform is presented in Section IV. Section V follows with a discussion of the implementation and measurement results, including comparisons with existing work. Finally, Section VI concludes the paper and outlines potential future research directions.

II. PRELIMINARY

A. ML-KEM STANDARD

ML-KEM is a key encapsulation mechanism derived from CRYSTALS-KYBER [3], an algorithm selected by NIST after the third round of the post-quantum cryptography standardization process [39]. It is defined in the NIST FIPS 203 standard [2], released in August 2024. The security of ML-KEM is based on the hardness of the Module-LWE (MLWE) problem [40]. ML-KEM is constructed in two steps. Initially, a public-key encryption (PKE) scheme is built, which includes the key generation (K-PKE.KeyGen), encryption (K-PKE.Encrypt), and decryption (K-PKE.Decrypt) algorithms. Then, an IND-CCA2-secure ML-KEM is constructed by using the Fujisaki-Okamoto (FO) transform [41].

ML-KEM features a modular design with algebraic operations on the polynomial ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$. This design utilizes k-dimensional matrices and vectors, where $k \in (2, 3, 4)$. Hence, ML-KEM provides flexible security levels by adjusting the parameter k, corresponding to ML-KEM-512, ML-KEM-768, and ML-KEM-1024. We refer to the FIPS 203 standard, [2], for detailed ML-KEM specifications.

B. SHA3 STANDARD

ML-KEM uses the secure hash algorithm SHA3 to generate randomness polynomials from symmetric primitives. Each primitive is initialized using a hash function (SHA3) or an extendable output function (XOF), as specified in the FIPS202 standard [35]. ML-KEM employs two hash

functions, SHA3-256 and SHA3-512, along with two XOF functions, SHAKE128 and SHAKE256. These functions are based on the Keccak-f[1600] permutation with a 1600-bit state. The differences between these functions relate to the rate parameter r, the padding rule, and the length of truncated output.

C. NUMBER THEORETIC TRANSFORM

The Number Theoretic Transform (NTT) is an integral part of ML-KEM. Specifically, polynomials are converted to their NTT representation before performing arithmetic operations. The results are then converted back to the normal representation. NTT accelerates polynomial multiplication with a complexity of $O(n \times \log n)$. ML-KEM utilizes the negative-wrapped convolution (NWC) NTT [42]. Consequently, the NTT representation of a polynomial f consists of two coefficient vectors, each being 128 degree-one residues modulo quadratic polynomials. Specifically,

$$f \bmod (x^2 - \zeta^{2\text{BitRev}_7(i)+1}) = \hat{f} = \hat{f}_{2i} + \hat{f}_{2i+1}x \tag{1}$$

for i from 0 to 127, ζ is the 256-th root of unity in $\mathbb{Z}_q$, and BitRev_7 being the bit reversal of a seven-bit integer. The vectors $\hat{f}_{2i}$ and $\hat{f}_{2i+1}$ are computed as follows:

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

$$\hat{f}_{2i+1} = \sum_{j=0}^{127} f_{2j+1}\zeta^{(2\text{BitRev}_7(i)+1)j} \tag{3}$$

Polynomial multiplication $h = f \times g$ involves independent pointwise multiplications (PWM) between 128 degree-one vectors, following the equation:

$$\begin{aligned}&\hat{h}_{2i} + \hat{h}_{2i+1}x \\ &= (\hat{f}_{2i} + \hat{f}_{2i+1}x)(\hat{g}_{2i} + \hat{g}_{2i+1}x) \bmod (x^2 - \zeta^{2\text{BitRev}_7(i)+1})\end{aligned} \tag{4}$$

From an implementation perspective, NTT exhibits a highly parallel structure. Thus, the Cooley-Tukey (CT) [43] and Gentleman-Sande [44] algorithms are widely known for fast NTT transformation. These algorithms adopt the divide-and-conquer strategy to compute butterfly operations at each NTT stage iteratively. References [45] and [46] present techniques to eliminate pre-processing and post-processing, reducing costly bit-reversal operations. Besides, reference [11] proposes a unified BU architecture that integrates CT and GS algorithms, facilitating a more flexible NTT design.

III. AREA-TIME EFFICIENT ML-KEM ARCHITECTURE

Figure 1 illustrates the overall architecture of the proposed area-time efficient ML-KEM design. Our architecture consists of three computational modules: a SHA3 module for hashing and sampling operations, a polynomial arithmetic unit (PAU), and a Transcoder module for Encode/Decode and Compress/Decompress operations. Besides, a memory bank (RAM0, RAM1, RAM2) is designed to store data during ML-KEM operations. The width of internal data paths is 64 bits. A Control Unit is designed to manage the operations of all modules.

This design functions as a crypto-core for the ML-KEM standard. The input data includes the operation mode (ML-KEM.KeyGen, ML-KEM.Encaps, ML-KEM.Decaps), security level (k), randomness seeds (d, z, m), and control signals (Reset, Enable). The output data consists of the keys (ek, dk, K), ciphertext (c), and status signals (Valid, Done).

The design idea is that all output data from the computational modules must be stored in the partitioned memories of the memory bank. On the contrary, the data needed for computation are directly read from the memory bank. In the case of PAU, a $1n$ memory unit, equivalent to storing one polynomial, is added to support performing polynomial multiply-accumulates with the MAC unit. This approach significantly reduces the overall memory cost of the system.

However, to ensure the effectiveness of this design, the operations of functional modules must be tightly scheduled and executed sequentially to simplify memory access patterns and avoid stall time. In the following sections, we will explore optimization strategies for each module and how their operations are coordinated to achieve high area-time efficiency.

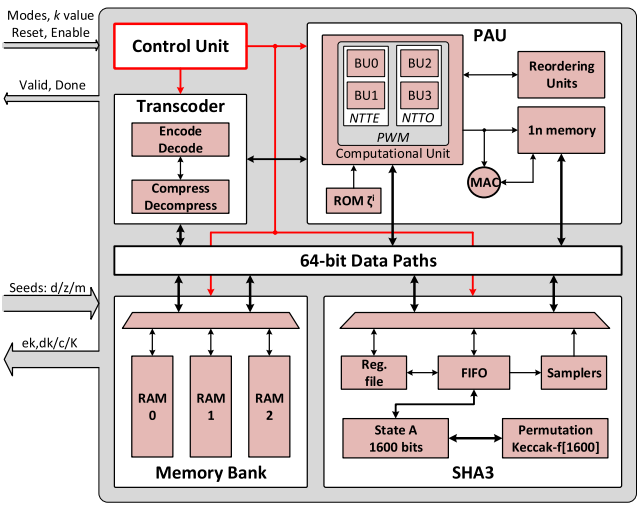


FIGURE 1. The proposed architecture of the ML-KEM standard.

A. THE PARTITIONED MEMORY BANK

Storing data is a challenge when implementing ML-KEM. Especially ML-KEM has many intermediate values, including initial seeds, pseudo-random bit streams, and randomness polynomials. In [18], [21], and [23], hardware resources have been wasted on storing intermediate values in FIFOs. Moreover, reference [28] utilized FIFOs to store all data types, achieving a 100% memory utilization ratio. In reference [29], a fragmented memory approach is proposed to improve the BRAM utilization ratio. However, separately storing multiple

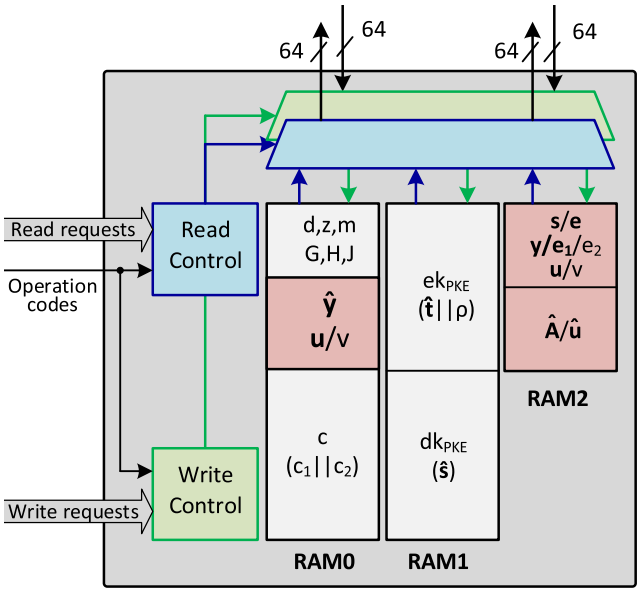


FIGURE 2. The organization of the partitioned memory bank.

intermediate values increases hardware costs. Besides that, in the ML-KEM standard, NIST requires that intermediate values be destroyed after use to prevent attackers from compromising them [2]. Consequently, multiple intermediate memory units make destruction a concern.

To address these issues, we propose a memory bank architecture, as shown in Figure 2. All data in the ML-KEM is stored in three partitioned memory blocks: RAM0, RAM1, and RAM2. The light red partitions represent the memory areas where intermediate data is destroyed by overwriting. Specifically, RAM0 is divided into three storage partitions: 1) the randomness seeds (d, z, m) and the results of symmetric primitives (G, H, J); 2) intermediate polynomials $\hat{\mathbf{y}}$ or u/v from Decompress(u)/Decompress(v) operation; 3) ciphertext $c(c1||c2)$. RAM1 has two storage partitions for an encryption key, $\text{ek}_{PKE}(\hat{\mathbf{t}}||\rho)$ and a decryption key, dk_{PKE} $(\hat{\mathbf{s}})$. Besides that, RAM2 also includes two storage partitions for storing the most intermediate polynomials for the component schemes: K-PKE.KeyGen $(\mathbf{s}, \mathbf{e}, \hat{\mathbf{A}})$, K-PKE.Encrypt $(\mathbf{y}, \hat{\mathbf{A}}, \mathbf{e_1}, e_2)$, and K-PKE.Decrypt (Decompress(u), Decompress(v), $\hat{\mathbf{u}}$). RAM0 and RAM1 are 512×64 bits, while RAM2 is 256×64 bits, ensuring support for all three security levels.

The seeds (σ, r) also serve as intermediate values for iterative generation processes $(\mathbf{s}, \mathbf{e}, \mathbf{y}, \mathbf{e_1}, e_2)$. Therefore, we store them in the register file within the SHA3 module. This method simplifies memory bank access patterns and allows destruction using a reset signal once they are no longer needed. On the other hand, output data from the SHA3 module is stored directly in the memory bank without further processing.

When interfacing with the memory bank, the computation modules must send read and write request signals and operation codes. The Read/Write Control units then encode this information into control signals for the multiplexers, determining the appropriate input/output ports and memory regions for data interaction. For example, when the SHA3 module issues a read or write request, the operation codes correspond to the specific hashing functions being performed by SHA3. For intermediate data that is overwritten, the associated operation codes correspond to operations such as NTT, INTT, PWM, Compress, and Decompress. The memory bank can also handle external read/write requests for initial seeds (d, z, m), keys (ek_{PKE}, dk_{PKE}, shared key K), and ciphertext (c).

B. SHA3 MODULE

Conventional SHA3 designs require two large FIFO buffers to store and process the input and output data [21], [47]. In ML-KEM, the size of these buffers can reach up to $r = 1344$ bits for the SHAKE128 function. Therefore, we propose a more efficient SHA3 design by introducing a multifunctional FIFO buffer that helps to couple hashing and sampling operations tightly. Figure 3a details the proposed SHA3 architecture with the 64-bit data path. The design includes a FIFO, a 1600-bit state A, a Keccak-f[1600] permutation unit, a padding unit, and samplers. Moreover, Figure 3b shows the FIFO design, which consists of 21 64-bit registers and interfaces with state A through modes 0, 1, and 2. Specifically, in mode 2, data is loaded into the FIFO, or hashed output is read out from the FIFO. In mode 1, the FIFO is updated with hashed data from state A in one clock cycle (CC). In mode 0, r-bit data from state A is XORed with the r-bit FIFO in one CC.

In ML-KEM, the operation of the SHA3 module can be cataloged into two types: performing hash functions (SHA3-512, SHA3-256) and performing XOF functions (SHAKE128, SHAKE256). For hash functions, input data is padded and loaded into the FIFO. Once FIFO is filled with r bits, the state A is updated to perform the Keccak-f[1600] permutation. Each Keccak-f[1600] round requires 24 CCs, while FIFO loading takes at most 21 CCs. Therefore, in the case of long data hashing (e.g., H, J primitives), loading new r-bit data into the FIFO is effectively hidden by overlapping with the Keccak-f[1600] permutation.

For the sampling process in ML-KEM, the XOF functions only require one-time input loading but demand multiple rounds of Keccak-f[1600] permutations. After each round, r bits from state A are updated into the FIFO using mode 1. These pseudorandom bits are then read out for sampling using mode 2. Simultaneously, a new Keccak-f[1600] permutation is performed on state A in 24 CCs. Once all pseudorandom bits are utilized for sampling, the FIFO is re-updated with new hashed data from state A within one CC and continues the sampling process. In our design, the uniform sampling time can be mainly hidden by the Keccak-f[1600] permutation, while the centered binomial distribution sampling is entirely absorbed. Consequently, our SHA3 design saves one output r-bit FIFO while maintaining hashing and sampling throughput as usual.

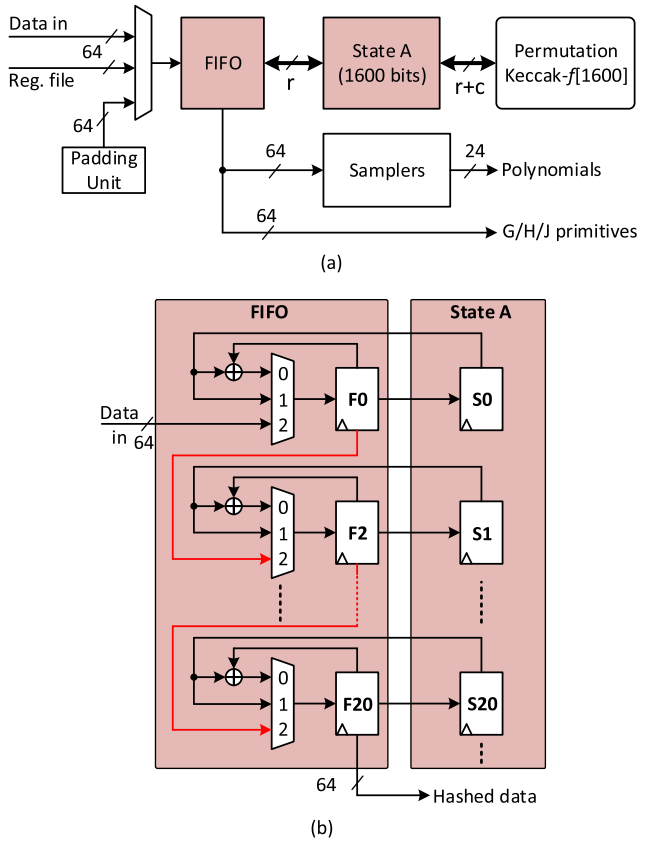


FIGURE 3. The proposed SHA3 module: (a) The top level of the SHA3 architecture, (b) The design of FIFO and its modes.

We design four independent sample-NTT units for sampling the polynomial $\hat{A}_{ij}$. Each unit has an acceptance rate of 81.27%. Valid samples are stored in four small-depth FIFOs (depth = 12). When each FIFO has at least four valid samples, they are read in batches of four coefficients and stored directly in the memory bank. Our SHA3 module generates a polynomial $\hat{A}_{ij}$ in approximately 120 CCs. Besides that, we sample eight coefficients per CC for the polynomial f using CBD functions. The coefficients are generated in the form $-\eta \leq f[i] \leq \eta$, with 3-bit size, where the most significant bit (MSB) represents the sign. Thus, eight coefficients have 24 bits and can be stored directly in the memory bank without encoding. The proposed SHA3 module generates a polynomial f in 83 CCs and 89 CCs, corresponding to the CBD2 ($\eta = 2$) and CBD3 ($\eta = 3$) functions, respectively.

C. POLYNOMIAL ARITHMETIC UNIT

The PAU module is designed to perform polynomial operations for ML-KEM. As outlined in Figure 1, its architecture consists of a Computation Unit (CU), Reordering Units (RUs), a ROM for twiddle factor storage, and an internal memory unit ($1n$ memory).

The CU design includes four butterfly units (BU0, BU1, BU2, and BU3), where the pair (BU0, BU1) performs NTT/INTT for the even-coefficient sub-polynomial and the pair (BU2, BU3) for the odd-coefficient sub-polynomial. In addition, the four butterfly units can be reconfigured to perform PWM operations. The RUs support coefficient reordering and storage during the NTT/INTT process. The $1n$ memory unit comprises four 32×24 memory blocks, supporting the MAC operations in matrix-vector and vector-vector polynomial multiplications.

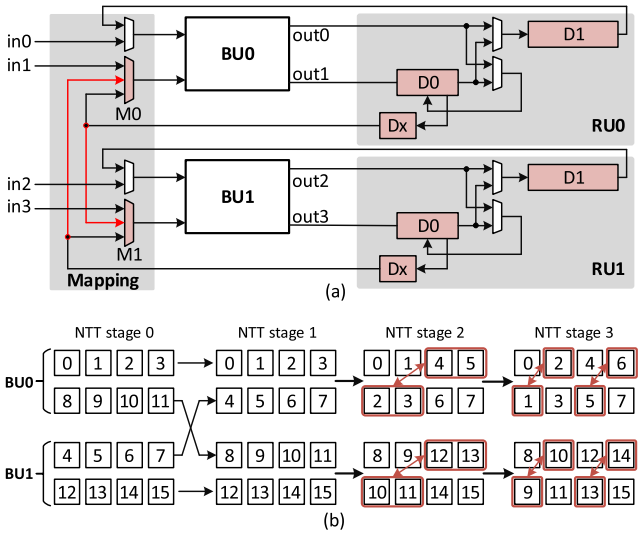


FIGURE 4. The proposed NTT accelerator architecture (a), an example data flow of 16-point NTT (b).

1) NTT DESIGN

For the NTT operation, we design two highly efficient NTT accelerators, NTTE for even-coefficient and NTTO for odd-coefficient sub-polynomials. These accelerators are optimized based on the NMI-NTT from [29]. The key difference is that we propose a scalable NTT design by introducing a mapping unit into the NMI-NTT architecture, as shown in Figure 4a, for the example of NTTE. Specifically, each accelerator consists of two parallel BU units (BU0 and BU1). RU0 and RU1 support coefficient reordering and storage at each NTT stage. M0 and M1 are responsible for distributing coefficients between the two BU units at a specific NTT stage.

Figure 4b illustrates the operation of the NTT accelerator through a data flow example for a 16-point NTT. Specifically, the coefficient streams are divided into two groups and processed by BU0 and BU. After the first NTT stage, half of the coefficient streams are redistributed directly between the two BUs without reordering. In the hardware design, two multiplexers, M0 and M1, are controlled by the NTT stage index to map the coefficients between BU0 and BU1 (the red data paths in Figure 4a).

In the subsequent NTT stages, the coefficients are processed and reordered within each pair of BU and RU. Notably, the size of RU0 and RU1 is reduced by half compared to [29] because reordering is not needed for the first NTT stage. Moreover, the design of the delay unit D1 does not require

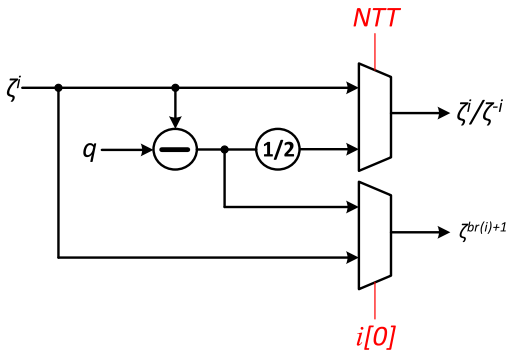


FIGURE 5. The hardware design for deriving ζ^{-i} and $\zeta^{2\text{BitRev}_7(i)+1}$ from ζ^i.

fragmentation, as in [29]. The size of D1 equals D0 + Dx, where $Dx = n/8 - x$, with x being the total number of pipeline registers in BU and RU. Therefore, RU0 and RU1 are compact and have a more straightforward control logic compared to [29].

The control indices for the NTT accelerator are derived from the NTT stage index. Therefore, they can be reversed to perform the INTT operation. Two NTT accelerators require 224 CCs for an NTT/INTT operation of one polynomial and 128 CCs for a PWM operation between two polynomials.

2) STORING TWIDDLE FACTOR

The NTT-based polynomial multiplier of ML-KEM requires three pre-computed twiddle factor arrays for NTT (ζ^i), INTT (ζ^{-i}), and PWM ($\zeta^{2\text{BitRev}_7(i)+1}$) operations. The ζ^i and ζ^{-i} arrays each contain $n/2 - 1$ powers of value ζ, while the $\zeta^{2\text{BitRev}_7(i)+1}$ array contains $n/2$ powers. Therefore, storing all three arrays requires about $1.5 \times n$. Several works have proposed methods to derive ζ^{-i} from ζ^i. Thus eliminating the need to store one array, with only an additional cost of one subtraction, as shown in the following equation:

$$\zeta^{-i} \equiv -\zeta^{n-i} \equiv q - \zeta^{n-i} \tag{5}$$

However, we found that ζ storage can be further reduced by deriving $\zeta^{2\text{BitRev}_7(i)+1}$ from ζ^i. Specifically, for even values $i = 2k$, $\zeta^{2\text{BitRev}_7(2k)+1}$ values can be derived directly from the ζ^i values of the final NTT stage. Moreover, for odd values $i = 2k + 1$, the value $2\text{BitRev}_7(i) + 1$ is given by:

$$2\text{BitRev}_7(2k+1) + 1 \equiv 2\text{BitRev}_7(2k) + 1 + n/2 \ (\text{mod} n) \tag{6}$$

By applying the binary property $\zeta^{k+n/2} \equiv -\zeta^k$ to equation (6), we obtain equation (7) as below:

$$\zeta^{2\text{BitRev}_7(2k+1)+1} \equiv q - \zeta^{2\text{BitRev}_7(2k)+1} \tag{7}$$

As a result, the $\zeta^{2\text{BitRev}_7(i)+1}$ can be derived from the ζ^i as in the ζ^{-i} case, making them unnecessary to store. Figure 5 illustrates the hardware cost of deriving ζ^{-i} and $\zeta^{2\text{BitRev}_7(i)+1}$ from ζ^i. Specifically, the ζ^{-i} twiddle factors are computed using one subtractor ($q - \zeta^i$) and followed by one 1/2 multiplier for the INTT operation. The NTT mode determines the selection between ζ^i and ζ^{-i} accordingly. The $\zeta^{2\text{BitRev}_7(i)+1}$ coefficients for odd i values are also computed using the subtractor ($q - \zeta^i$). Then, bit 0 of i value is used to determine the output of $\zeta^{2\text{BitRev}_7(i)+1}$. Besides that, generating the index for twiddle factor reading also requires a small hardware cost, involving multiplexers to select the read index of NTT/INTT/PWM mode. In summary, our method reduces ζ memory usage three times.

D. TRANSCODER MODULE

The data path in our design has a 64-bit width. Therefore, the Transcoder module is designed to encode the output of PAU into 64-bit data streams and write it into the memory bank. These data include the encryption and decryption keys (ek_{PKE}, dk_{PKE}), intermediate polynomials ($\hat{\mathbf{y}}$, $\hat{\mathbf{u}}$), and ciphertext ($c = \text{Compress}(u)||\text{Compress}(v)$). The Transcoder module also decodes the data (ek_{PKE}, dk_{PKE}, $\hat{\mathbf{y}}$, $\hat{\mathbf{u}}$) to perform PWM operations in the PAU. In the case of Decompress(u) and Decompress(v) operations, the decompressed data are stored back into the memory bank. These data (u, v) are stored in a specific order: four coefficients in RAM0 and the following four in RAM2. Then, the PAU can read eight coefficients (u, v) directly from the memory bank.

Algorithm 1 The Proposed Hardware-Friendly Design of the Compress Function

Input: The integer x, the parameters $q = 3329$ and d.
Output: $y = \text{Compress}_d(x)$.
1: $x_1 := (x \times 1260) >> (11 - d)$;
2: $x_1 := x_1 >> 11$;
3: $x_2 := (x << d) - x_1 \times 3329$;
4: **if** $x_2 > 1664$ **then**
5: $\quad x_1 := x_1 + 1$
6: **else**
7: $\quad x_1 := x_1$
8: **Return** $y = x_1 \bmod 2^d$

The hardware design of the Encode/Decode operations is relatively straightforward, with hardware costs consisting of some pipelined registers and a multiplexer. In contrast, the Compress operation is more complex and area-consuming since it requires a division of q. However, we exploit Barrett reduction to replace the division by q with a multiplication by an approximate inverse of q. We adopt a variant of the Barrett reduction proposed and analyzed in [37] and [48]. Algorithm 1 describes the low-cost design for the Compress operation with bit-shifting, subtraction, and addition operations. Notably, the Decompress operation can also be performed by reusing the multiplication with $q = 3329$ at Line 3 in Algorithm 1. Therefore, the proposed Transcoder design can flexibly configure its functions while maintaining low hardware overhead.

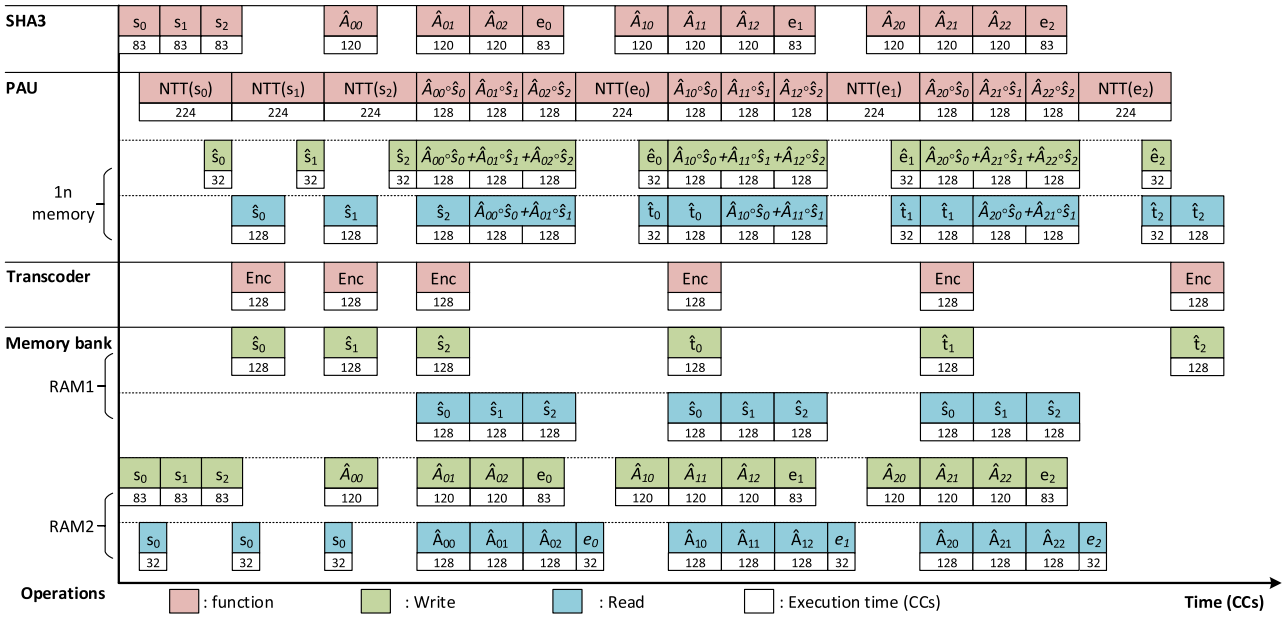


FIGURE 6. The optimization execution data flow for primary operations of K-PKE.KeyGen in ML-KEM-768.

E. OPTIMIZING EXECUTION DATA FLOW

The partitioned memory strategy is cost-advantageous due to its enhanced utilization ratio. However, it requires sequential memory access. All intermediate and result data are stored in partitions within memory banks. In particular, intermediate data is overwritten to ensure it is destroyed when no longer needed. Therefore, the execution order of operations must be tightly scheduled to avoid excessive storage of intermediate data and memory access stall time.

Figure 6 illustrates how we execute operations for the K-PKE.KeyGen scheme of ML-KEM-768. The most time-consuming operation is the matrix-vector polynomial multiplication. Therefore, the NTT, PWM, and INTT operations are identified as primary operations and executed sequentially in the PAU. Other operations, sampling, encoding, and accumulation, are performed in parallel with the primary operations. Therefore, their execution time is absorbed mainly by the PAU activities.

In particular, NTT($\mathbf{s}$) operations are executed first, and their results should be stored in RAM1. Our NTT accelerator generates eight output coefficients per one CC, corresponding to 32 CCs per polynomial. However, immediately processing and storing eight coefficients (96 bits) into 64-bit width memory is impractical. Hence, $\hat{\mathbf{s}}_0/\hat{\mathbf{s}}_1/\hat{\mathbf{s}}_2$ is temporarily stored in $1n$ memory and read out in parallel during the subsequent operation. Reading one polynomial takes 128 CCs, constrained by the execution time of one PWM operation. This constraint helps fix the number of input-output coefficients and simplifies the Transcoder design. Afterwards, the encoded $\hat{\mathbf{s}}_0$, $\hat{\mathbf{s}}_1$, $\hat{\mathbf{s}}_2$ are stored in RAM1.

In the next phase, the PAU sequentially executes $\hat{t}_0 = \hat{\mathbf{A}}_{0\mathbf{j}} \times \hat{\mathbf{s}} + \hat{e}_0$, $\hat{t}_1 = \hat{\mathbf{A}}_{1\mathbf{j}} \times \hat{\mathbf{s}} + \hat{e}_1$, and $\hat{t}_2 = \hat{\mathbf{A}}_{2\mathbf{j}} \times \hat{\mathbf{s}} + \hat{e}_2$ of the $\hat{\mathbf{A}} \times \hat{\mathbf{s}} + \hat{\mathbf{e}}$ operation. The PWM operations are performed first and accumulated using the MAC unit, then stored into the $1n$ memory. Afterward, the polynomial e_i is transformed into the NTT representation to complete $\hat{t}_i = \hat{\mathbf{A}}_{\mathbf{ij}} \times \hat{\mathbf{s}} + \hat{e}_i$. The result $\hat{t}_i$ is then read out at the subsequent operation, following the same 128 CC constraint as in the case of $\hat{\mathbf{s}}$. In our design, data is delayed one CC before being stored in $1n$ memory. This technique enables efficient MAC operations. For example, when $\hat{t}_0 = \hat{\mathbf{A}}_{0\mathbf{j}} \times \hat{\mathbf{s}}$, $\hat{t}_0$ is read from $1n$ memory at the same time the result of NTT(e_0) appears. Then, performing $\hat{t}_0 = \hat{t}_0 + \hat{e}_0$ and storing its results into $1n$ memory after one CC. The ready output signals from NTT and PWM operations are used to read data for MAC operations when needed.

Additionally, our SHA3 module samples polynomials faster than the PAU operations. Therefore, we only generate polynomials when required for the next operation. This approach allows these intermediate polynomials to be stored and overwritten in two partitions of RAM2. The above optimizations can be similarly applied to the K-PKE.Encrypt and K-PKE.Decrypt schemes. Hence, the NTT, PWM, and INTT operations remain the primary ones and are executed sequentially. Moreover, other operations, such as sampling, encoding/decoding, and compression/decompression, are overlapped with PAU operations. As a result, the total execution time for key generation, key encapsulation, and key decapsulation is substantially reduced.

In the ML-KEM standard, the security level can be adjusted by changing the dimensions k of the matrix and vector polynomials, which results in different numbers of operations. Based on the number of NTT, INTT, and PWM operations, our design can be flexibly configured to support all three security levels (ML-KEM-512, 768, and 1024) with a low additional cost.

IV. ASIC DESIGN

We fabricated the proposed ML-KEM design on a general-purpose 180 nm CMOS technology. Our ASIC supports all three security levels: ML-KEM-512, ML-KEM-768, and ML-KEM-1024. The input (seeds $d/z/m$) and output ($ek, dk/c/K$) data paths are changed from 64-bit to 8-bit to reduce the number of I/O pins required for the ASIC. This change only affects the initial data loading and final result readout time. However, we are restricted to using only the SRAM libraries provided by the manufacturer. Specifically, the largest size for dual-port SRAM is 128 × 32 bits (512B), and the maximum supported data width is 32 bits. Consequently, we had to divide our memory banks into provided memory units in smaller sizes. Figure 7 illustrates how data are stored in the supported SRAMs.

Specifically, for ML-KEM-768 and ML-KEM-1024, the encryption key ek_{PKE}, decryption key dk_{PKE}, and ciphertext c sizes exceed 128 × 32 bits. Therefore, ek_{PKE} and c are stored in four single-port SRAMs of size 256 × 32, corresponding to (ek0, ek1) and (ct0, ct1). Since dk_{PKE} requires simultaneous read and write access, we utilize four dual-port SRAMs of size 128 × 24 (dk0, dk1, dk2, dk3). When the PAU requires dk_{PKE}, eight coefficients are read in one CC.

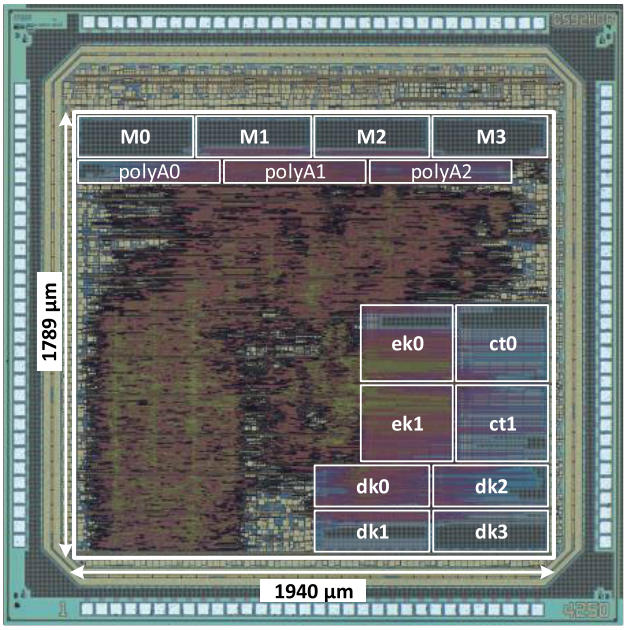
Intermediate polynomials ($\mathbf{s}, \mathbf{e}, \mathbf{y}, \mathbf{e_1}, e_2, \hat{\mathbf{y}}, \hat{\mathbf{u}}$) from the RAM0 and RAM2 partitions are stored in four dual-port SRAMs of size 128 × 24 (M0, M1, M2, and M3). Besides that, the memory partition for the $\hat{A}_{ij}$ polynomials in RAM2 is implemented using three dual-port SRAMs of size 64 × 16 (polyA0, polyA1, polyA2), sufficient for storing up to two polynomials. These intermediate data are overwritten during algorithm execution.

There are exceptions where data sizes do not match the provided SRAMs, such as memory partitions for data d, z, m, G, H, and J in RAM0 and the $1n$ memory of the PAU. These memories are synthesized using DFP standard cells instead of SRAM. These memory solutions increase the ASIC area. Our ASIC design uses 7.37 kB of SRAM with a comparable execution speed to the FPGA design.

V. IMPLEMENTATION RESULTS AND COMPARISONS

A. FPGA IMPLEMENTATION RESULTS

This section reports the proposed highly area-time efficient ML-KEM implementation results on FPGA and ASIC platforms. Regarding FPGA implementation, we deployed the ML-KEM design on the Artix-7 platform with Xilinx Vivado 2022.2. The FPGA reports have been obtained from default settings. Our ML-KEM has a 169 MHz operating frequency on a hardware footprint of 7.6k LUTs, 5.7k FFs, 2.3k Slices, 3 BRAMs, and 4 DSPs. Table 1 details the FPGA resource breakdown, where SHA3 and PAU are the most expensive hardware costs. Our FPGA performs key generation, encapsulation, and decapsulation phases in ML-KEM within 10.6/13.6/21.3 μs for ML-KEM-512, 18.3/21.3/33.2 μs for ML-KEM-768, and 26.6/30.2/45.0 μs for ML-KEM-1024.



Operating voltage	0.9 to 2.0 V
Max. frequency (MHz)	20 to 106
Area (µm²)	3,470,000
Gate count (kGE)	160
Memory	7.37 kB
ML-KEM Instance	ML-KEM-512/768/1024

FIGURE 7. ML-KEM ASIC micrograph on the die size of 2.5 × 2.5 mm².

TABLE 1. The FPGA resource breakdown of the proposed ML-KEM architectures.

Module	LUTs	FFs	Slices	BRAMs	DSPs
SHA3	4043	3192	1158	0	0
PAU	2313	1630	815	0	4
Transcoder	457	381	153	0	0
Memory bank	561	164	106	3	0
Control and others	246	285	85	0	0
Total	7620	5652	2317	3	4

B. ASIC IMPLEMENTATION RESULTS

Figure 7 shows the fabrication results of our ML-KEM on 180 nm CMOS technology. The chip die size is 2.5×2.5 mm^2. Our ML-KEM occupies approximately 3.47 mm^2 of the die area, including 160 kGE for computational logic and 7.37 kB SRAM for memory. The maximum operating frequency (f_{MAX}) ranges from 20 to 106 MHz, corresponding to the operating voltage (VDD) from 0.9 V to 2.0 V. At the standard supply voltage, VDD = 1.8 V, the measured f_{MAX} is 93 MHz. In comparison, the PnR synthesis result is 101 MHz.

Figure 8 shows our measurement setup for testing the proposed ML-KEM design. The ML-KEM chip is mounted on a QFP160 socket with header pins for input and output signal connections. Two power supply devices provide operating voltages and measure the current consumption for the I/O pins and the ML-KEM core design. We also utilize a DE2-115 FPGA development board to interface with and control the chip via the Quartus Prime Standard

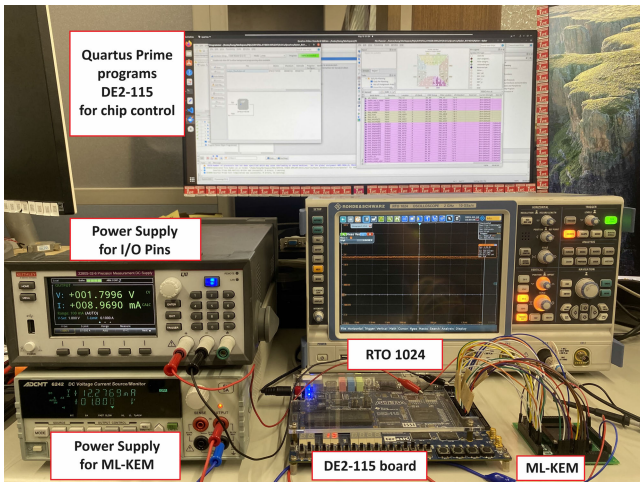


FIGURE 8. The measurement setup for ML-KEM ASIC chip.

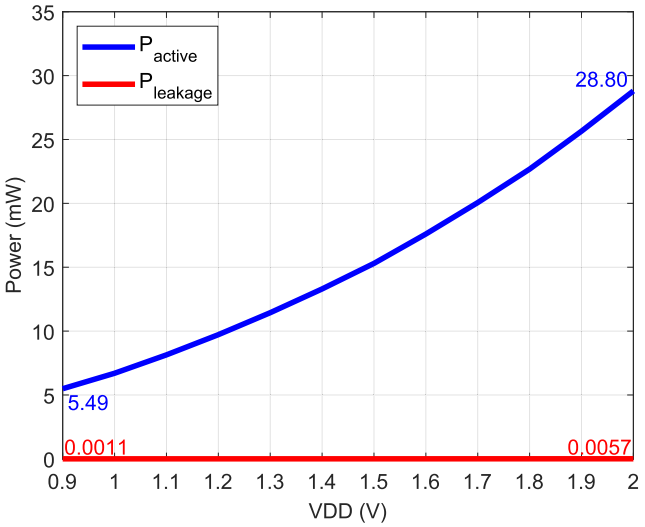


FIGURE 9. Power consumption at 20 MHz frequency and various supply voltages.

Edition software. Specifically, the test program is transferred onto the FPGA to send control signals (clock, enable, reset, modes, security level k) and initial seeds (d, z, m) to the chip via switches and jumper wires. The data exchange bandwidth between the FPGA and the chip is 8 bits. An RTO 1024 oscilloscope is used to observe the ciphertext comparison result in the final stage of decapsulation.

With this setup, we tested a complete ML-KEM internal algorithms sequence: Keygen ⇒ Encapsulation ⇒ Decapsulation. The DE2-115 board sends control and configuration signals to the ML-KEM chip. The operational status of ML-KEM is monitored and displayed via an LED on the DE2-115 board. The data (d, z, m) is automatically transmitted when the chip requests it. Upon completion of the execution processes, the design functions correctly if the high signal (VDD) can be observed on RTO 1024.

Figure 9 reports the measured power consumption of the ML-KEM chip at the lowest operation frequency f_{MAX} = 20 MHz with different VDDs under room temperature. The active power, P_{active} is measured when the ML-KEM chip performs key generation, encapsulation, and decapsulation phases. The leakage power, $P_{leakage}$ is measured without the operation clock. The lowest and highest P_{active} achieved are 5.49 mW at VDD = 0.9 V and 28.8 mW at VDD = 2.0 V.

C. COMPARISON WITH OTHER WORKS

Table 2 compares our results with the existing FPGA implementations of ML-KEM/CRYSTALS-Kyber. We report the area-time product (ATP) by Total Clock Cycles (CCs) × Equivalent Number of Slices (ENS) for a fair comparison at the algorithmic level. ENS is calculated by Slices + BRAMs×200 + DSPs×100, as in [29].

References [18], [26], and [29] propose CRYSTALS-Kyber implementations for resource-constrained devices. Therefore, their NTT accelerators utilize a dual-BU configuration for even and odd subpolynomials. Reference [18] adopts the Karatsuba algorithm to reduce five PWM operations to four. Hence, NTT can be reconfigured to perform PWM. However, [18] performs poorly due to considerable FIFO usage and an inefficient data flow. Reference [26] designs a split-radix NTT design, which is lower-cost than radix-2 NTT. However, its ALU requires significant BRAM, and the storage of the public key and ciphertext is not mentioned. Reference [29] has the lowest hardware cost by proposing a compact SHA3 design and a low-memory cost NTT. Nevertheless, this SHA3 design has very slow hashing and requires a $4r$-bit memory for sampling. In contrast, our SHA3 eliminates the output memory. Besides, a high-speed and low-memory cost NTT accelerator is designed. Thus, our ML-KEM has the lowest memory cost with 3 BRAMs. Reports show that our design performs ATP better at 1.8×, 1.4×, and 1.3× compared to [18], [26], and [29] across all three security levels.

References [19], [20], [21], [24], [25], [28], and [30] introduce high-performance ML-KEM/CRYSTALS-Kyber implementations. These implementations opt for highly parallel NTT accelerators. In [19], Bisheh et al. design an NTT with a 2 × 2 BU configuration and adopt k NTTs for CRYSTALS-Kyber with k as the security level. However, this implementation lacks system-level optimizations, leading to significant memory costs. Subsequently, the authors introduce a more memory-efficient version at the expense of slower execution time [20]. References [21] and [24] also implement k NTTs as in [19]. However, [21] focuses on optimizing the radix-2 NTT design and system architecture, while [24] extends the split-radix NTT approach. Consequently, the performance of [21] is better than that of [24]. Reference [25] designs a mixed-2/4 radix NTT for CRYSTALS-Kyber. It reports low hardware utilization without mentioning public key and ciphertext storage costs, as in [26]. References [28] and [30] also propose NTT accelerators with a similar number of BUs as ours. The design in [28] stores all data in FIFOs to enhance the utilization ratio. However, their intermediate data storage remains costly. In contrast, [30] introduced instruction sets to schedule

TABLE 2. FPGA implementation results and comparison with previous works.

Work[1]	Freq (MHz)	Cycle Count			Total time (μs)	Area					ENS[2] (k)	ATP[3]
		Keygen (kCCs)	Encap. (kCCs)	Decap. (kCCs)		LUTs (k)	FFs (k)	SLICEs (k)	BRAMs	DSPs		
ML-KEM-512, Security level 1 (Kyber512)												
[18]	161	3.8	5.1	6.7	95.2	7.4	4.6	2.1	3	2	2.9	45.2 (1.8×)
[19]	200	1.9	2.4	3.7	40.4	10.5	9.8	3.5	13	8	6.9	55.6 (2.2×)
[20]	115	4.0	7.0	10.0	182.7	18.0	5.0	5.0	6	15	7.7	161.7 (6.4×)
[21]	220	2.1	3.3	4.5	44.8	9.3	8.2	3.4	6	4	5.0	49.1 (1.9×)
[23]	208	1.1	1.5	2.1	22.6	16.1	13.8	5.3	0	2	5.5	25.7 (1.0×)
[24]	213	1.7	2.4	3.5	35.7	12.0	14.0	4.7	9	12	7.7	58.9 (2.3×)
[25]	200	2.4	3.0	4.2	48.3	7.3	3.2	2.2	5	4	3.6	35.0 (1.4×)
[26]	200	3.0	3.9	5.0	66.5	6.9	3.3	2.0	3	2	2.8	34.4 (1.4×)
[27]	210	1.8	1.9	3.2	32.8	25.7	18.6	8.7	14	21	13.7	94.2 (3.7×)
[28]	270	1.8	2.7	3.7	30.4	10.1	8.2	3.0	0	4	3.4	28.3 (1.1×)
[29]	185	3.3	4.5	6.1	75.1	5.5	3.4	1.5	3.5	2	2.4	33.4 (1.3×)
[30]	311	2.5	3.1	4.6	34.7	8.1	6.1	2.5	5.5	4	4.0	40.8 (1.6×)
This work	**169**	**1.8**	**2.3**	**3.6**	**45.5**	**7.6**	**5.7**	**2.3**	**3**	**4**	**3.3**	**25.4 (1.0×)**
ML-KEM-768, Security level 3 (Kyber768)												
[18]	161	6.3	7.9	10.0	149.1	7.4	4.6	2.1	3	2	2.9	70.2 (1.7×)
[19]	200	2.6	3.3	4.8	53.6	11.8	10.4	4.0	14	12	8.0	85.1 (2.1×)
[20]	115	7.0	10.0	14.0	269.6	16.0	6.0	4.0	9	16	7.4	229.4 (5.6×)
[21]	220	2.7	3.9	5.0	52.9	10.4	9.5	3.8	8.5	6	6.1	70.8 (1.7×)
[23]	208	1.7	2.4	3.0	34.1	16.8	13.7	5.6	0	2	5.8	41.2 (1.0×)
[24]	210	2.1	3.4	4.5	47.6	14.0	17.0	5.6	13	18	10.0	100.3 (2.5×)
[25]	200	4.2	5.0	7.0	81.3	7.3	3.2	2.2	5	4	3.6	59.0 (1.5×)
[26]	200	5.1	6.0	7.9	96.5	6.9	3.3	2.0	3	2	2.8	54.8 (1.4×)
[27]	210	2.6	2.7	4.1	44.7	25.7	18.6	8.7	14	21	13.7	128.8 (3.2×)
[28]	270	3.3	4.2	5.4	47.7	10.8	8.3	3.1	0	4	3.5	45.9 (1.1×)
[29]	185	5.6	7.1	9.2	118.4	5.5	3.4	1.5	3.5	2	2.4	52.6 (1.3×)
[30]	311	4.1	5.6	6.9	34.7	8.1	6.1	2.5	5.5	4	4.0	66.4 (1.6×)
This work	**169**	**3.1**	**3.6**	**5.6**	**72.8**	**7.6**	**5.7**	**2.3**	**3**	**4**	**3.3**	**40.6 (1.0×)**
ML-KEM-1024, Security level 5 (Kyber1024)												
[18]	161	9.4	11.3	13.9	212.3	7.4	4.6	2.1	3	2	2.9	100.3 (1.8×)
[19]	185	3.5	4.1	6.3	74.8	13.3	11.6	4.6	16	16	9.4	130.6 (2.3×)
[20]	112	10.0	14.0	18.0	365.2	16.0	6.0	5.0	12	17	9.1	382.2 (6.7×)
[21]	220	3.6	4.8	6.0	65.2	11.5	10.6	4.2	10.5	8	7.1	102.6 (1.8×)
[23]	208	2.7	3.4	4.1	49.0	17.8	14.0	6.0	0	2	6.2	63.5 (1.1×)
[24]	204	3.9	4.5	5.6	68.6	16.0	19.0	6.4	15	24	11.8	164.8 (2.9×)
[25]	200	6.8	8.0	10.2	126.1	7.3	3.2	2.2	5	4	3.6	90.0 (1.6×)
[26]	200	8.01	9.03	11.2	142.6	6.9	3.3	2.0	3	2	2.8	80.2 (1.4×)
[27]	210	3.4	3.5	5.8	60.5	25.7	18.6	8.7	14	21	13.7	174.0 (3.1×)
[28]	263	5.1	6.0	7.6	71.9	11.6	8.9	3.4	0	4	3.8	71.4 (1.3×)
[29]	185	8.5	10.1	12.9	170.3	5.5	3.4	1.5	3.5	2	2.4	75.6 (1.3×)
[30]	311	6.3	8.2	9.9	34.7	8.1	6.1	2.5	5.5	4	4.0	97.6 (1.7×)
This work	**169**	**4.5**	**5.1**	**7.6**	**101.8**	**7.6**	**5.7**	**2.3**	**3**	**4**	**3.3**	**56.7 (1.0×)**

[1] FPGA platform recommended by NIST, Artix-7 device.

[2] The Equivalent Number of Slices ENS = Slices/$(\frac{LUTs}{4} + \frac{FFs}{8})$ + BRAMs×200 + DSPs×100.

[3] The Area-Time Product (ATP) is calculated as Total Clock Cycles (Keygen + Encapsulation + Decapsulation) × ENS.

operations for computational modules, improving operating frequency and reducing hardware costs. Nonetheless, their SHA3 module is as slow as in [29], degrading overall execution performance. Compared to these works, our ML-KEM design achieves the lowest cost while maintaining comparable execution time. Therefore, our ATP outperforms those of [19], [20], [21], [24], [25], [28], and [30] by up to 2.3×, 6.7×, 1.9×, 2.9×, 1.6×, 1.3×, and 1.7× across all three security levels.

Moreover, [23], [27] propose designs targeting low-latency applications. In [23], Ni et al. introduce a multi-path delay commutator NTT accelerator for CRYSTALS-Kyber. The authors also utilize FIFOs for data storage, similar to [28]. Besides, 12 × 12-bit multiplications are performed using LUTs instead of DSPs. Therefore, they report 0 BRAM usage and low ENS. Our design has ATP comparable to [23]. Reference [27] proposes a memory-based pipelined NTT design for ML-KEM, supporting all three security levels. However, this design has low hardware efficiency. As a result, our ML-KEM performs ATP improvements of 3.7×, 3.2×, and 3.1× at ML-KEM-512, ML-KEM-768, and ML-KEM-1024, compared to [27].

Table 3 compares our ASIC results with those from references [11], [12], [13], [14], [20], and [22]. To the best of our knowledge, our work is the first ASIC implementation for the ML-KEM standard in the literature. We report the ATP trade-off as kGE × Total Clock Cycles (CCs), where GE represents the NAND gate equivalent, to ensure meaningful comparison across different CMOS technologies.

References [11], [12], and [13] implement HW/SW co-designs for CRYSTALS-Kyber based on open-source RISC-V processors. [11] designs a single hardware core supporting multiple operations, while [12] designs separate hardware cores for each operation. Moreover, [13] implements CRYSTALS-Kyber as a co-processor. Therefore, [13] has the most hardware logic cost. Our ASIC achieves superior

TABLE 3. ASIC implementation results and comparison with previous works.

Work	Process (nm)	Method	Area		Freq. (MHz)	Cycle Count			Total Time μs	Total Power (mW)	Total Energy (μJ)	ATP[1]
			Logic gates (kGE)	SRAM (kB)		Keygen. (kCCs)	Encap. (kCCs)	Decap. (kCCs)				
ML-KEM-512, Security level 1 (Kyber512)												
[11]	40	HW/SW	106	40.25	72	74.5	131.7	142.9	4840	5.5	26.59	37 (30.1×)
[12]	65	HW/SW	170	465	45	150.1	193.1	204.8	12177	2.6	140.87	93 (75.6×)
[13]	28	HW/SW	979	12	300	18.5	45.9	80.0	481	32.12	12.8	141 (115×)
[20]	65	HW	95	80	200	4.0	7.0	10.0	105	N/A	N/A	2.0 (1.6×)
[14]	28	HW	623	36.75	540	1.2	2.3	2.5	8.9	N/A	N/A	3.7 (3.0×)
This work	**180**	**HW**	**160**	**7.37**	**93**	**1.8**	**2.3**	**3.6**	**82.8**	**77.6**	**6.4**	**1.2 (1.0×)**
ML-KEM-768, Security level 3 (Kyber768)												
[11]	40	HW/SW	106	40.25	72	111.5	177.5	190.6	6661	5.4	35.51	51 (25.9×)
[12]	65	HW/SW	170	465	45	273.4	325.9	340.4	20889	2.6	243.46	160 (81.3×)
[20]	65	HW	93	175	200	7.0	10.0	14.0	155	N/A	N/A	2.9 (1.5×)
[14]	28	HW	623	36.75	540	1.8	3.2	3.5	12.3	N/A	N/A	5.3 (2.7×)
This work	**180**	**HW**	**160**	**7.37**	**93**	**3.1**	**3.6**	**5.6**	**132.3**	**77.6**	**10.3**	**2.0 (1.0×)**
ML-KEM-1024, Security level 5 (Kyber1024)												
[11]	40	HW/SW	106	40.25	72	148.5	223.5	241.0	8514	5.7	48.33	65 (23.6×)
[12]	65	HW/SW	170	465	45	349.7	405.5	424.7	26298	2.6	307.68	201 (73.0×)
[13]	28	HW/SW	979	12	300	39.7	81.5	136.5	859	32.12	24.26	252 (91.6×)
[20]	65	HW	104	190	200	10.0	14.0	18.0	210	N/A	N/A	4.4 (1.6×)
[14]	28	HW	623	36.75	540	2.7	4.6	4.9	17.7	N/A	N/A	7.6 (2.7×)
[22]	65	HW	769	34.82	280	9.1	11.4	13.9	127.2	N/A	55.1	26.4 (9.6×)
This work	**180**	**HW**	**160**	**7.37**	**93**	**4.5**	**5.1**	**7.6**	**184.9**	**77.6**	**14.4**	**2.8 (1.0×)**

[1] The Area-Time Product (ATP) is calculated as Total Clock Cycles (Keygen + Encapsulation + Decapsulation) × kGE.

performance, with ATP better of up to 30.1×, 81.3×, and 115.0× compared to [11], [12], and [13], respectively.

References [14], [20], and [22] report ASIC synthesis results for their hardware implementations of CRYSTALS-Kyber. Reference [20] has the lowest logic hardware cost. However, this implementation requires considerable SRAM and has the slowest execution speed. References [14] and [22] achieve faster execution than our implementation, but require significantly more hardware. As a result, our ATP demonstrates better efficiency, with improvements ranging from 1.5× to 9.6× compared to [14], [20], and [22].

Additionally, our ASIC needs significantly less SRAM than other ASIC implementations. The measured results indicate that our ASIC has the lowest energy consumption, with 6.4 μJ, 10.3 μJ, and 14.4 μJ for ML-KEM-512, ML-KEM-768, and ML-KEM-1024, respectively.

VI. CONCLUSION AND FUTURE WORK

Recognizing the critical need for quantum-resistant key establishment, this paper presented a hardware design that achieves significant area-time efficiency for the ML-KEM standard. Key to our approach was a novel SHA3 architecture that tightly integrated hashing and sampling, thereby eliminating the need for output storage while maintaining throughput. We also introduced a high-speed NTT accelerator with low memory usage for twiddle factors. Furthermore, our data memory organization is highly optimized and saves, thanks to leveraging partitioned memories and overwriting intermediate data. Operation scheduling is tightly coupled with parallel and pipelining processing techniques to optimize memory access and reduce execution time.

The resulting ML-KEM design supports all three NIST security levels and exhibits compelling performance. Our FPGA implementation demonstrated an area-time product up to 6.7× better than comparable designs. Furthermore, our 180 nm CMOS ASIC implementation achieved even greater ATP improvements (1.5× to 115.0×) while boasting the lowest SRAM usage (7.37 kB) and minimal energy consumption (6.4 μJ, 10.3 μJ, and 14.4 μJ for the respective security levels).

The contributions in this paper can be effectively applied to other PQC standards that have a common mathematical structure (e.g., FIPS 204/CRYSTALS-Dilithium [49]) or share computational requirements (e.g., FALCON [50] and FIPS 205/SPHINCS+ [51]). Besides that, SCA is an essential part of PQC implementation, and our design is resistant to timing attacks. However, since SCA is not the primary focus of this work, we plan to explore this field in future research. Therefore, our long-term vision is to develop the above-proposed modules for multi-standard and equip them with SCA countermeasures. Finally, the HW/SW co-design solution will be explored by integrating these modules into an open-source RISC-V processor for deployment in secure end-user devices.

ACKNOWLEDGMENT

The VLSI chip in this study was fabricated in the chip fabrication program of VLSI Design and Education Center (VDEC), The University of Tokyo, with the collaboration of Rohm Corporation and Toppan Printing Corporation.

REFERENCES

[1] National Institute of Standards and Technology. (Aug. 2024). *NIST Releases First 3 Finalized Post-Quantum Encryption Standards*. [Online]. Available: https://www.nist.gov/news-events/news/2024/08/nist-releases-first-3-finalized-post-quantum-encryption-standards

[2] Federal Information Processing Standards Publication (FIPS), "Module-Lattice-Based key encapsulation mechanism standard," Dept. of Commerce, Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Tech. Rep. NIST FIPS 203, doi: 10.6028/NIST.FIPS.203.

[3] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehl. (Jan. 2021). *CRYSTALS-Kyber: Algorithm Specifications and Supporting Documentation*. [Online]. Available: https://pq-crystals.org/kyber/data/kyber-specification-round3-20210131.pdf

[4] W.-K. Lee and S. O. Hwang, "High throughput implementation of post-quantum key encapsulation and decapsulation on GPU for Internet of Things applications," *IEEE Trans. Services Comput.*, vol. 15, no. 6, pp. 3275–3288, Nov. 2022.

[5] J. Huang, H. Zhao, J. Zhang, W. Dai, L. Zhou, R. C. C. Cheung, Ç. K. Koç, and D. Chen, "Yet another improvement of plantard arithmetic for faster kyber on low-end 32-bit IoT devices," *IEEE Trans. Inf. Forensics Security*, vol. 19, pp. 3800–3813, 2024.

[6] D. C. Lawo, R. Frantz, A. C. Cano Aguilera, X. A. I. Clemente, M. P. Podles, J. L. Imaña, I. T. Monroy, and J. J. Vegas Olmos, "Falcon/Kyber and Dilithium/Kyber network stack on Nvidia's data processing unit platform," *IEEE Access*, vol. 12, pp. 38048–38056, 2024.

[7] X. Ji, J. Dong, T. Deng, P. Zhang, J. Hua, and F. Xiao, "HI-kyber: A novel high-performance implementation scheme of kyber based on GPU," *IEEE Trans. Parallel Distrib. Syst.*, vol. 35, no. 6, pp. 877–891, Jun. 2024.

[8] Y. Kim, S. Yoon, and S. C. Seo, "Vectorized implementation of kyber and dilithium on 32-bit Cortex–A series," *IEEE Access*, vol. 12, pp. 104414–104428, 2024.

[9] J. Zhang, Y. Yan, J. Huang, and Ç. K. Koç, "Optimized software implementation of Keccak, Kyber, and dilithium on RV{32,64}IM{B}{V}," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2025, no. 1, pp. 632–655, Dec. 2024.

[10] X. Ji, J. Dong, J. Huang, Z. Yuan, W. Dai, F. Xiao, and J. Lin, "ECO-CRYSTALS: Efficient cryptography CRYSTALS on standard RISC-V ISA," *IEEE Trans. Comput.*, vol. 74, no. 2, pp. 401–413, Feb. 2025.

[11] U. Banerjee, T. S. Ukyab, and A. P. Chandrakasan, "Sapphire: A configurable crypto-processor for post-quantum lattice-based protocols," in *Proc. IACR Trans. Crypto. Hardw. Embedded Syst.*, Aug. 2019, pp. 17–61.

[12] T. Fritzmann, G. Sigl, and J. Sepúlveda, "RISQ-V: Tightly coupled RISC-V accelerators for post-quantum cryptography," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2020, no. 4, pp. 239–280, Aug. 2020.

[13] G. Xin, J. Han, T. Yin, Y. Zhou, J. Yang, X. Cheng, and X. Zeng, "VPQC: A domain-specific vector processor for post-quantum cryptography based on RISC-V architecture," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 8, pp. 2672–2684, Aug. 2020.

[14] Y. Zhao, R. Xie, G. Xin, and J. Han, "A high-performance domain-specific processor with matrix extension of RISC-V for module-LWE applications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 7, pp. 2871–2884, Jul. 2022.

[15] Z. Ye, R. Song, H. Zhang, D. Chen, R. C.-C. Cheung, and K. Huang, "A highly-efficient lattice-based post-quantum cryptography processor for IoT applications," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2024, no. 2, pp. 130–153, Mar. 2024.

[16] T. Wang, C. Zhang, X. Zhang, D. Gu, and P. Cao, "Optimized hardware–software co-design for kyber and dilithium on RISC-V SoC FPGA," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2024, no. 3, pp. 99–135, Jul. 2024.

[17] P. Ravi, A. Chattopadhyay, J. P. D'Anvers, and A. Baksi, "Side-channel and fault-injection attacks over lattice-based post-quantum schemes (kyber, Dilithium): Survey and new results," *ACM Trans. Embedded Comput. Syst.*, vol. 23, no. 2, pp. 1–54, Mar. 2024.

[18] Y. Xing and S. Li, "A compact hardware implementation of CCA-secure key exchange mechanism CRYSTALS-KYBER on FPGA," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2021, no. 2, pp. 328–356, Feb. 2021.

[19] M. Bisheh-Niasar, R. Azarderakhsh, and M. Mozaffari-Kermani, "High-speed NTT-based polynomial multiplication accelerator for post-quantum cryptography," in *Proc. IEEE 28th Symp. Comput. Arithmetic (ARITH)*, Jun. 2021, pp. 94–101.

[20] M. Bisheh-Niasar, R. Azarderakhsh, and M. Mozaffari-Kermani, "Instruction-set accelerated implementation of CRYSTALS-kyber," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 11, pp. 4648–4659, Nov. 2021.

[21] V. B. Dang, K. Mohajerani, and K. Gaj, "High-speed hardware architectures and FPGA benchmarking of CRYSTALS-kyber, NTRU, and saber," *IEEE Trans. Comput.*, vol. 72, no. 2, pp. 306–320, Feb. 2023.

[22] A. Aikata, A. C. Mert, M. Imran, S. Pagliarini, and S. S. Roy, "KaLi: A crystal for post-quantum security using kyber and dilithium," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 70, no. 2, pp. 747–758, Feb. 2023.

[23] Z. Ni, A. Khalid, D. Kundi, M. O'Neill, and W. Liu, "HPKA: A high-performance CRYSTALS-kyber accelerator exploring efficient pipelining," *IEEE Trans. Comput.*, vol. 72, no. 12, pp. 3340–3353, Dec. 2023.

[24] G. Li, D. Chen, G. Mao, W. Dai, A. I. Sanka, and R. C. C. Cheung, "Algorithm-hardware co-design of split-radix discrete Galois transformation for KyberKEM," *IEEE Trans. Emerg. Topics Comput.*, vol. 11, no. 4, pp. 824–838, Oct. 2023.

[25] W. Guo and S. Li, "Highly-efficient hardware architecture for CRYSTALS-kyber with a novel conflict-free memory access pattern," in *Proc. IEEE Trans. Circ. Syst. I, Reg. Papers*, Aug. 2023, vol. 70, no. 11, pp. 4505–4515.

[26] W. Guo and S. Li, "Split-radix based compact hardware architecture for CRYSTALS-kyber," *IEEE Trans. Comput.*, vol. 73, no. 1, pp. 97–108, Jan. 2024.

[27] H. Kim, H. Jung, A. Satriawan, and H. Lee, "A configurable ML-KEM/Kyber key-encapsulation hardware accelerator architecture," in *Proc. IEEE Trans. Circ. Syst. II, Express Briefs*, Aug. 2024, vol. 71, no. 11, pp. 4678–4682.

[28] Z. Ni, A. Khalid, W. Liu, and M. O'Neill, "A highly hardware efficient ML-KEM accelerator with optimised architectural layers," *ACM Trans. Embedded Comput. Syst.*, vol. 24, no. 2, pp. 1–24, Mar. 2025.

[29] T.-H. Nguyen, D.-T. Dam, P.-P. Duong, B. Kieu-Do-Nguyen, C.-K. Pham, and T.-T. Hoang, "Efficient hardware implementation of the lightweight CRYSTALS-kyber," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 72, no. 2, pp. 610–622, Feb. 2025.

[30] Y. Cui, J. Chen, Z. Ni, Z. Zhang, C. Wang, and W. Liu, "Instruction-based high-performance hardware controller of CRYSTALS-kyber with balanced resource utilization," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 72, no. 5, pp. 2394–2407, May 2025.

[31] S. Khan, A. Khalid, C. Rafferty, Y. A. Shah, M. O'Neill, W.-K. Lee, and S. O. Hwang, "Efficient, error-resistant NTT architectures for CRYSTALS-kyber FPGA accelerators," in *Proc. IFIP/IEEE 31st Int. Conf. Very Large Scale Integr. (VLSI-SoC)*, Oct. 2023, pp. 1–6.

[32] A. Sarker, A. C. Canto, M. Mozaffari Kermani, and R. Azarderakhsh, "Error detection architectures for Hardware/Software co-design approaches of number-theoretic transform," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 7, pp. 2418–2422, Jul. 2023.

[33] Y. Zhao, S. Pan, H. Ma, Y. Gao, X. Song, J. He, and Y. Jin, "Side channel security oriented evaluation and protection on hardware implementations of kyber," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 70, no. 12, pp. 5025–5035, Dec. 2023.

[34] D. Xu, K. Wang, and J. Tian, "A hardware-friendly shuffling countermeasure against side-channel attacks for kyber," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 72, no. 3, pp. 504–508, Mar. 2025.

[35] National Institute of Standards and Technology (NIST). (Aug. 2015). *SHA-3 Standard: Permutationbased Hash and Extendable-output Functions*. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf

[36] K. Javeed, S. Rubab, and D. Gregg, "Efficient reconfigurable modular multipliers for post-quantum digital signatures," in *Proc. 31st IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Nov. 2024, pp. 1–4.

[37] T.-H. Nguyen, C.-K. Pham, and T.-T. Hoang, "A high-efficiency modular multiplication digital signal processing for lattice-based post-quantum cryptography," *Cryptography*, vol. 7, no. 4, p. 46, Sep. 2023.

[38] K. Javeed and D. Gregg, "Efficient number theoretic transform architecture for CRYSTALS-kyber," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 72, no. 1, pp. 263–267, Jan. 2025.

[39] G. Alagic, D. Apon, D. Cooper, Q. Dang, T. Dang, J. Kelsey, J. Lichtinger, Y.-K. Liu, C. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, and D. S.-Tone. (Jul. 2022). *Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process*. [Online]. Available: https://www.nist.gov/publications/status-report-third-round-nist-post-quantum-cryptography-standardization-process

[40] A. Langlois and D. Stehlé, "Worst-case to average-case reductions for module lattices," *Designs, Codes Cryptography*, vol. 75, no. 3, pp. 565–599, Jun. 2015.

[41] D. Hofheinz, K. Hövelmanns, and E. Kiltz, "A modular analysis of the fujisaki-okamoto transformation," in *Proc. Theory Crypto. Conf.*, Jan. 2017, pp. 341–371.

[42] V. Lyubashevsky, D. Micciancio, C. Peikert, and A. Rosen, "SWIFFT: A modest proposal for FFT hashing," *Fast Softw. Encryp. (FSE)*, vol. 5086, pp. 54–72, Jul. 2008.

[43] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, no. 90, pp. 297–301, Apr. 1965.

[44] W. M. Gentleman and G. Sande, "Fast Fourier transforms: For fun and profit," in *Proc. Fall Joint Comput. Conf. (AFIPS)*, Nov. 1966, pp. 563–578.
[45] S. S. Roy, F. Vercauteren, N. Mentens, D. D. Chen, and I. Verbauwhede, "Compact ring-LWE cryptoprocessor," *Crypto. Hardw. Embedded Syst. (CHES)*, vol. 8731, pp. 371–391, Jan. 2014.
[46] T. Pöppelmann, T. Oder, and T. Güneysu, "High-performance ideal lattice-based cryptography on 8-bit ATxmega microcontrollers," in *Proc. Int. Conf. Crypto. Info. Secu. Latin Amer. (LATINCRYPT)*, Jan. 2015, pp. 346–365.
[47] M. Bisheh-Niasar, R. Azarderakhsh, and M. M. Kermani, "A monolithic hardware implementation of kyber: Comparing apples to apples in PQC candidates," in *Proc. Int. Conf. Crypto. Info. Secu. Latin Amer. (LATINCRYPT)*, Jan. 2021, pp. 108–126.
[48] M. Knezevic, F. Vercauteren, and I. Verbauwhede, "Faster interleaved modular multiplication based on Barrett and Montgomery reduction methods," *IEEE Trans. Comput.*, vol. 59, no. 12, pp. 1715–1721, Dec. 2010.
[49] *Module-Lattice-Based Digital Signature Standard*, Standard NIST FIPS 204, National Institute of Standards and Technology (NIST), Washington, DC, USA, doi: 10.6028/NIST.FIPS.204.
[50] P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang. (2020). *Falcon: Fast-Fourier Lattice-based Compact Signatures Over NTRU (v1.2)*. [Online]. Available: https://falcon-sign.info/falcon.pdf
[51] *Stateless Hash-Based Digital Signature Standard*, Standard NIST FIPS 205, National Institute of Standards and Technology (NIST), Aug. 2024, doi: 10.6028/NIST.FIPS.205.

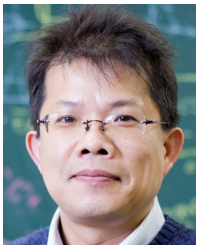
TRONG-HUNG NGUYEN (Graduate Student Member, IEEE) received the B.Sc. degree in control engineering and automation from Hanoi University of Science and Technology (HUST), Hanoi, Vietnam, in 2014, and the M.S. degree in electronic engineering from The University of Electro-Communications (UEC), Tokyo, Japan, in 2022. He is currently pursuing the Ph.D. degree in information and network engineering with UEC. His research interests include digital signal processing, hardware accelerators, and post-quantum cybersecurity.

TUAN-KIET DANG (Graduate Student Member, IEEE) received the B.Eng. degree in embedded systems from Da Nang University of Science and Technology, Da Nang, Vietnam, and the M.Sc. degree in computer and network engineering from the University of Electro-Communication (UEC), Tokyo, Japan. He is currently pursuing the Ph.D. degree with UEC. Since 2023, he has been a Research Assistant with UEC. His research interests include computer architecture, SoC design, high performance digital circuit design, and hardware security.

DUC-THUAN DAM (Graduate Student Member, IEEE) received the B.Sc. degree in electrical and electronic engineering and the M.S. degree in electronic engineering from Le Quy Don Technical University, Hanoi, Vietnam, in 2013 and 2019, respectively. He is currently pursuing the Ph.D. degree in information and network engineering with The University of Electro-Communications (UEC), Tokyo, Japan. His research interests include forward error correction codes, post-quantum cryptography, hardware implementation, and RISC-V.

KHAI-DUY NGUYEN (Graduate Student Member, IEEE) received the B.Sc. degree in electronics from the University of Science and Technology, The University of Da Nang, Da Nang, Vietnam, in 2020, and the M.S. degree in electronics from The University of Electro-Comunications (UEC), Tokyo, Japan, in 2024. He is currently pursuing the Ph.D. degree in information and network engineering with the University of Electro-Communications (UEC), Tokyo, Japan.

PHUC-PHAN DUONG (Graduate Student Member, IEEE) received the B.Sc. and M.S. degrees from the Department of Electronics and Telecommunications from the Posts and Telecommunications Institute of Technology (PTIT), Hanoi, Vietnam, in 2011 and 2014, respectively. He is currently pursuing the Ph.D. degree in information and network engineering with The University of Electro-Communications (UEC), Tokyo, Japan. From 2013 to 2023, he was a Lecturer Assistant with the Academy Of Cryptography Techniques. His research interests include cryptography, embedded systems design, and hardware security.



CONG-KHA PHAM (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electronics engineering from Sophia University, Tokyo, Japan. He is currently a Professor with the Department of Information and Network Engineering, The University of Electro-Communications (UEC), Tokyo, Japan, and is dedicated to teaching both undergraduate and postgraduate students. His research expertise lies in the design and implementation of hardware systems using FPGAs and integrated circuits. His current research portfolio includes projects on energy harvesting power solutions, low-power data-centric sensor network systems (including miniaturization and long-distance wireless communication), the development of super low-voltage devices, investigation of memory-based information detection systems, and the practical hardware implementation of systems using FPGAs, and integrated circuits.

TRONG-THUC HOANG (Member, IEEE) received the B.Sc. and M.S. degrees in electronic engineering from the Ho Chi Minh City University of Science (HCMUS), Ho Chi Minh City, Vietnam, in 2012 and 2017, respectively, and the Ph.D. degree in engineering from The University of Electro-Communications (UEC), Tokyo, Japan, in 2022. From 2012 to 2017, he was a Lecturer Assistant with HCMUS. From 2019 to 2020, he was a Research Assistant with UEC. From 2019 to 2022, he was a Research Assistant with the Cyber-Physical Security Research Center (CPSEC), National Institute of Advanced Industrial Science and Technology (AIST), Tokyo, Japan. Since April 2022, he has been an Assistant Professor with the Department of Computer and Network Engineering, UEC. His research interests include digital signal processing, computer architecture, cyber-security, and ultra-low power systems-on-a-chip.