An Open-Hardware ML-KEM Polynomial Ring Accelerator on Chipyard RISC-V SoC: System-Level Integration and Evaluation

Yi-Chang Tsai , Yu-Han Lin , Wen-Jyi Hwang *

Posted Date: 21 May 2026

doi: 10.20944/preprints202605.1405.v1

Keywords: ML-KEM; post-quantum cryptography; polynomial ring multiplication; open hardware; RISC-V SoC; chipyard; number theoretic transform; hardware accelerator; system-on-chip integration

 doi:10.20944/preprints202605.1405.v1

Article

An Open-Hardware ML-KEM Polynomial Ring Accelerator on Chipyard RISC-V SoC: System-Level Integration and Evaluation

Yi-Chang Tsai, Yu-Han Lin and Wen-Jyi Hwang *

Department of Computer Science and Information Engineering, National Taiwan Normal University, Taipei 106, Taiwan

* Correspondence: whwang@ntnu.edu.tw

Abstract

With the standardization of the Module-Lattice-Based Key Encapsulation Mechanism (ML-KEM) in NIST FIPS 203 (2024), efficient hardware support for polynomial ring operations has become critical for practical post-quantum cryptography deployment. The dominant computational workload of ML-KEM arises from matrix–vector multi-plications over polynomial rings, which involve repeated Number Theoretic Transform (NTT), pointwise multiplication, and modular addition operations. This work proposes an ML-KEM polynomial ring accelerator leveraging Open Intellectual Property (Open IP) and integrates it into an open hardware Chipyard RISC-V System-on-Chip (SoC) via a Memory-Mapped I/O (MMIO) interface. The design incorporates an NTT-based datapath with multiplier and adder arrays, and employs a scratchpad memory to enable intermediate data reuse and reduce memory access overhead. The proposed architecture is implemented on a Kintex-7 Field Programmable Gate Array (FPGA) platform and evaluated at both kernel and system levels. Experimental results show that the accelerator reduces matrix–vector multiplication latency to 7,372 cycles, achieving up to 40× speedup over a software baseline. At the SoC level, the complete ML-KEM implementation achieves performance improvements of 1.6× to 2.1× across different parameter sets. These results demonstrate that integrating Open IP within an open hardware SoC provides an effective and reproducible approach for accelerating ML-KEM.

Keywords: ML-KEM; post-quantum cryptography; polynomial ring multiplication; open hardware; RISC-V SoC; chipyard; number theoretic transform; hardware accelerator; system-on-chip integration

1. Introduction

With the rapid advancement of quantum computing technologies, conventional public-key cryptosystems based on integer factorization and discrete logarithm problems, such as RSA and elliptic curve cryptography, are theoretically vulnerable to quantum attacks enabled by algorithms such as Shor's algorithm, thereby undermining their security foundations [1]. As a result, the development and standardization of quantum-resistant cryptographic schemes, known as Post-Quantum Cryptography (PQC), have become critical research and transition directions in the fields of cryptography and information security [2]. To facilitate the practical deployment of PQC, the National Institute of Standards and Technology (NIST) officially released Federal Information Processing Standards (FIPS) 203 in 2024, standardizing the Module-Lattice-Based Key Encapsulation Mechanism (ML-KEM) [3]. Derived from the CRYSTALS-Kyber algorithm, ML-KEM is based on the hardness of module-lattice problems and relies heavily on polynomial ring arithmetic as its computational foundation. Due to the extensive use of polynomial operations, particularly matrix–

 doi:10.20944/preprints202605.1405.v1

vector polynomial multiplication, efficient ML-KEM implementations on embedded systems and System-on-Chip (SoC) platforms has become a key challenge in PQC hardware research [4–6].

In ML-KEM, polynomial multiplication is typically performed using the Number Theoretic Transform (NTT), which converts convolution operations into pointwise multiplications in the transform domain, thereby reducing computational complexity. Previous studies have identified polynomial multiplication and the associated NTT operations as major performance bottlenecks in ML-KEM implementations, leading to extensive research on hardware optimization of NTT and polynomial multiplication units [4–7]. For example, Yaman et al. proposed a dedicated NTT-based hardware architecture to accelerate polynomial multiplication in CRYSTALS-Kyber [7]. However, most of these works focus on the design and evaluation of individual computational modules. In practical implementations, the dominant workload of ML-KEM arises from matrix–vector multiplication over polynomial rings, which involves multiple NTTs, pointwise multiplications, and modular additions. Therefore, optimizing only standalone NTT or polynomial multiplication modules is often insufficient to fully evaluate system-level performance impacts [4,5]. Meanwhile, the emergence of the RISC-V open instruction set architecture and the open hardware ecosystem has enabled researchers to integrate domain-specific accelerators into reproducible SoC platforms and evaluate their performance at the system level. Recent works have explored integrating PQC accelerators into RISC-V-based systems, including hardware–software co-design implementations of Kyber and Dilithium on Field Programmable Gate Array (FPGA) based SoCs [8]. Among these platforms, Chipyard has become a widely adopted open-source SoC design framework, providing a generator-based methodology to construct complete RISC-V systems with Rocket cores, on-chip interconnects, and memory hierarchies [9]. Such open hardware platforms not only facilitate the development of cryptographic accelerators but also enable system-level validation within full processor and operating system environments, thereby improving the reproducibility and practicality of PQC hardware research.

Based on this background, this work proposes an ML-KEM polynomial ring accelerator leveraging Open Intellectual Property (Open IP) and integrates it into a RISC-V SoC built using the Chipyard framework. The proposed design constructs a complete Rocket Core-based system and incorporates an open-source NTT hardware module proposed by Yaman et al. [7] to accelerate matrix–vector polynomial computations over polynomial rings. Unlike prior works that primarily focus on standalone NTT or polynomial multiplication modules, the proposed accelerator is integrated into the SoC via a Memory-Mapped I/O (MMIO) interface, allowing it to operate within a full processor and operating system environment and enabling system-level hardware validation.

The main contributions of the proposed work are summarized as follows:

1. An ML-KEM polynomial ring accelerator based on Open IP and open hardware;
2. Integration into a Chipyard-based RISC-V SoC via an MMIO interface;
3. System-level implementation and evaluation on an FPGA-based platform;
4. Validation of open hardware platforms for reproducible PQC system research.

2. Background and Related Works

This section reviews the background and related work relevant to ML-KEM hardware acceleration. It first introduces the fundamental polynomial arithmetic in ML-KEM, with a focus on NTT-based computation. Next, prior studies on ML-KEM and Kyber hardware accelerators are surveyed, including both arithmetic-level optimizations and system-level integration approaches. Finally, representative open hardware platforms and RISC-V-based SoC frameworks are discussed to highlight their roles in enabling reproducible and scalable PQC hardware research. Through this review, the limitations of existing approaches are identified, motivating the proposed Open IP-based ML-KEM accelerator with full system-level integration on an open hardware SoC platform.

2.1. ML-KEM and NTT-based Polynomial Arithmetic

 doi:10.20944/preprints202605.1405.v1

ML-KEM is a PQC standard defined in FIPS 203 by NIST in 2024 [3]. The scheme is derived from the CRYSTALS-Kyber cryptosystem, and its security relies on the hardness of the module Learning-With-Errors (module-LWE) problem [5]. In ML-KEM, most computations are performed over a polynomial ring defined as

$$R_q=Z_q[x] / (x^n+1), \tag{1}$$

where the Kyber parameters are typically n=256 and q=3329.

During the Key Generation (KeyGen), Encapsulation (Encaps), and Decapsulation (Decaps) procedures, a large number of matrix–vector polynomial multiplications are required. These operations involve repeated polynomial multiplications and modular additions, making them the dominant computational workload in ML-KEM implementations [4,5]. The matrix–vector multiplication over polynomial rings can be expressed as

$$\begin{aligned} C = A^T \odot B &= [a_1(x) \quad \cdots \quad a_k(x)] \odot \begin{bmatrix} b_1(x) \\ \vdots \\ b_k(x) \end{bmatrix}, \\ &= a_1(x) \odot b_1(x) \oplus \ldots \oplus a_k(x) \odot b_k(x), \end{aligned} \tag{2}$$

where A and B are $k \times 1$ polynomial vectors in R_q, and each polynomial $a_i(x), b_i(x) \in R_q$ has degree less than n. The operator $\odot$ denotes pointwise multiplication, and $\oplus$ denotes modular addition.

To reduce the computational complexity of polynomial multiplication, ML-KEM employs the NTT, as illustrated in Figure 1. The NTT converts polynomials from the coefficient domain to the transform domain, allowing polynomial multiplication to be performed as element-wise operations. Following the computation flow shown in Figure 1, each pair of polynomials is first transformed into the NTT domain, and the matrix–vector polynomial multiplication is then computed as

$$C(\mathrm{x}) = \mathrm{INTT}\left(\mathrm{NTT}\left(a_1(x)\right) \odot \mathrm{NTT}\left(b_1(x)\right) \oplus \cdots \oplus \mathrm{NTT}\left(a_k(x)\right) \odot \mathrm{NTT}\left(b_k(x)\right)\right). \tag{3}$$

As shown in Figure 1, this computation flow consists of multiple NTT operations, followed by pointwise multiplications and modular accumulation, and finally an Inverse NTT (INTT) operation. Due to the repeated execution of these operations, NTT-based polynomial arithmetic constitutes the primary performance bottleneck in Kyber/ML-KEM implementations [4–6]. Therefore, efficient hardware acceleration of these operations is critical for improving the overall system performance of ML-KEM, particularly in embedded and SoC-based environments.

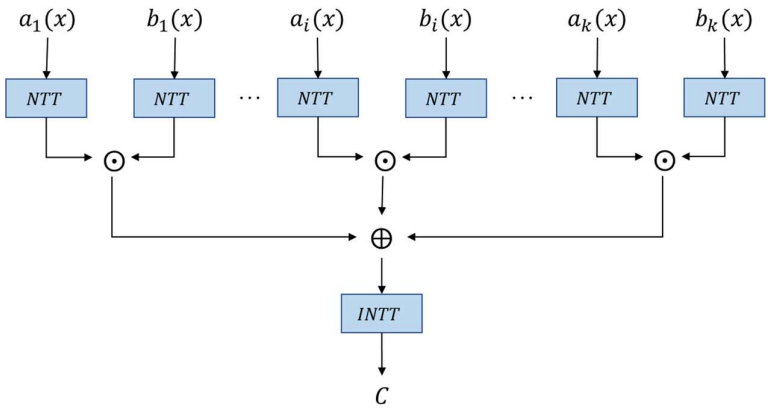

Figure 1. Matrix–vector multiplication over polynomial rings.

 doi:10.20944/preprints202605.1405.v1

2.2. ML-KEM Hardware Accelerators Survey

With the advancement of NIST PQC standardization, an increasing number of studies have investigated hardware acceleration techniques for ML-KEM and Kyber. Early works primarily focused on the design of efficient NTT architectures and polynomial multiplication units. For example, Karabulut et al. [26] proposed a RISC-V Instruction Set Architecture (ISA) extension for NTT acceleration, while Waris et al. [6], Huang et al. [22], Bisheh-Niasar et al. [24], and Zhang et al. [25] proposed high-speed NTT-based or polynomial-multiplication-oriented hardware architectures to accelerate Kyber and related post-quantum cryptographic computations. Liu et al. [12] further introduced a configurable NTT/INTT architecture that mitigates memory access conflicts, while Yaman et al. [7] developed an NTT-based polynomial multiplication accelerator specifically tailored for CRYSTALS-Kyber. These studies demonstrate that optimizing NTT and polynomial arithmetic is critical for improving the performance of ML-KEM.

Beyond standalone arithmetic modules, some studies have explored more complete Kyber-oriented hardware architectures. For example, Chen et al. [23] proposed an FPGA-based processor architecture for vectors of polynomials in Kyber, bridging the gap between low-level NTT acceleration and higher-level algorithm support. More recent research has further evolved toward full ML-KEM accelerator architectures. Kim et al. [13] proposed a configurable hardware accelerator capable of supporting multiple security levels of ML-KEM/Kyber. Ni et al. [14] presented a hardware-efficient ML-KEM design that reduces area consumption while maintaining high throughput. In addition, Cui et al. [15] introduced an instruction-based hardware controller for Kyber, enabling closer interaction between the processor and the accelerator. These works extend beyond individual arithmetic units and aim to support larger portions of the ML-KEM algorithm.

In parallel, several studies have explored system-level integration of PQC accelerators. Wang et al. integrated a Kyber accelerator into an FPGA-based RISC-V SoC platform using a hardware–software co-design approach [8]. Dolmeta et al. investigated memory-mapped NTT/INTT accelerators within RISC-V systems [16]. Furthermore, Dam et al. demonstrated the integration of NTT/INTT accelerators in the Chipyard RISC-V SoC framework [19,20], while Abdulrahman et al. implemented NTT acceleration on the OpenTitan platform using the Open-Titan Big Number (OTBN) extension [17]. These studies highlight that, in practical deployments, accelerator performance is influenced not only by arithmetic efficiency but also by system-level factors such as bus architecture, memory access patterns, and hardware–software interaction. Table 1 summarizes recent PQC hardware accelerator designs. Most prior works focus on accelerating specific computational modules, such as NTT and polynomial multiplication [6,7,12], or cryptographic primitives like Keccak [27]. Some designs extend to full ML-KEM acceleration [13,14], while others explore integration into RISC-V systems through hardware–software co-design or memory-mapped interfaces [8,16]. More recently, open-hardware-based implementations have emerged, including accelerators developed on platforms such as OpenTitan and RISC-V SoCs [17,19,20]. However, existing works are still largely limited to either partial algorithm acceleration or isolated hardware components, with relatively few studies addressing full ML-KEM system integration and validation on open hardware platforms.

Table 1. ML-KEM hardware accelerator survey and comparison of recent designs.

Work	Year	Algorithm	Architecture	Integration	Impl.	Open IP
Chen et al. [23]	2020	Kyber	Polynomial vector processor	Standalone accelerator	FPGA	No
Huang et al. [22]	2020	Kyber	NTT-based polynomial multiplication	Standalone accelerator	FPGA	No
Karabulut et al. [26]	2020	NTT	RISC-V ISA extension for NTT	CPU-integrated (ISA extension)	ASIC	No

 doi:10.20944/preprints202605.1405.v1

Waris et al. [6]	2021	Kyber	NTT/INTT-based polynomial multiplier	Standalone accelerator	FPGA	No
Yaman et al. [7]	2021	Kyber	Polynomial multiplication accelerator	Standalone accelerator	FPGA	Yes
Bisheh-Niasar et al. [24]	2021	Kyber	NTT-based polynomial multiplier	Standalone accelerator	FPGA	No
Zhang et al. [25]	2021	Kyber	Efficient NTT architecture	Standalone accelerator	FPGA	No
Celik et al. [27]	2023	Kyber	Keccak hardware acceleration	RISC-V CPU (Ibex, MMIO/interrupt-based)	FPGA	No
Liu et al. [12]	2024	NTT/INTT	Configurable NTT/INTT accelerator	Standalone accelerator	ASIC	No
Kim et al. [13]	2024	ML-KEM	Configurable full KEM accelerator	Standalone full accelerator	ASIC	No
Wang et al. [8]	2024	Kyber / Dilithium	HW/SW co-design with polynomial accelerators	RISC-V SoC (HW/SW co-design)	FPGA	No
Dolmeta et al [16]	2024	Kyber	Memory-mapped NTT/INTT accelerator	RISC-V SoC (MMIO-based)	FPGA	No
Dam et al. (ICDV) [19]	2024	Kyber	NTT black-box accelerator	Chipyard RISC-V SoC (MMIO / peripheral)	FPGA/ASIC	Partial
Ni et al. [14]	2025	ML-KEM	Full ML-KEM accelerator	Standalone full accelerator	ASIC	No
Cui et al. [15]	2025	Kyber	Instruction-based hardware controller	CPU–accelerator (instruction-based)	ASIC	No
Abdulrahman et al. [17]	2025	ML-KEM / ML-DSA	OpenTitan OTBN extension	OpenTitan SoC (OTBN-based)	ASIC	Partial
Dam et al. (Electronics) [20]	2026	ML-KEM	Tightly integrated NTT accelerator with custom instructions	Chipyard RISC-V SoC (RoCC tightly-coupled)	ASIC	Partial
Proposed Work	2026	ML-KEM	ML-KEM Polynomial ring accelerator	Chipyard RISC-V SoC (MMIO / peripheral)	FPGA	Yes

From the perspective of hardware design methodology, the role of Open IP has become increasingly important. Open IP improves design reusability and development efficiency, and enables reproducible research environments, which are particularly valuable for PQC standardization and deployment. Nevertheless, as shown in Table 1, most existing ML-KEM and Kyber hardware accelerators remain closed or proprietary, with only a few works providing reusable open-source hardware modules. For instance, Yaman et al. [7] provide reusable polynomial

 doi:10.20944/preprints202605.1405.v1

multiplication components, while some SoC-based studies (e.g., [17,19,20]) are built on open hardware platforms but only partially release their accelerator designs. A closer examination reveals two key limitations in current Open IP-based PQC hardware designs. First, most open-source implementations focus on individual computational blocks (e.g., NTT or Keccak), lacking integrated IPs that support the complete ML-KEM algorithm flow. Second, even when integrated into open-hardware SoC platforms, the modularity and reusability of these designs are often limited, making it difficult to establish a scalable PQC IP ecosystem. As a result, there is still a lack of design approaches that simultaneously achieve full algorithm support, system-level integration, and reusable Open IP.

2.3. Open Hardware and RISC-V SoC Platforms

In recent years, open hardware platforms have become an essential foundation for SoC architecture research and validation, enabling researchers to design and evaluate hardware accelerators in reproducible and scalable environments. With the rapid development of the RISC-V open ISA, several open hardware platforms have been widely adopted in cryptographic and PQC research, including Chipyard, Ibex, OpenTitan, and OpenHW CORE-V.

Ibex is a lightweight RISC-V processor core designed for embedded control and security-oriented applications [10], and it is integrated into the OpenTitan project. OpenTitan is an open-source silicon root-of-trust platform that emphasizes security, verification, and reliability [11]. However, its system architecture is relatively fixed and tailored for security modules and control functions, making it less suitable for flexible SoC architecture exploration. Similarly, the OpenHW Group provides several open-source RISC-V cores, such as CV32E40P [18], along with comprehensive verification frameworks. Nevertheless, these efforts primarily focus on processor core development and verification, offering limited support for full SoC generation and accelerator integration.

In contrast, Chipyard provides a complete and modular SoC generation framework that enables rapid construction of customized RISC-V systems using a generator-based hardware design methodology [9]. The platform supports the integration of various processor cores (e.g., Rocket and BOOM), on-chip interconnects (TileLink), memory hierarchies, and custom hardware accelerators. It also offers a mature simulation and FPGA prototyping environment. Moreover, Chipyard-generated SoCs can support operating systems such as Linux, allowing hardware accelerators to be evaluated within a realistic software execution environment. This capability is particularly important for PQC systems, where the overall performance depends not only on the hardware accelerator itself but also on its interaction with the operating system, drivers, and application-level software. Compared with designs validated only at the Register-Transfer Level (RTL) or simulation level, SoC platforms with operating system support provide a more realistic evaluation of system-level behavior and performance.

Based on this comparison, the proposed work adopts Chipyard as the SoC platform for ML-KEM acceleration for several key reasons. First, it provides full SoC generation capabilities, enabling rapid construction of systems that include processors, memory subsystems, and interconnects. Second, it supports flexible accelerator integration mechanisms, such as MMIO and TileLink-based interfaces. Third, it offers a well-established verification flow, including both simulation and FPGA-based validation. Finally, it enables system-level evaluation with operating system support, improving the practicality and reproducibility of experimental results. As PQC hardware research evolves from arithmetic-level optimization to system-level design, the integration methodology between accelerators and processors has become a critical factor affecting overall performance. Existing approaches can be broadly categorized into three types: (1) loosely coupled architectures, where accelerators are integrated as peripherals or memory-mapped modules [16,19]; (2) tightly coupled architectures, implemented through instruction-set extensions or custom processor interfaces [20,26]; and (3) hardware–software co-design approaches that combine software control with hardware acceleration [8]. In addition, recent studies have explored PQC accelerator designs on

 doi:10.20944/preprints202605.1405.v1

open hardware platforms such as OpenTitan [17], although most of these efforts are still limited to specific computational modules or partial algorithm support.

To systematically compare prior work in terms of system integration level, algorithm completeness, and open hardware support, Table 2 summarizes representative PQC hardware accelerators at the system level. As shown in Table 2, although some designs achieve integration with RISC-V SoC platforms, they often lack either full ML-KEM algorithm support or comprehensive open hardware integration. In contrast, the proposed design in this work provides complete ML-KEM acceleration, integrates seamlessly into a RISC-V SoC, and is validated within an open hardware platform with operating system support. This approach addresses existing limitations in system-level evaluation and reproducibility in PQC hardware research.

From a system-level design perspective, open hardware not only influences the design methodology but also directly determines the deployability and verifiability of PQC accelerators in real-world systems. Compared with conventional closed hardware platforms, open hardware provides complete SoC architectures, standardized bus interfaces, and hardware–software co-design environments, enabling researchers to evaluate accelerator performance under conditions that closely resemble practical deployments. However, as summarized in Table 2, although many existing PQC hardware accelerator designs support integration with RISC-V SoC platforms, there remain significant differences in the level of open hardware support.

Table 2. Comparison of System-Level PQC Accelerator Integration.

Work	Algorithm	NTT Accelerator	Full ML-KEM	RISC-V SoC	Open Hardware
Wang et al. (TCHES) [8]	Kyber	✓	Partial	✓	✗
Dolmeta et al. [16]	Kyber	✓	✗	✓	✗
Abdulrahman et al. [17]	ML-KEM	✓	Partial	✓	Partial
Dam et al. (ICDV) [19]	Kyber	✓	✗	✓	Partial
Dam et al. (Electronics) [20]	ML-KEM	✓	Partial	✓	Partial
Proposed Work	ML-KEM	✓ (Integrated NTT datapath)	✓ (Full matrix–vector)	✓ (Chipyard SoC + OS support)	✓ (Open IP + reproducible)

Specifically, some studies, such as those by Wang et al. [8] and Dolmeta et al. [16], integrate accelerators into RISC-V SoC platforms, but their implementations are often based on specific experimental platforms or customized systems. As a result, they lack comprehensive support from fully open hardware frameworks, which limits their reproducibility and scalability. In contrast, more recent works have begun to adopt open hardware platforms such as OpenTitan and Chipyard. For example, Abdulrahman et al. [17] implemented partial ML-KEM support on OpenTitan using OTBN instruction extensions, while Dam et al. [19,20] integrated NTT accelerators into a Chipyard-based

 doi:10.20944/preprints202605.1405.v1

RISC-V SoC. Nevertheless, these studies primarily focus on accelerating NTT or partial algorithm flows, and still exhibit limitations in terms of full ML-KEM support and depth of system-level integration. Furthermore, their open hardware support is often partial, and does not yet form a fully reusable system architecture. As further observed from Table 2, existing works that simultaneously achieve (1) full ML-KEM support, (2) integration within a RISC-V SoC, and (3) complete open hardware reproducibility remain limited. Most prior designs satisfy only one or two of these criteria for example, providing SoC integration without full algorithm support, or offering partially open hardware designs without system-level validation. This observation indicates that current PQC hardware research still lacks a comprehensive design paradigm that enables full algorithm implementation and system-level validation on open hardware platforms.

Motivated by these limitations, the proposed work adopts open hardware as the core design foundation and leverages the generator-based SoC framework and complete software execution environment provided by Chipyard to realize system-level integration of an ML-KEM accelerator. By implementing the accelerator on an open RISC-V SoC platform and validating it within an operating system environment, the proposed design ensures both reproducibility and scalability, while enabling realistic performance evaluation under practical application scenarios. Compared with prior work, this study further emphasizes an integrated design paradigm of "Open Hardware + Full ML-KEM + System-Level Evaluation," addressing existing gaps in system-level validation and open hardware support in PQC accelerator research.

3. Polynomial ring multiplication accelerator for ML-KEM in RISC-V SoC

This section presents the proposed hardware accelerator architecture designed to implement matrix–vector multiplication over the polynomial ring R_q, and describes its integration and operation within a 64-bit RISC-V Rocket Core-based SoC platform. This computation constitutes the dominant and most computationally intensive operation in the ML-KEM algorithm. The primary design objective of the proposed work is to reduce redundant main memory accesses of polynomial coefficients across different stages of matrix–vector multiplication, including the NTT, pointwise multiplication, and modular addition. To achieve this goal, a polynomial ring accelerator architecture based on scratchpad memory [21] is proposed, in which all intermediate results are processed entirely within the accelerator. Specifically, transformation, multiplication, and accumulation operations are performed locally without external memory interaction. Through this data reuse mechanism, polynomial coefficients are transferred via I/O only at the beginning and end of the computation. This approach effectively eliminates repeated data movement across the system bus and significantly reduces system latency dominated by data transfer overhead.

3.1. System Architecture of the RISC-V SoC

The SoC architecture adopted in the proposed work is illustrated in Figure 2. The design is based on a RISC-V SoC platform, featuring a single 64-bit Rocket core that supports the RV64GC instruction set. The system operates at a clock frequency of 100 MHz. The processor is equipped with a 16 KB L1 instruction cache and a 16 KB L1 data cache, along with a shared 512 KB L2 cache. In terms of system interconnection, the processor core is connected to the L2 cache and other system components through the system bus, and accesses external DDR main memory via the memory bus. The hardware accelerator is integrated into the SoC through the periphery bus, enabling communication with the processor and memory subsystem. The baseline RISC-V SoC is constructed using the Chipyard framework, which provides a generator-based hardware design methodology for rapid system integration.

The proposed ML-KEM Polynomial Ring Accelerator (MPRA) is integrated into the SoC using the Chisel BlackBox interface provided by Chipyard. The accelerator is incorporated as an MMIO, allowing it to be accessed by the processor through standard load/store operations. In this integration model, the accelerator shares the main memory access channel with the processor, enabling efficient data exchange without requiring specialized instruction extensions.

 doi:10.20944/preprints202605.1405.v1

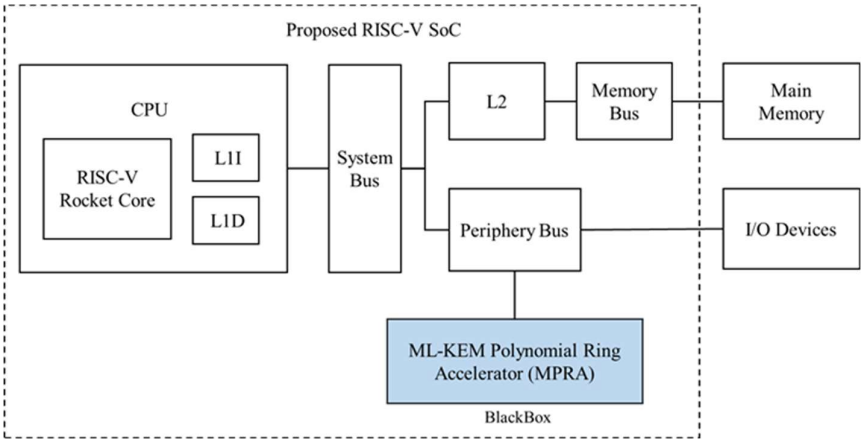

Figure 2. System architecture of the proposed RISC-V SoC with the integrated MPRA.

The implemented SoC is synthesized and deployed using Vivado 2021.1 on a Digilent Genesys 2 development board, which is equipped with a Xilinx Kintex-7 FPGA (XC7K325T-2FFG900C). The MPRA accelerator is designed and integrated in Chisel at the RTL level, and subsequently synthesized and implemented on the FPGA platform. This hardware implementation environment enables realistic evaluation of system-level performance, particularly the impact of data movement on system latency. Furthermore, it validates the design objective of performing continuous polynomial processing within the accelerator, thereby avoiding repeated accesses to main memory.

3.2. Design of the ML-KEM Polynomial Ring Accelerator

The proposed MPRA datapath is illustrated in Figure 3. The architecture is designed to perform matrix–vector multiplication over the polynomial ring R_q, which consists of three primary operations: NTT/INTT operations, polynomial pointwise multiplication, and modular addition. The implementations of the NTT and pointwise multiplication units are based on the design methodology proposed by Yaman et al. [7]. In the proposed work, the overall architecture is further optimized with a focus on data reuse and efficient memory access. A flexible buffering mechanism is introduced between the I/O interface and the computation core to reduce bus access latency and improve computational efficiency.

The functionality of each module in the datapath is described as follows:

1. Local Buffer

The local buffer serves as an intermediate buffer between the CPU interface and the computation core. It temporarily stores polynomial coefficients received from the processor and supports staged data loading and processing. This design re-duces the dependency on frequent external memory access during computation.

2. Butterfly Array

The butterfly array performs NTT and INTT operations. It employs a parallel butterfly structure to execute modular multiplication and addition over finite fields, enabling efficient transformation between the time domain and the frequency domain.

3. Multiplier Array

The multiplier array supports pointwise multiplication in the NTT domain. It performs modular multiplication on corresponding polynomial coefficients and serves as the core computational unit for polynomial ring multiplication.

4. Adder Array

The adder array performs modular addition of polynomial coefficients. It is primarily used for accumulation in matrix–vector multiplication, where intermediate results are combined across multiple polynomial products.

5. Scratchpad Memory

 doi:10.20944/preprints202605.1405.v1

The scratchpad memory stores intermediate results generated during NTT, pointwise multiplication, and modular addition stages. By keeping intermediate data within the accelerator, the design enables continuous computation without repeatedly transferring data across the system bus, thereby significantly reducing main memory access overhead.

6. Controller

The controller manages the overall execution flow and scheduling of operations. It coordinates data movement and computation among the modules and interacts with the processor through a status register. This mechanism enables synchroni-zation and provides a programmable interface for controlling the accelerator.

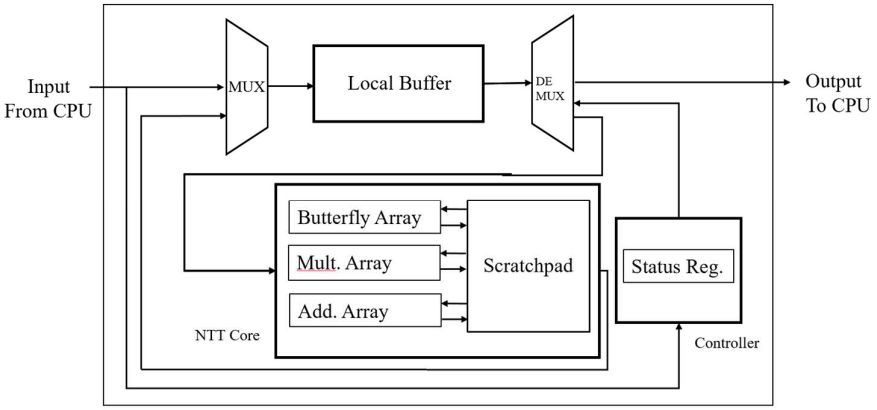

Figure 3. Datapath architecture of the proposed MPRA circuit.

3.3. Matrix–Vector Multiplication over Polynomial Rings for ML-KEM

In the ML-KEM algorithm, matrix–vector multiplication is one of the most computationally intensive operations. Therefore, the proposed work adopts it as the primary target for hardware acceleration. Taking ML-KEM-512 (Kyber512) as an example, the security parameter is $k = 2$. Each element in the matrix and vector is a polynomial defined over the ring

$$R_q = Z_{3329}[x]/(x^{256} + 1).$$

Let the polynomial vectors be defined as

$$A = [a_1(x), a_2(x)], \qquad B = [b_1(x), b_2(x)]^T.$$

The matrix–vector multiplication can be expressed as

$$C = A^T \odot B,$$

which can be expanded as

$$C = a_1(x) \odot b_1(x) \oplus a_2(x) \odot b_2(x),$$

where $\odot$ denotes polynomial multiplication over R_q, and $\oplus$ denotes modular addition.

To reduce the computational complexity of polynomial multiplication, ML-KEM employs the NTT to convert polynomials from the time domain to the NTT domain. In this domain, polynomial multiplication can be transformed into pointwise multiplication. Therefore, the computation can be reformulated as

$$C = INTT\left(\overline{a_1}(x) \odot \overline{b_1}(x) \oplus \overline{a_2}(x) \odot \overline{b_2}(x)\right),$$

where

$$\overline{a_i}(x) = NTT\left(a_i(x)\right), \overline{b_i}(x) = NTT\left(b_i(x)\right),$$

and $\odot$ denotes pointwise multiplication in the NTT domain.

As illustrated in Figure 4, the input polynomials $a_1(x), a_2(x), b_1(x), b_2(x)$ are first transformed into the NTT domain as $\overline{a_1}(x), \overline{b_1}(x), \overline{a_2}(x), \overline{b_2}(x)$, respectively. The pointwise multiplications are then performed to produce intermediate results

$$e(x) = \overline{a_1}(x) \odot \overline{b_1}(x), \qquad f(x) = \overline{a_2}(x) \odot \overline{b_2}(x).$$

These intermediate results are accumulated using modular addition,

 doi:10.20944/preprints202605.1405.v1

$$g(x) = e(x) \oplus f(x),$$

and finally transformed back to the time domain using the INTT to obtain the output polynomial

$$C = INTT\big(g(x)\big).$$

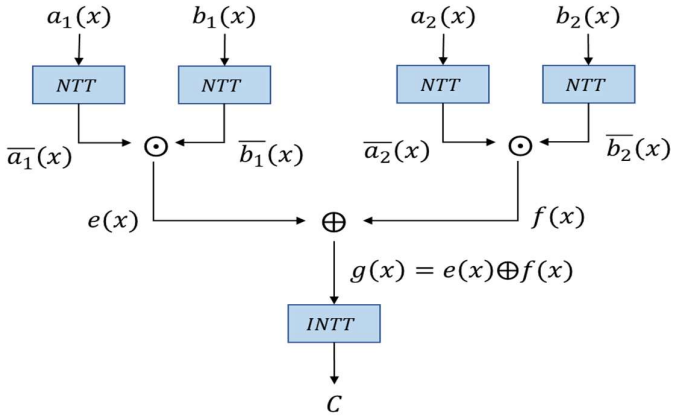

Figure 4. Matrix–Vector Multiplication over Polynomial Rings in ML-KEM-512.

In the matrix–vector multiplication of ML-KEM, each polynomial multiplication requires an initial NTT transformation, followed by pointwise multiplication and accumulation in the NTT domain. To avoid redundant memory accesses across these computation stages, the proposed polynomial ring accelerator employs an on-chip scratchpad memory to store intermediate results, enabling continuous in-accelerator processing of polynomial coefficients. As illustrated in Figure 5, at the beginning of the computation, polynomial coefficients are transferred from the CPU to the accelerator through the system bus and temporarily stored in the local buffer. The buffered data are then forwarded to the NTT core, where the butterfly array performs the NTT transformation. The transformed coefficients, such as $\overline{a_1}(x)$, are subsequently written into the scratchpad memory. This design allows the NTT-domain representation of polynomials to be retained within the accelerator and reused by subsequent computation stages, including pointwise multiplication and modular accumulation. As a result, once the NTT transformation is completed, the corresponding polynomial coefficients do not need to be reloaded from external memory for further operations. By maintaining intermediate results within the scratchpad memory, the proposed architecture minimizes off-chip memory accesses and eliminates redundant data transfers across the system bus. This dataflow organization not only improves computational efficiency but also enables effective reuse of transformed polynomial data, which is critical for accelerating matrix–vector multiplication in ML-KEM.

 doi:10.20944/preprints202605.1405.v1

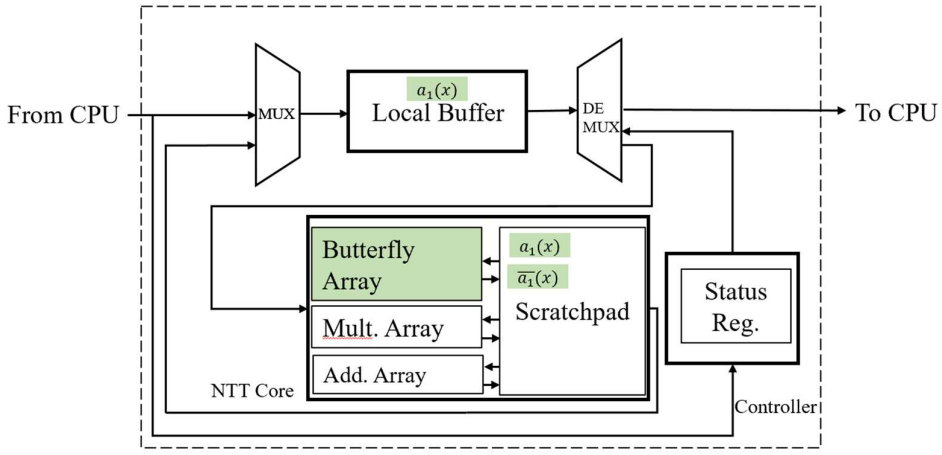

Figure 5. NTT transformation and storage of polynomial coefficients in the accelerator scratchpad.

After completing the NTT transformation, the accelerator proceeds to the pointwise multiplication stage. As illustrated in Figure 6, the transformed polynomial coefficients stored in the scratchpad memory, such as $\overline{a_1}(x)$ and $\overline{b_1}(x)$, are read and forwarded to the multiplier array to perform pointwise multiplication. This operation produces an intermediate result denoted as $e(x)$.The resulting polynomial $e(x)$ is then written back to the scratchpad memory for subsequent processing. Since matrix–vector multiplication involves multiple polynomial multiplications followed by accumulation, the NTT-domain coefficients stored in the scratchpad can be reused across different computation stages. This reuse mechanism avoids repeated NTT transformations and eliminates redundant accesses to external memory, thereby improving the overall computational efficiency of the accelerator.

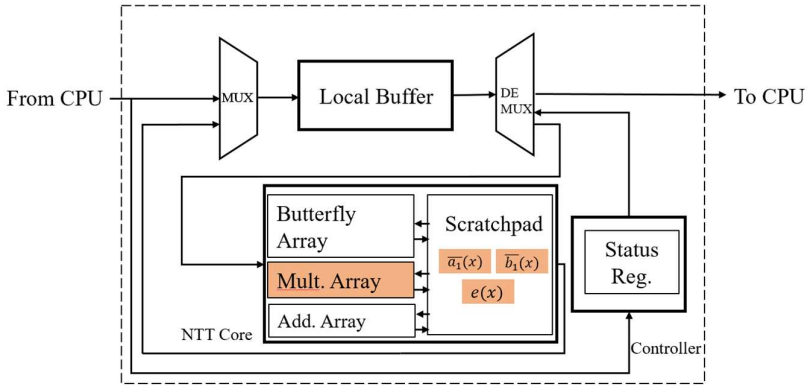

Figure 6. Pointwise multiplication using reused NTT-domain polynomial coefficients stored in the scratchpad.

After completing the pointwise multiplication, the accelerator proceeds to the accumulation stage. As illustrated in Figure 7, the intermediate results $e(x)$ and $f(x)$ are read from the scratchpad memory and forwarded to the adder array to perform modular addition. This operation produces the accumulated result

$g(x) = e(x) \oplus f(x)$,

where $\oplus$ denotes modular addition over the ring R_q.

Since matrix–vector multiplication involves the accumulation of multiple polynomial multiplication results, the adder array is responsible for performing this accumulation directly in the NTT domain. All intermediate results are maintained within the scratchpad memory throughout the computation. Through this data reuse mechanism, previously computed intermediate results can be

 doi:10.20944/preprints202605.1405.v1

directly accessed and updated by subsequent operations without requiring transfers across the system bus. This approach eliminates redundant accesses to external memory and effectively reduces system-level latency caused by data movement.

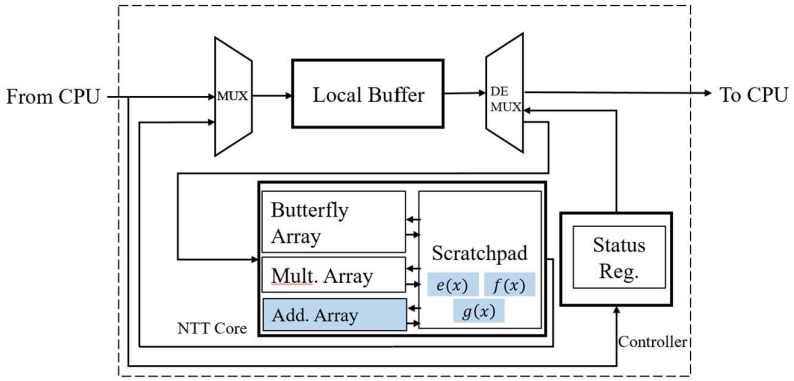

Figure 7. Modular addition accumulation in the polynomial ring computation.

After completing all pointwise multiplications and accumulation operations, the accelerator proceeds to the inverse transformation stage. As illustrated in Figure 8, the accumulated result $g(x)$ is read from the scratchpad memory and forwarded to the butterfly array to perform the INTT, which converts the data from the NTT domain back to the time-domain polynomial representation. After the INTT operation, the final polynomial result C is obtained. This result is temporarily stored in the local buffer and subsequently transferred back to the CPU through the I/O interface.

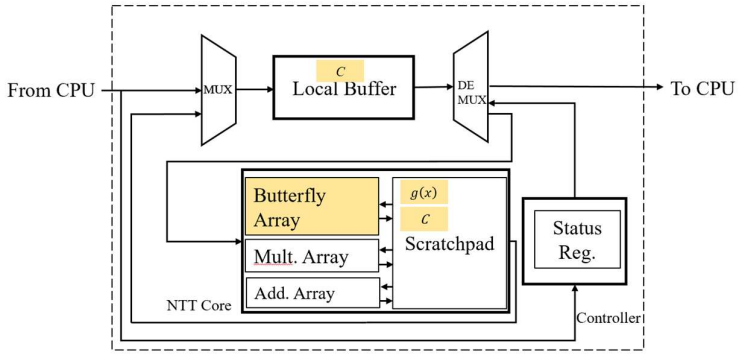

Figure 8. INTT and final result write-back to the CPU.

Through the proposed datapath design, the polynomial ring accelerator is capable of performing the complete computation flow—including NTT, pointwise multiplication, modular addition, and INTT—within a unified hardware architecture. In this design, polynomial coefficients are transferred via I/O only at the beginning and end of the computation. By enabling all intermediate operations to be executed within the accelerator, the proposed architecture significantly reduces the reliance on main memory accesses. This approach improves data locality and enhances computational efficiency, thereby accelerating matrix–vector multiplication in ML-KEM.

4. Implementation Results and Discussion

This section presents the FPGA implementation results, micro-benchmark evaluations, and system-level performance analysis of ML-KEM workloads. A systematic comparison with existing RISC-V and FPGA-based platforms is also provided. All experiments are conducted on a Xilinx Kintex-7 FPGA operating at 100 MHz.

 doi:10.20944/preprints202605.1405.v1

4.1. Hardware Resource Utilization

This subsection analyzes the hardware resource overhead introduced by integrating the proposed polynomial ring accelerator into the RISC-V SoC. Table 3 presents a comparison of hardware resource utilization on the Xilinx Kintex-7 FPGA platform, with and without the integration of the NTT-based accelerator. The integration of the proposed accelerator results in a moderate increase in logic resource usage. Specifically, the lookup table (LUT) utilization increases from 33.05% to 38.52%, while Flip-Flop (FF) utilization rises from 10.53% to 12.91%. This increase is primarily attributed to the additional computational units within the accelerator, including the butterfly array, multiplier array, adder array, and associated control logic. In terms of memory resources, Block RAM (BRAM) utilization increases from 32.13% to 39.77%. This increase is mainly due to the implementation of the local buffer and scratchpad memory, which are used to store intermediate results during NTT transformations and polynomial ring computations. These on-chip memory structures enable efficient data reuse and reduce the dependence on external memory access. Furthermore, DSP utilization increases from 1.79% to 3.69%, primarily to support pointwise multiplication operations within the multiplier array.

Overall, the proposed accelerator introduces only a modest hardware overhead while significantly improving the computational efficiency of polynomial ring operations. These results demonstrate the practicality and effectiveness of integrating the proposed design into a RISC-V SoC platform.

Table 3. FPGA resource utilization of the proposed SoC.

Implementation	LUT	FF	BRAM	DSP
Proposed SoC without MPRA	67356/203800 (33.05%)	42918/407600 (10.53%)	143/445 (32.13%)	15/840 (1.79%)
Proposed SoC with MPRA	78514/203800 (38.52%)	52613/407600 (12.91%)	177/445 (39.77%)	31/840 (3.69%)

4.2. Performance Analysis of Polynomial Ring Computation

To further highlight the differences between the proposed architecture and prior work, this subsection compares the design characteristics of representative hardware accelerators for ML-KEM-related computations. Table 4 summarizes several representative studies, including those by Karabulut et al. [26], Yaman et al. [7], Dam et al. [19], Celik et al. [27], and the proposed design in this work. Most existing studies primarily focus on accelerating single polynomial multiplication using NTT/INTT-based approaches. For example, Karabulut et al. [26] propose a RISC-V instruction set extension to accelerate NTT operations. Yaman et al. [7] develop a dedicated hardware accelerator that leverages NTT and pointwise multiplication to improve the efficiency of polynomial multiplication in CRYSTALS-Kyber. Similarly, Dam et al. [19] integrate an NTT-based accelerator into a RISC-V SoC platform to enhance polynomial computation performance. In contrast, Celik et al. [27] adopt a different optimization approach by accelerating the Keccak hash function within the Kyber algorithm. Their design targets the most time-consuming cryptographic primitive identified through profiling and implements a hardware Keccak core within a RISC-V SoC using a hardware–software co-design methodology. However, their work does not address polynomial arithmetic operations such as NTT, polynomial multiplication, or matrix–vector multiplication. Despite these advancements, most prior works primarily target the acceleration of individual polynomial operations or specific cryptographic primitives. However, the dominant computational workload in the ML-KEM algorithm is matrix–vector multiplication over polynomial rings, which involves multiple polynomial multiplications followed by modular accumulation. As a result, accelerating only a single operation still requires software-controlled data movement and scheduling across multiple computation stages.

 doi:10.20944/preprints202605.1405.v1

The proposed architecture directly targets matrix–vector multiplication over polynomial rings at the hardware level. The accelerator integrates NTT/INTT computation units, a multiplier array for pointwise multiplication, and an adder array for modular addition, enabling the entire polynomial computation flow to be executed within a unified hardware datapath. In addition, a scratchpad memory is employed to store intermediate results, allowing NTT-transformed coefficients to be reused in subsequent pointwise multiplication and accumulation stages. This design avoids frequent accesses to external memory across different computation stages and improves the overall efficiency of polynomial ring computations.

Table 4. The design characteristics of representative hardware accelerators for ML-KEM-related computations.

Work	Target Operation	NTT Accelerator	Polynomial Multiplication	Modular Addition	Matrix–Vector Multiplication	Hash Accel-erator	SoC-Level Evaluation
Yaman et al. [7]	Polynomial Multiplication	✓	✓	✗	✗	✗	Partial
Dam et al. [19]	Polynomial Multiplication	✓	✓	✗	✗	✗	Partial
Karabulut et al. [26]	Polynomial Multiplication	✓	✓	✗	✗	✗	Partial
Celik et al. [27]	Keccak Acceleration	✗	✗	✗	✗	✓	Full
Proposed Work	Matrix–Vector Multiplication over Polynomial Rings	✓	✓	✓	✓	✗	Full

Table 5 presents the performance comparison of the proposed polynomial ring accelerator with a software implementation and existing hardware–software co-design approaches. In the proposed design, a polynomial multiplication consists of two NTT transformations, one pointwise multiplication, and one INTT transformation. The entire computation requires 5483 clock cycles, corresponding to a latency of 54.83 μs at an operating frequency of 100 MHz. The cycle distribution across different computation stages can be summarized as follows:

- Data load time requires 2967 cycles;
- NTT computation requires 280 cycles;
- Pointwise multiplication requires 122 cycles;
- INTT computation requires 150 cycles;
- Data store time requires 1964 cycles.

From this breakdown, it can be observed that the computational cost of the NTT/INTT cores is relatively small compared to the overall execution time, while data movement accounts for a significant portion of the total latency. Therefore, reducing data access and transfer overhead is critical for improving system-level performance. Compared with the reference software implementation, which requires 143,196 cycles for polynomial computation, the proposed hardware accelerator significantly reduces execution time. This improvement is primarily achieved through the hardware implementation of NTT and pointwise multiplication, enabling polynomial computations to be performed within the accelerator. In addition, the proposed design is compared with existing hardware–software co-design approaches. For example, Karabulut et al. [26] accelerate NTT operations using RISC-V instruction set extensions; however, data movement and control are still handled by the processor, resulting in 43,756 cycles for NTT computation. Dam et al. [19] propose an

 doi:10.20944/preprints202605.1405.v1

SoC design integrating an NTT module, where the combined cost of NTT computation and data movement is 9842 cycles.

Table 5. Performance Breakdown of Polynomial Ring Operations.

Implementations	Steps		Clocks	Total Clocks	Latency (μs)
Proposed work	NTT (2 NTTs)	Data Load Time	2967	5483	54.83
		NTT Core for NTT	280		
	Pairwise Multiplication		122		
	INTT (1 INTT)	NTT Core for INTT	150		
		Data Store Time	1964		
Dam et al. [19]	NTT (1 NTT)	Data Load Time	2084	9842	✗
		NTT Core for NTT	5682		
		Data Store Time	2076		
Karabulut et al. [26]	NTT (1 NTT)	Data Load Time NTT Core for NTT Data Store Time	43756	43756	✗
Kyber C code [30]	NTT		66394	143196	1431.96
	Pairwise Multiplication		18686		
	INTT		56098		

In addition to polynomial ring operations, the proposed work further evaluates the overall performance of the proposed accelerator at the matrix–vector multiplication level, which speeds up the computation for Equation (2) in Section 2.1. In the Kyber/ML-KEM algorithm, matrix–vector multiplication represents one of the dominant computational workloads, and its performance directly impacts overall system efficiency. As shown in Table 6, a pure software implementation requires 296,485 clock cycles to complete a single matrix–vector multiplication, corresponding to a latency of approximately 2964.85 μs at a clock frequency of 100 MHz. When only pointwise multiplication is accelerated in hardware, without optimizing modular addition using the adder array, the required number of clock cycles is reduced to 12,037 cycles, achieving a performance improvement of approximately 24.6× compared to the software baseline. By further incorporating the proposed adder array to accelerate modular addition, the accumulation of intermediate results in the NTT domain is significantly improved. As a result, the total computation time is further reduced to 7,372 cycles, corresponding to a latency of 73.72 μs. Compared with the Kyber C reference implementation [30], the proposed architecture achieves an overall performance improvement of approximately 40.2×. These results demonstrate that efficient modular accumulation within the accelerator, combined with reduced intermediate data movement, plays a critical role in improving the performance of matrix–vector multiplication in ML-KEM.

Table 6. Performance comparison of matrix–vector multiplication for ML-KEM.

Implementations	Clocks	Latency (μs)	Speed up
Kyber C code [30]	296485	2964.85	1
Proposed HW Accelerator with Pointwise Multiplication Only	12037	120.37	24.6

 doi:10.20944/preprints202605.1405.v1

Proposed HW Accelerator with Pointwise Multiplication and Modular Addition Support	7372	73.72	40.2

4.3. SoC-Level Performance Evaluation of ML-KEM

To evaluate the practical system-level performance of the proposed polynomial ring accelerator, the ML-KEM algorithm is integrated into the RISC-V SoC platform for end-to-end testing. By offloading the primary computational workload in the Kyber/ML-KEM algorithm—namely polynomial ring operations and matrix–vector multiplication—to the proposed hardware accelerator, the overall execution time can be effectively reduced. Table 7 presents the system-level performance comparison of ML-KEM under different security parameter sets, including ML-KEM-512, ML-KEM-768, and ML-KEM-1024. The baseline results are obtained by executing the NIST reference Kyber C implementation on the Rocket core, while the accelerated results are measured using the proposed polynomial ring accelerator. As shown in the table, under the ML-KEM-512 parameter set, the execution time of encapsulation is reduced from 929,391 cycles to 554,479 cycles, achieving a speedup of approximately 1.67×. Similarly, decapsulation is reduced from 1,037,658 cycles to 492,533 cycles, achieving a speedup of approximately 2.10×.

Table 7. SoC-Level Performance of ML-KEM with the Proposed Accelerator.

Algorithm	Operation	Kyber C code [30]	Proposed work	Speed-up
ML-KEM 512	Encaps	929391	554479	1.67
	Decaps	1037658	492533	2.10
ML-KEM 768	Encaps	1447649	877912	1.64
	Decaps	1604701	788022	2.03
ML-KEM 1024	Encaps	2120848	1295500	1.63
	Decaps	2317043	1193890	1.94

Similar performance improvements are observed under higher security levels, including ML-KEM-768 and ML-KEM-1024. For ML-KEM-768, the proposed design achieves speedups of 1.64× and 2.03× for encapsulation and decapsulation, respectively. For ML-KEM-1024, the corresponding speedups are 1.63× and 1.94×. These results indicate that offloading polynomial ring computations to the NTT-based hardware accelerator, combined with an effective data reuse mechanism, can significantly reduce the overall computational workload of ML-KEM on the SoC platform. Furthermore, to provide a system-level comparison, the proposed design is evaluated against the ML-KEM hardware accelerator reported by Celik et al. [27], as shown in Table 8. Their design employs an Ibex core (RV32IMC) operating at 50 MHz, whereas the proposed system uses a Rocket core (RV64GC) operating at 100 MHz. Based on the comparison results for Kyber-768, the proposed design achieves a substantial reduction in the number of required clock cycles for both encapsulation and decapsulation operations. This demonstrates that the proposed polynomial ring accelerator maintains strong performance advantages even after full system-level integration.

Table 8. SoC-level performance comparison for Kyber-768.

Work	CPU	ISA	Clock Rate	Execution Time	Encaps (SW)	Encaps (SW/HW)	Decaps (SW)	Decaps (SW/HW)
Celik et al. [27]	Ibex Core	RV32IMC	50MHz	Cycles	5886277	3115537	6222787	3957289
				Latency (μs)	117725.54	62310.74	124455.74	79145.78
		RV64GC		Cycles	1447649	877912	1604701	788022

 doi:10.20944/preprints202605.1405.v1

Proposed Work	Rocket Core	100MHz	Latency (μs)	14476.49	8779.12	16047.01	7880.22

Overall, the experimental results demonstrate that integrating the proposed polynomial ring accelerator into the RISC-V SoC platform effectively accelerates the most critical computational components of the ML-KEM algorithm. Consistent and significant performance improvements are observed across different security parameter sets. Compared with existing RISC-V SoC-based implementations, the proposed design achieves a substantial reduction in end-to-end execution cycles at an operating frequency of 100 MHz. These results validate the effectiveness of the proposed dataflow optimization strategy, particularly in reducing data movement overhead, and highlight its practical benefits for accelerating complete cryptographic workloads.

5. Conclusions

This paper presents an ML-KEM polynomial ring computation hardware accelerator based on Open IP and open hardware design principles, and demonstrates its system- level integration and implementation on a Chipyard-based RISC-V SoC platform. Focusing on the dominant computational workload in ML-KEM—matrix–vector multiplication over polynomial rings—the proposed architecture integrates NTT/INTT, pointwise multiplication, and modular addition into a unified hardware datapath. By employing scratchpad memory to enable intermediate data reuse, the design effectively reduces data movement across the system bus and improves overall computational efficiency. In contrast to prior works that primarily optimize individual modules such as NTT or standalone polynomial multiplication units, the proposed work adopts a system-level design approach. The accelerator is integrated into the RISC-V SoC via a MMIO interface, allowing it to operate within a complete processor and operating system environment. This approach enhances integration flexibility and enables practical hardware–software co-design, improving both reproducibility and applicability for PQC system deployment.

Experimental results on a Kintex-7 FPGA platform demonstrate significant performance improvements for ML-KEM-related computations. At the polynomial computation level, the proposed design substantially reduces the number of execution cycles compared to software implementations. At the matrix–vector multiplication level, the scratchpad-based data reuse mechanism achieves approximately 16× performance improvement. At the full ML-KEM system level, speedups of approximately 1.6× to 2.1× are observed across different security parameter sets. These results further indicate that data movement overhead is a key factor affecting system performance, highlighting the importance of on-chip memory and data reuse strategies in PQC hardware acceleration. Through the systematic comparisons presented in Table 1 and Table 2, it is observed that most existing works are limited to arithmetic module optimization or partial algorithm acceleration, and often lack integration with open hardware platforms and system-level validation. In contrast, the proposed work simultaneously achieves (1) complete ML-KEM polynomial ring acceleration, (2) integration within a RISC-V SoC, and (3) implementation and validation on an open hardware platform with operating system support. This establishes a reproducible, scalable, and system-oriented research framework for PQC hardware design.

Future work can be extended from the perspective of Open IP and open hardware ecosystems. First, standardized PQC hardware interfaces can be developed to integrate reusable open IP modules, such as NTT units, polynomial arithmetic engines, and hash functions (e.g., Keccak), forming a modular PQC accelerator IP library. Second, further exploration of integration across different open hardware SoC platforms, such as OpenTitan and OpenHW CORE-V, can provide insights into architectural trade-offs and performance characteristics in PQC system design. In addition, leveraging generator-based hardware design frameworks such as Chipyard, future work may enable parameterized and automated generation of PQC accelerators, improving design reproducibility and deployment efficiency. At the system level, future efforts may also focus on integrating operating systems and driver frameworks to establish standardized hardware–software interfaces, allowing applications to access PQC acceleration through unified APIs. Finally, taking advantage of the

 doi:10.20944/preprints202605.1405.v1

transparency of open hardware, future research may explore side-channel attack resistance and formal verification techniques to further enhance the security and trustworthiness of PQC hardware systems in practical deployments.

Author Contributions: Conceptualization, Y.-C.T. ; methodology, Y.-C.T.; software, Y.-C.T. and Y.-H.L.; validation, Y.-C.T. and W.-J.H.; formal analysis, Y.-C.T.; investigation, Y.-C.T.; resources, Y.-C.T. and W.-J.H.; data curation, Y.-C.T.; writing—original draft preparation, Y.-C.T.; writing—review and editing, Y.-C.T. and W.-J.H.; visualization, Y.-C.T.; supervision, W.-J.H.; project administration, W.-J.H.; funding acquisition, W.-J.H. All authors have read and agreed to the published version of the manuscript.

Funding: The original research work presented in this paper was made possible, in part, by the National Science and Technology Council, Taiwan, under grants MOST 111-2221-E-003-009-MY2 and NSTC113-2221-E-003-027-MY2.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data sharing is not applicable to this article.

Acknowledgments: The authors would like to thank the members of the laboratory for their technical support and helpful discussions throughout this work.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

BRAM	Block RAM
CPU	Central Processing Unit
DDR	Double Data Rate
DSP	Digital Signal Processing
FF	Flip-Flop
FIPS	Federal Information Processing Standards
FPGA	Field-Programmable Gate Array
INTT	Inverse Number Theoretic Transform
IP	Intellectual Property Core
ISA	Instruction Set Architecture
LUT	Look-Up Table
ML-DSA	Module-Lattice-Based Digital Signature Algorithm
ML-KEM	Module-Lattice-Based Key Encapsulation Mechanism
MMIO	Memory-Mapped I/O
MPRA	ML-KEM Polynomial Ring Accelerator
NTT	Number Theoretic Transform
OS	Operating System
OTBN	OpenTitan Big Number
PQC	Post-Quantum Cryptography
RISC-V	Open-standard RISC instruction set architecture
RSA	Rivest–Shamir–Adleman
RTL	Register-Transfer Level
RV64GC	RISC-V 64-bit General-purpose ISA with Compressed extension
SHAKE	Secure Hash Algorithm Keccak
SoC	System-on-Chip
XOF	Extendable Output Function

 doi:10.20944/preprints202605.1405.v1

Appendix A

Table A1. A list of symbols used in this study.

R_q	Polynomial ring defined as $Z_q[x]/(x^n+1)$.
q	Modulus used in ML-KEM; in the proposed work, $q = 3329$.
n	Polynomial degree parameter; in ML-KEM, $n = 256$.
k	Security-level-dependent dimension parameter in ML-KEM.
$a_i(x), b_i(x)$	Input polynomials in R_q.
A	Polynomial vector or matrix operand in matrix–vector multiplication.
B	Polynomial vector operand in matrix–vector multiplication.
C	Output polynomial or matrix–vector multiplication result.
$a_1(x), a_2(x)$	Example input polynomials from vector A in the ML-KEM-512 case.
$b_1(x), b_2(x)$	Example input polynomials from vector B in the ML-KEM-512 case.
$\bar{a_i}(x)$	NTT-domain representation of $a_i(x)$, i.e., $NTT(a_i(x))$.
$\bar{b_i}(x)$	NTT-domain representation of $a_i(x)$, i.e., $NTT(b_i(x))$.
$e(x)$	Intermediate result of the first pointwise multiplication in the NTT domain.
$f(x)$	Intermediate result of the second pointwise multiplication in the NTT domain.
$g(x)$	Accumulated intermediate result in the NTT domain before INTT.
$NTT(\cdot)$	Number Theoretic Transform.
$INTT(\cdot)$	Inverse Number Theoretic Transform.
$\circledcirc$	Polynomial multiplication over R_q.
$\odot$	Pointwise multiplication in the NTT domain.
$\oplus$	Modular addition over R_q.

References

1. Shor, P.W. Algorithms for quantum computation: Discrete logarithms and factoring. In Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS), Santa Fe, NM, USA, 20–22 November 1994; pp. 124–134.
2. National Institute of Standards and Technology (NIST). Migration to Post-Quantum Cryptography; NIST Interagency Report (IR) 8547 (Initial Public Draft); NIST: Gaithersburg, MD, USA, 2024. https://doi.org/10.6028/NIST.IR.8547.ipd.
3. National Institute of Standards and Technology (NIST). Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM); FIPS 203; NIST: Gaithersburg, MD, USA, 2024. https://doi.org/10.6028/NIST.FIPS.203.
4. Tan, W.; Lao, Y.; Parhi, K.K. KyberMat: Efficient accelerator for matrix–vector polynomial multiplication in CRYSTALS-Kyber. In Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD), San Francisco, CA, USA, 29 October–2 November 2023. https://doi.org/10.1109/ICCAD57390.2023.10323839.
5. Bos, J.W.; Ducas, L.; Kiltz, E.; Lepoint, T.; Lyubashevsky, V.; Schanck, J.M.; Schwabe, P.; Seiler, G.; Stehlé, D. CRYSTALS-Kyber: A CCA-secure module-lattice-based KEM. In Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P), London, UK, 24–26 April 2018; pp. 353–367. https://doi.org/10.1109/EuroSP.2018.00032.
6. Waris, A.; Aziz, A.; Khan, B.M. Area-time efficient pipelined number theoretic transform architecture for CRYSTALS-Kyber. IEEE Access 2021, 9, 109424–109438. https://doi.org/10.1109/ACCESS.2021.3099274.
7. Yaman, F.; Mert, A.C.; Öztürk, E.; Savaş, E. A hardware accelerator for polynomial multiplication operation of CRYSTALS-Kyber PQC scheme. In Proceedings of the Design, Automation and Test in Europe Conference (DATE), Grenoble, France, 1–5 February 2021. https://doi.org/10.23919/DATE51398.2021.9474139.

 doi:10.20944/preprints202605.1405.v1

8. Wang, T.; Zhang, C.; Zhang, X.; Gu, D.; Cao, P. Hardware–software co-design for Kyber and Dilithium on RISC-V SoC FPGA. IACR Trans. Cryptogr. Hardw. Embed. Syst. 2024, 3, 99–135. https://doi.org/10.46586/TCHES.V2024.I3.99-135.
9. Amid, A.; Biancolin, D.; Lee, A.; et al. Chipyard: An integrated design framework for custom SoCs. IEEE Micro 2020, 40, 10–21. https://doi.org/10.1109/MM.2020.2996616.
10. Pramstaller, N.; Zaruba, F.; Benini, L.; et al. Ibex: A small, efficient RISC-V processor core. OpenTitan Project Documentation, 2020. Available online: https://opentitan.org (accessed on 2 April 2026).
11. OpenTitan Project. OpenTitan: Open source silicon root of trust. Available online: https://opentitan.org (accessed on 2 April 2026).
12. Liu, S.-H.; Kuo, C.-Y.; Mo, Y.-N.; Su, T. An area-efficient, conflict-free, and configurable architecture for accelerating NTT/INTT. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 2024, 32, 519–529. https://doi.org/10.1109/TVLSI.2023.3336951.
13. Kim, H.; Jung, H.; Satriawan, A.; Lee, H. A configurable ML-KEM/Kyber hardware accelerator. IEEE Trans. Circuits Syst. II 2024, 71, 4678–4682. https://doi.org/10.1109/TCSII.2024.3442228.
14. Ni, Z.; Khalid, A.; Liu, W.; O'Neill, M. A highly hardware-efficient ML-KEM accelerator. ACM Trans. Embed. Comput. Syst. 2025, 24, 1–24. https://doi.org/10.1145/3708469.
15. Cui, Y.; Chen, J.; Ni, Z.; Zhang, Z.; Wang, C.; Liu, W. Instruction-based hardware controller of CRYSTALS-Kyber. IEEE Trans. Circuits Syst. I 2025, 72, 2394–2407.
16. Dolmeta, A.; Valpreda, E.; Martina, M.; Masera, G. Integration of NTT/INTT accelerator on RISC-V. In Proceedings of the ACM Computing Frontiers Conference (CF), Ischia, Italy, 7–9 May 2024; pp. 59–62. https://doi.org/10.1145/3637543.3652872.
17. Abdulrahman, A.; Oberhansl, F.; Pham, H.N.; Philipoom, J.; Schwabe, P.; Stelzer, T.; Zankl, A. Towards ML-KEM and ML-DSA on OpenTitan. In Proceedings of the IEEE Symposium on Security and Privacy, San Francisco, CA, USA, 2025. https://doi.org/10.1109/SP61157.2025.00220.
18. OpenHW Group. CORE-V CV32E40P RISC-V processor core user manual. Available online: https://docs.openhwgroup.org/projects/cv32e40p-user-manual (accessed on 2 April 2026).
19. Dam, D.-T.; Nguyen, T.-H.; et al. RISC-V SoC with NTT-Blackbox. In Proceedings of the ICDV 2024; pp. 49–54. https://doi.org/10.1109/ICDV61346.2024.10617195.
20. Dam, D.-T.; Nguyen, K.-D.; Le, D.-H.; Pham, C.-K. High-efficiency NTT for ML-KEM on RISC-V. Electronics 2026, 15, 100. https://doi.org/10.3390/electronics15010100.
21. Rumelili Köksal, C.I.; Örs Yalçın, S.B. Efficient modeling and usage of scratchpad memory. Electronics 2025, 14, 1032. https://doi.org/10.3390/electronics14051032.
22. Huang, Y.; Zhao, Y.; Chen, Z.; Li, X. High-Speed NTT-Based Polynomial Multiplication Accelerator for CRYSTALS-Kyber Post-Quantum Cryptography. IEEE Access 2020, 8, 203000–203012. https://doi.org/10.1109/ACCESS.2020.3036204.
23. Chen, Z.; Ma, Y.; Chen, T.; Lin, J.; Jing, J. Towards efficient Kyber on FPGAs: A processor for vector of polynomials. In Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC), Beijing, China, 13–16 January 2020; pp. 247–252.
24. Bisheh-Niasar, M.; Azarderakhsh, R.; Mozaffari Kermani, M. High-Speed NTT-Based Polynomial Multiplication Accelerator for Post-Quantum Cryptography. In Proceedings of the 28th IEEE Symposium on Computer Arithmetic (ARITH), Virtual Conference, 14–16 June 2021; pp. 94–101. https://doi.org/10.1109/ARITH51176.2021.00028.
25. Zhang, X.; Liu, D.; Chen, Z.; Jing, J. Towards Efficient Hardware Implementation of NTT for Kyber on FPGAs. In Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), Daegu, Republic of Korea, 22–28 May 2021; pp. 1–5. https://doi.org/10.1109/ISCAS51556.2021.9401170.
26. Karabulut, A.; et al. RANTT: A RISC-V architecture extension for the number theoretic transform. In Proceedings of the International Conference on Field-Programmable Logic and Applications (FPL), Gothenburg, Sweden, 31 August–4 September 2020; pp. 26–32. https://doi.org/10.1109/FPL50879.2020.00016.
27. Celik, A.; Yilmaz, F.; Korkmaz, M.A.; Ors, B. Implementation of CRYSTALS-Kyber post-quantum algorithm using RISC-V processor. In Proceedings of the IEEE International Conference on Electronics,

 doi:10.20944/preprints202605.1405.v1

Circuits and Systems (ICECS), Istanbul, Turkey, 4–7 December 2023. https://doi.org/10.1109/ICECS58634.2023.10382743.

28. Avanzi, R.; Bos, J.; Ducas, L.; Kiltz, E.; Lepoint, T.; Lyubashevsky, V.; Schanck, J.M.; Schwabe, P.; Seiler, G.; Stehlé, D. CRYSTALS-Kyber algorithm specifications and supporting documentation; Third-round submission to the NIST post-quantum cryptography standardization process, 2020. Available online: https://csrc.nist.gov/projects/post-quantum-cryptography (accessed on 2 April 2026).