International Journal of Software and Informatics, ISSN 1673-7288
http://www.ijsi.org, ijsi@iscas.ac.cn, +86-10-62661048
IJSI, 2025, 15(4): 435–463, doi: 10.21655/ijsi.1673-7288.00358

International Journal of Software and Informatics

Research Article

Hardware Optimization of Lattice-based Digital Signature Algorithm Based on FPGA

Yue Hu (胡跃), Xuyang Zhao (赵旭阳), Wei Wang (王威), Qian Yuan (袁谦), Jieyu Zheng (郑婕妤), Yafang Yang (杨亚芳)

(School of Computer Science, Fudan University, Shanghai 200433, China)
Corresponding author: Xuyang Zhao, xuyangzhao21@m.fudan.edu.cn

Abstract Digital signature algorithms play a vital role in network security infrastructure. The majority of current digital signature schemes rely on RSA and ECC. However, with the rapid advancement of quantum computing, traditional public-key cryptographic schemes face increasing security risks. As a result, researching and deploying cryptographic schemes capable of resisting quantum attacks has become a critical research direction. Following multiple rounds of evaluation and analysis, National Institute of Standards and Technology (NIST) announced the post-quantum digital signature standard ML-DSA in August 2024, with Dilithium as its core algorithm. In light of the high-dimensional polynomial matrix operations characteristic of Dilithium, this paper proposes various optimization strategies based on the FPGA platform. These include multifunctional systolic array operation units with configurable parameters, dedicated polynomial parallel sampling modules, reconfigurable storage units designed for multiple parameter sets, and high-parallelism timing state machines tailored for complex multi-module architectures. These optimizations aim to overcome performance bottlenecks and achieve enhanced signature operation efficiency, ultimately realizing a digital signature hardware architecture that supports three security levels simultaneously. The proposed hardware architecture is deployed and evaluated on the Xilinx Artix-7 FPGA platform and compared against existing implementations. The results demonstrate that the proposed design achieves improvements in signature operation efficiency by factors of 7.4, 8.3, and 5.6 across the three security levels, respectively. This advancement provides a robust performance foundation for quantum-resistant digital signature applications and offers valuable insights for the engineering and practical deployment of lattice cryptographic schemes.

Keywords post-quantum cryptography; lattice-based cryptography; digital signature algorithm; FPGA; hardware implementation

Citation Hu Y, Zhao XY, Wang W, Yuan Q, Zheng JY, Yang YF. Hardware optimization of lattice-based digital signature algorithm based on FPGA, *International Journal of Software and Informatics*, 2025, 15(4): 435–463. http://www.ijsi.org/1673-7288/358.htm

---

This is the English version of the Chinese article “基于FPGA的格基数字签名算法硬件优化. 软件学报, 2025, 36(10): 4461–4482. doi: 10.13328/j.cnki.jos.007389”
Yue Hu and Xuyang Zhao are co-first authors of this paper.
Funding items: National Key Research and Development Program (2022YFB2701601); Collaborative Innovation Fund of Shanghai (XTCX-KJ-2023-54); Blockchain Key Technology Research and Development Special Fund of Shanghai Science and Technology Commission (23511100300)
Received 2024-06-30; Revised 2024-09-05; Accepted 2024-12-30; IJSI published online 2025-12-30

The public key cryptography algorithm system serves as the cornerstone of information security in modern society. RSA and ECC, as the two most important algorithms within this system[1], underpin advanced cryptographic applications such as key encapsulation mechanisms, digital signature protocols, and homomorphic encryption. These applications ensure the confidentiality, integrity, and non-repudiation of core data elements for communicating parties, leading to their widespread adoption on emerging computing platforms like cloud computing, the Internet of Vehicles, and blockchain. However, with the rapid development of quantum computing technology, traditional cryptographic schemes based on integer factorization and discrete logarithms face the risk of being broken in polynomial time by computer systems running quantum algorithms. The most renowned quantum algorithms in this context are the Shor's algorithm[2, 3], which is based on the Fourier transform, and the search algorithm for unstructured databases proposed by Grover[4]. Among them, Grover's algorithm is often used for the analysis and cryptanalysis of symmetric algorithms (such as the AES algorithm), while Shor's algorithm is primarily used to attack cryptographic schemes based on elliptic curves. To address the threat posed by quantum computing to the security of existing public key cryptography infrastructure, post-quantum cryptography (PQC) has garnered significant attention and become a key research focus in both academic and industrial circles[5, 6].

NIST initiated a global call for standardization proposals for PQC algorithms in 2016[5, 7]. These proposals encompassed five fundamental construction methods: hash-based[8], code-based[9], multivariate-based[10], isogeny-based[11], and lattice-based[12]. The lattice-based approach received significant research attention and focus due to its superior ability to balance security, public/private key size, and implementation efficiency. According to the selection results of the NIST's PQC standardization project released in July 2022, three of the first four standardized algorithms were lattice-based cryptographic schemes. Among them, CRYSTALS-Kyber[13] was selected as the standard for key encapsulation mechanisms, while CRYSTALS-Dilithium[14] and Falcon[15] were chosen as digital signature algorithms. In a recent research report, NIST conducted a comprehensive performance comparison of these two digital signature algorithms on the ARM-Cortex-M4 platform[16]. Falcon's design involves extensive floating-point operations, which complicates the practical deployment of the signature scheme. The results indicate that Dilithium's signature generation efficiency is significantly higher than Falcon's, while also consuming fewer storage resources. From a security perspective, the Dilithium scheme ensures that a unique public key and plaintext message correspond to a given signature, providing strong unforgeability of signatures. Consequently, NIST explicitly recommended Dilithium as the standardized scheme for post-quantum digital signature algorithms in its published report, with the final standardization document released in August 2024[17], naming it ML-DSA as the digital signature standard.

Dilithium is a classic digital signature scheme based on lattice theory, whose security relies on the module learning with errors (MLWE) and module short integer solution (MSIS) problems[18]. Most operations in Dilithium involve multiplication over polynomial rings, which can be accelerated using the number-theoretic transform (NTT) technique. This design structure is highly suitable for optimization and practical deployment on both software and hardware platforms, giving it promising prospects for engineering and practical application. Furthermore, the Dilithium scheme adopts the Fiat-Shamir with Aborts structure, where the signing process includes a series of conditional checks and rejection steps. This process ensures that the generated signatures do not leak any private key information. In view of the rapid development of quantum computing and the irreplaceable role of digital signature technology in network communication security, coupled with the high demands for deployment flexibility and security standard diversity from emerging computing platforms like blockchain and cloud computing, designing and implementing specialized, reconfigurable, and loosely-coupled integrated circuits for post-quantum digital signature algorithms has become an important research direction in the PQC field. Such hardware aims to provide high-performance and quantum-resistant services, including identity authentication, data integrity protection, non-repudiation, authorization, and privacy protection, for core data across diverse application scenarios. We propose a novel complete hardware implementation scheme targeting the latest parameter sets of the Dilithium algorithm, based on the Artix-7 series FPGA platform. Through targeted optimization of functional modules and by fully leveraging the module-level parallelism of FPGAs to further break through the performance bottlenecks of signature operations, this work aims to provide valuable hardware design references for the practical adoption process of the Dilithium algorithm. The specific contributions are as follows:

(1) To address the characteristic of large computational data volume in the signature algorithm, we design a dedicated high-throughput systolic array unit as the core of the polynomial arithmetic module. The memory access logic of NTT operations are desiged by configurable hardware resources enabling unified implementation of eight parallel NTT, INTT, point-wise multiplication (PWM), and modular addition operations. This design not only significantly improves the computational efficiency of critical processes but also standardizes the timing differences between different types of operations, facilitating the design and optimization of an overall pipelined architecture.

(2) The characteristics of polynomial coefficients and the types of sampling operations within the signature algorithm are analyzed. Targeting these, the internal register structure of the Keccak core is optimized, leading to the proposal of two hash computation modules: a general-purpose type and a sampling-optimized type. This approach effectively reduces hardware resource consumption while increasing the parallelism of sampling operations and achieving a higher clock frequency.

(3) For the parameter sets corresponding to the three security levels of the signature algorithm, the variations in polynomial coefficient types and vector storage depth are studied. The storage array is designed and implemented based on the parameters for the highest security level. The reconfiguration of storage units is accomplished through the utilization of top-level signals, thereby optimizing hardware resource utilization while simultaneously addressing multi-dimensional storage requirements.

(4) The number of cycles for core computational steps is rigorously analyzed. Simultaneously, by fully considering the high-throughput operation characteristics of the systolic array unit, the key computational steps of the signature algorithm are decomposed and reorganized at a fine granularity. Core computational functions are implemented at the vector scale, leading to a novel segmented timing and state control logic design scheme, which further enhances the overall operational efficiency of the algorithm.

1 Related Work on Software and Hardware Implementations of Signature Algorithms

Optimizing algorithm implementations on various software and hardware platforms is a significant research direction in the field of cryptographic engineering, holding considerable practical importance for promoting the engineering application and practical deployment of post-quantum cryptographic algorithms. Software optimization implementations typically utilize SIMD instruction sets to accelerate complex and time-consuming operations in lattice-based cryptosystems through parallel processing. These primarily fall into two categories: AVX2/AVX-512 instruction sets based on Intel processors and NEON instruction sets based on ARM processors. Since AVX2 was a mandatory reference implementation for all submissions during the PQC standardization process, and NIST selected the ARM Cortex-M4 processor (ARMv7 architecture) as one of the performance evaluation platforms for the competition[19], there are already numerous pure software optimization schemes for the Dilithium signature algorithm[20, 21]. However, given that instruction sets primarily accelerate operations at the register level, this limits further overall performance improvements in software implementations. Pure hardware schemes, designed using programmable fundamental logic resources, can achieve higher computational performance and played a crucial role in advancing the practical adoption of the Dilithium signature algorithm. They also represented a key research area in post-quantum cryptographic engineering[22–27]. Additionally, software-hardware co-design approaches primarily focus on implementing the more computationally intensive parts of the signature algorithm as hardware accelerators, while the software part handles data scheduling and logical control. This approach balances the flexibility of software with the efficiency of hardware and thus has garnered considerable research attention[28–33]. Among them, Banerjee *et al*.[28] were the first to implement all second-round candidate algorithms, including Dilithium, based on a RISC-V processor, presenting a relatively general design that showed significant performance improvement compared to implementations on ARM platforms. Zhao *et al*.[30] also proposed a new high-performance design for MLWE-LBC based on a RISC-V processor. Their work considered both data-level parallelism in RLWE structures and instruction-level parallelism in MLWE structures, designing a highly parallel NTT processor architecture based on the concept of vector processors proposed by Xin *et al*.[31], further enhancing the implementation efficiency of signature operations. Wang *et al*.[25, 32] first proposed a pure hardware design for the Dilithium algorithm based on the Zynq-7000 platform and subsequently proposed a design based on the PolarFire platform that supports both Dilithium and Kyber algorithms. According to the test results in the literature, the overall implementation performance improved as the level of hardware implementation increased.

Compared to pure software and software-hardware co-design approaches, research on pure hardware implementations of the Dilithium algorithm is relatively less extensive. Ricci *et al*.[23] were the first to fully implement the second-round Dilithium signature algorithm using VHDL, effectively improving operational efficiency through parallel design. Land *et al*.[24] proposed an efficient and compact pure hardware design for the third-round Dilithium algorithm on the Xilinx Artix-7 platform, supporting all three parameter sets. As mentioned earlier in the context of co-design, Wang *et al*.[25] proposed a specialized implementation scheme for the Dilithium algorithm on the Zynq-7000 platform. Considering that all core computational processes of the signature algorithm in their design were implemented as hardware modules, with the software part only handling packing/unpacking operations and input/output enable signals, it can also be regarded as a pure hardware implementation. Since the hardware implementation covers the entire signature algorithm flow, the final performance was significantly improved compared to co-design schemes. Zhao *et al*.[26] proposed a novel compact and high-performance hardware architecture for the third-round Dilithium algorithm. They first designed several targeted functional modules and proposed an efficient pipelined operational mode. Compared to existing pure hardware designs, their work achieved optimal results in both operational efficiency and resource consumption.

Furthermore, considering that polynomial multiplication is complex and time-consuming in lattice-based cryptography, designing dedicated polynomial arithmetic acceleration engines based on FPGA platforms is also an important research direction in hardware implementation. Among these, Mert *et al*.[34–36] proposed a series of designs for dedicated NTT modules. Their core idea is to ensure computational correctness by designing memory access logic without data conflicts, thereby enhancing the parallelism of NTT operations. Zhao *et al*.[37] conducted an in-depth study of the operand variation patterns in butterfly operations to determine the upper bit-width limits for all registers in the datapath corresponding to the modulus q, thus designing a compact and low-latency butterfly unit (BFU). Based on this, they designed and compared three NTT modules with different parallelization modes. The results show that compared to the single-structure module, the dual-parallel and quad-parallel structures increase computational efficiency by two and four times, respectively, while hardware resources increase by only 1.45 and 2.58 times, primarily due to resource reuse during instantiation. This provides a valuable reference for designing high-performance arithmetic modules. Xing *et al.*[38] proposed a new design approach for BFUs, whose core idea can be summarized as abstracting various types of operations in lattice-based schemes into a series of combinations and permutations of basic operations and then homogeneously designing these operations into a composite functional module at the hardware level. This design pattern can effectively improve hardware resource utilization, and a similar design philosophy is also reflected in the work of Zhao *et al.*[30] and Hu *et al.*[39]. It should be noted that this highly homogeneous design approach might increase the stage delay in the pipeline, potentially affecting the maximum frequency of the entire module. Therefore, the design architecture should be comprehensively considered based on specific application requirements. Besides the classical iterative polynomial arithmetic modules, another design architecture exists for high-performance applications: the pipelined design. Its core idea is to hardwire the operand matching rules of the multi-stage NTT (or similar algorithm) computation process using hardware resources. This increases the parallelism of the output ports while instantiating the entire datapath based on the computational stage depth, ultimately forming a two-dimensional arrangement paradigm of BFUs. This design concept was first embodied in the hardware architecture for the FFT algorithm proposed by Mook *et al.*[40] and was further specifically optimized and extended for the core computations of the Dilithium algorithm in the aforementioned work of Zhao *et al.*[26, 30]. This architecture only requires continuously feeding operands into the input of the arithmetic unit to obtain the final result, thus offering high throughput and being particularly suitable for application scenarios involving large data volumes, as well as single and repetitive operations.

With the formal publication of the NIST standards, research on the migration and application of PQC is entering a critical stage. From the perspective of practical deployment and engineering applications, pure hardware design schemes offer the following advantages compared to the other two implementation approaches.

(1) Significant performance improvement. Dedicated circuit structures for specific algorithms are typically designed within independent hardware resource spaces, utilizing timing state machines to ensure correct functional implementation without needing to invoke software resources for logic control. This means the design focus is more on the characteristics of the algorithm itself rather than on interaction rules between multiple platforms, especially when optimizing for a single algorithm where the generality advantage of software-hardware co-design schemes is less pronounced. Furthermore, hardware design is inherently a bit-level implementation, enabling computational parallelism from both the functional module and overall architecture dimensions. Due to these two characteristics, pure hardware designs typically achieve higher performance improvements.

(2) More mature architecture. After decades of research and development, hardware design concepts have yielded abundant scientific achievements and tool support in both academia and industry. Among them, the register-transfer level (RTL) design method has become mainstream[41]. Specifically, a hardware system can be abstracted into two parts: the datapath and the control network, composed of different hardware units. The datapath

typically involves medium-scale components like adders, multipliers, registers, and multiplexers, primarily responsible for operand computation and storage. The control network corresponds to the timing state machine design, whose core is a complex hardware structure composed of basic logic units like counters, selectors, flip-flops (FFs), and control signals. Designers can directly or indirectly specify circuit behavior each cycle, thereby controlling the operating mode of the datapath and realizing the required computational flow. Additionally, the fundamental RTL design guidelines include the speed-area trade-off, pipelining, and ping-pong operation. Specifically, the first is generally considered an overarching design principle, classifying hardware designs into high-performance and lightweight solutions by clarifying the priority differences among the four metrics: latency, clock cycle, area, and power consumption. This facilitates proposing more suitable hardware design ideas based on algorithm structure characteristics and application scenario requirements and is conducive to further practical deployment and application. Pipelining, at the micro level, involves inserting registers into a combinational logic circuit to reduce the delay of the longest path. If the inserted registers are viewed as a series of simple arithmetic units, the pipelining concept can be extended to the macro level to analyze and implement more complex functional modules, as shown in Section 3.2 of this paper. Ping-pong operation uses FIFO or register resources for data buffering, typically used for data alignment in pipeline structures or data normalization at the input/output ends of different modules.

(3) Controllable industrialization costs. Chinese and international PQC standardization schemes still have room for further optimization and development, and a new public key cryptosystem has not yet been fully established. Therefore, the current migration process towards quantum-resistant security is in a critical transitional phase. This implies that current software and hardware implementations of PQC algorithms must consider not only comprehensive performance but also practical deployment costs, preparing the technical foundation for future specialized cryptographic chips based on PQC algorithms. As mentioned, FPGA chips, after decades of development, possess mature technical systems and a complete industrial ecosystem. Combined with existing cryptographic board design patterns, they enable rapid and low-cost product development and technological iteration. More importantly, existing industrially proven pure hardware design achievements can be integrated as IP cores onto bus structures, efficiently and smoothly building design schemes for specialized cryptographic chips. This industrial advantage, not possessed by other design approaches, is highly suitable for the low-cost engineering characteristics of the transitional phase.

Based on the above analysis, research on the pure hardware design and implementation of the Dilithium algorithm holds significant value. Furthermore, considering the design characteristics of high-dimensional polynomial vectors in signature operations, further improving the algorithm's computational efficiency should be given higher priority during the design process. Existing pure hardware implementation schemes have room for further optimization: Soni *et al*.[22] completed the hardware design of the signature algorithm based on the HLS method, and Ricci *et al*.[23] directly instantiated the algorithm flow using a large amount of hardware resources; neither work proposed superior hardware optimization techniques. Land *et al*.[24] used DSP units to optimize the design of the polynomial arithmetic module, effectively improving the operational efficiency of the signature algorithm, but the parallelism and resource reuse rate of the overall hardware architecture were low. Wang *et al*.[25, 32] implemented two different quad-parallel polynomial arithmetic modules based on different technical pathways, improving computational efficiency while also saving DSP resource consumption to some extent.

However, for the signature algorithm, the iterative NTT structure struggles to further enhance the multiplication efficiency of high-dimensional polynomial vectors. In contrast, Zhao *et al.*[26] designed a pipelined polynomial arithmetic module, using four BFUs to handle adjacent stage computations in the NTT process, supplemented by appropriate timing state machine design, effectively improving the overall operational efficiency of the signature algorithm. However, their designed arithmetic module only inputs and outputs a single polynomial coefficient per clock cycle, requiring $256 \times k + 296$ clock cycles to complete k NTT operations, resulting in a low proportion of effective computation time. Additionally, in the literature, a cascaded design for the Keccak core was adopted; while this can double the efficiency of the SHA3 function computation, the overly complex combinational logic limits the maximum clock frequency. Based on these two points, the work of Zhao *et al.*[26] has potential for further improvement in terms of computational efficiency.

2 Preliminary Knowledge

This section provides the preliminary knowledge relevant to the specific work in this paper. It first presents the basic definitions of lattice hard problems, followed by a detailed analysis of the core computational process of the Dilithium algorithm, establishing a theoretical foundation for the subsequent hardware optimization design.

2.1 Basic definitions and notation

First, the basic definition of a polynomial ring is given. Let n and q be positive integers, where n is a power of 2, and $\mathbb{Z}$ and $\mathbb{R}$ represent the sets of integers and real numbers, respectively. Define the set $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z} = \{0, 1, \ldots, q-1\}$ and the set $\mathbb{R}_q = \mathbb{R}/q$, then $\mathcal{R} = \mathbb{Z}[x]/(x^n+1)$ is the polynomial ring, where elements of $\mathcal{R}$ are integer-coefficient polynomials of n-degree. Define $\mathcal{R}_q = \mathbb{Z}_q/(x^n+1)$, where the polynomial coefficients are in $\mathbb{Z}_q$. A polynomial f in $\mathcal{R}_q$ can be represented as $a = \sum_{i=0}^{n-1} a_i x^i$, where $a_i \in \mathbb{Z}_q$, and the polynomial can be represented in vector form as $a = (a_0, a_1, \ldots, a_{n-1})$. Furthermore, vectors on the ring $\mathcal{R}/\mathcal{R}_q$ are denoted by $\boldsymbol{a}$, and matrices are denoted by $\boldsymbol{A}$. Let $\boldsymbol{a}^{\mathrm{T}}$ and $\boldsymbol{A}^{\mathrm{T}}$ denote their transposes, respectively. Based on this, for an element $\boldsymbol{a} = \sum_{i=0}^{n-1} w_i x^i \in \mathcal{R}$, its infinity norm l_∞ can be defined as $\|\boldsymbol{a}\|_\infty := \max\|\boldsymbol{a}_i\|_\infty$. Similarly, the infinity norm of a vector $\boldsymbol{a} = [\boldsymbol{a}_i]_j \in \mathcal{R}^k$ can be defined as $\|\boldsymbol{a}\|_\infty := \max\|\boldsymbol{a}_i\|_\infty$. Additionally, a dedicated operator $S_\eta \subseteq \mathcal{R}$ is defined to denote the set of elements satisfying $\|\boldsymbol{w}\|_\infty \leq \eta$.

2.2 Definitions of MLWE and MSIS

The LWE problem is one of the underlying mathematical principles for constructing lattice-based cryptographic schemes. By defining the distribution $(\boldsymbol{a}, b) = \boldsymbol{a}^{\mathrm{T}} s + e$, with vector $\boldsymbol{a} \leftarrow \mathbb{Z}_q^n$ and noise $e \leftarrow \psi_\alpha$, we let the secret s follow a uniform distribution over $\mathbb{Z}_q$. Then, the computational LWE problem is, given the distribution $(\boldsymbol{a}, b)$, to compute the secret s; the decisional LWE problem is to distinguish whether a given $(\boldsymbol{a}, b)$ is from a uniform distribution over $\mathbb{Z}_q^n \times \mathbb{Z}_q$. Because the classical LWE problem requires large sampling spaces and computational overhead, it is not directly suitable for building cryptographic schemes. RLWE introduces an additional polynomial ring and utilizes negative cyclic shift operations to generate matrix elements, effectively reducing sampling difficulty. However, its more stable algebraic structure also somewhat reduces security. Consequently, further research proposed the MLWE problem based on module lattices. We define the operation $\boldsymbol{b} := \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e}$, with matrix $\boldsymbol{A} \leftarrow \mathcal{R}_q^{k \times l}$, parameter $k = \mathrm{poly}(\lambda)$, and vector $(\boldsymbol{s}, \boldsymbol{e}) \leftarrow S_\eta^l \times S_\eta^k$. It can be seen that the MLWE problem is composed of multiple RLWE instances, balancing the security of LWE and the computational efficiency of RLWE. Therefore, most lattice-based cryptographic schemes choose to build upon the MLWE problem.

Given a matrix $\boldsymbol{A} \leftarrow \mathcal{R}_q^{k\times l}$, where $k = \mathrm{poly}(\lambda)$, the MSIS problem, parameterized by $\beta > 0$, is to find a non-zero short preimage $\vec{\boldsymbol{x}} \in \mathcal{R}_q^l$ in the lattice defined by $\boldsymbol{A}$ that satisfies $\boldsymbol{A}\vec{\boldsymbol{x}} = \mathbf{0}$ and $\|\vec{\boldsymbol{x}}\| \leq \beta$. According to the MSIS assumption under the l_p ($p \in [1, \infty]$) norm, with appropriately chosen parameters, no polynomial-time algorithm can find the correct preimage $\vec{\boldsymbol{x}}$ with non-negligible probability. The security of the lattice digital signature algorithm Dilithium is based simultaneously on the MLWE problem and the MSIS problem, with particular focus on the MSIS assumption under the infinity norm.

2.3 Introduction to CRYSTALS-Dilithium algorithm

CRYSTALS-Dilithium consists of three probabilistic polynomial-time algorithms: KeyGen, Sign, and Verify. Among them, $\mathrm{KeyGen}(1^\lambda)$ is the key generation algorithm, which takes the security parameter 1^λ as input and outputs a public-private key pair (pk, sk). $\mathrm{Sign}(sk, M)$ is the signing algorithm, which takes the private key sk and a message M to be signed as input and then executes a looped signing process until it produces a signature σ satisfying the security boundary conditions. $\mathrm{Verify}(pk, M, \sigma)$ is the signature verification algorithm, which takes the public key pk, the message M to be verified, and its signature σ as input and outputs $b \in \{0, 1\}$ to determine whether it is a valid signature.

As shown in Algorithm 1, the core computational processes in the Dilithium algorithm include polynomial multiplications over the ring, such as $\hat{\boldsymbol{A}} \circ \hat{\boldsymbol{y}}$, $\hat{c} \circ \hat{\boldsymbol{s}}_1$, $\hat{c} \circ \hat{\boldsymbol{s}}_2$, and $\hat{c} \cdot \hat{\boldsymbol{t}}_0$. These are typically accelerated using the NTT technique to improve computational efficiency. Because the ring parameters (n, q) chosen for Dilithium satisfy the formula $q \equiv 1 \pmod{2n}$, the classical NTT can be used to accelerate the polynomial operations. In this case, there exist 512 primitive unit roots in $\mathbb{Z}_q^*$ satisfying the formula $x^n + 1 \cong (x-\alpha)(x-\alpha^3)(x-\alpha^{511})$. According to the Chinese remainder theorem, each polynomial $\boldsymbol{a} \in \mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$ can be uniquely represented as $[a(\alpha), a(\alpha^3), \ldots, a(\alpha^{2n-1})]^{\mathrm{T}} \in \mathbb{P}_q^n$. Therefore, polynomial multiplication can be decomposed into sub-polynomial multiplications and ultimately broken down into coefficient-wise point multiplications using an iterative approach. The NTT process for 256-dimensional polynomial coefficients involves eight levels of computation, thereby reducing the complexity of polynomial multiplication over the ring from $\mathrm{O}(n^2)$ to $\mathrm{O}(n \log n)$. Additionally, polynomial coefficient sampling is also a crucial and relatively time-consuming process in the signature algorithm. The Dilithium algorithm uses two types of XOF instances: the processes ExpandS, ExpandMask, and SampleInBall correspond to the SHAKE-256 instance, while the process ExpandA corresponds to the SHAKE-128 instance. Among these, the two most time-consuming processes are $H(\rho \parallel \boldsymbol{t}_1)$ and the sampling of matrix $\boldsymbol{A}$ (ExpandA). When designing the hardware implementation, the optimized realization of these core computational steps requires primary consideration.

3 Hardware Implementation Scheme for the Dilithium Algorithm

Analyzing Algorithm 1 from a hardware implementation perspective leads to the following conclusions: firstly, the Dilithium algorithm involves a large number of high-dimensional polynomial vectors and matrices, which correspond to extensive polynomial multiplication operations. Secondly, the signature algorithm encompasses numerous sub-operation types, primarily including three core sampling algorithms and over ten auxiliary operations. In view of these two critical computational bottlenecks, the hardware implementation requires the targeted design of dedicated functional modules. Simultaneously, it is necessary to reasonably plan the constraints of various factors such as data scheduling, resource reuse, and clock cycles, thereby designing and implementing efficient timing state control logic to further enhance the overall signature operation efficiency from the top level. Based on this analysis, the top-level architecture proposed in this paper consists of four main components.

Algorithm 1. Core computational processes of Dilthium algorithm.

Function KeyGen()
$(\rho, \rho', K) \in \{0,1\}^{256} \times \{0,1\}^{512} \times \{0,1\}^{256} \Leftarrow H(\zeta \| k \| l)$ // $\zeta \Leftarrow \{0,1\}^{256}$
$(\boldsymbol{s}_1, \boldsymbol{s}_2) \in S_\eta^l \times S_\eta^\kappa \Leftarrow \mathrm{ExpandS}(\rho')$
$\hat{\boldsymbol{A}} \in R_q^{k\times l} \Leftarrow \mathrm{ExpandA}(\rho)$
$\boldsymbol{t} \Leftarrow \mathrm{NTT}^{-1}(\hat{\boldsymbol{A}} \circ \hat{\boldsymbol{s}}_1) + \boldsymbol{s}_2$ // $\hat{\boldsymbol{s}}_1 = \mathrm{NTT}(\boldsymbol{s}_1)$
$(\boldsymbol{t}_1, \boldsymbol{t}_0) \Leftarrow \mathrm{Power2Round}_q(\boldsymbol{t}, d)$
$tr \in \{0,1\}^{256} \Leftarrow H(\rho \parallel \boldsymbol{t}_1)$
return $pk = (\rho, \boldsymbol{t}_1), sk = (\rho, K, tr, \boldsymbol{s}_1, \boldsymbol{s}_2, \boldsymbol{t}_0)$

Function Sign(sk, M)
$\hat{\boldsymbol{s}}_1 = \mathrm{NTT}(\boldsymbol{s}_1), \hat{\boldsymbol{s}}_2 = \mathrm{NTT}(\boldsymbol{s}_2), \hat{\boldsymbol{t}}_0 = \mathrm{NTT}(\boldsymbol{t}_0)$
$\hat{\boldsymbol{A}} \in R_q^{k\times l} \Leftarrow \mathrm{ExpandA}(\rho)$
$\mu \in \{0,1\}^{512} \Leftarrow H(tr \parallel M), \rho' \in \{0,1\}^{512} \Leftarrow H(K \parallel rnd \parallel \mu)$
$\kappa \Leftarrow 0,\ (\boldsymbol{z}, \boldsymbol{h}) \Leftarrow \perp$
while $(\boldsymbol{z}, \boldsymbol{h}) = \perp$ **do**
  $\boldsymbol{y} \in S_{\gamma_1}^{\iota} \Leftarrow \mathrm{ExpandMask}(\rho', \kappa)$
  $\boldsymbol{w} \Leftarrow \mathrm{NTT}^{-1}(\hat{\boldsymbol{A}} \circ \hat{\boldsymbol{y}})$ // $\hat{\boldsymbol{y}} = \mathrm{NTT}(\boldsymbol{y})$
  $\boldsymbol{w}_1 \Leftarrow \mathrm{HighBits}(\boldsymbol{w})$
  $\tilde{c} \in \{0,1\}^{256} \Leftarrow H(\mu \parallel \boldsymbol{w}_1)$
  $c \in R_q \Leftarrow \mathrm{SampleInBall}(\tilde{c})$
  $\hat{c} \Leftarrow \mathrm{NTT}(c)$
  $\boldsymbol{z} \Leftarrow \boldsymbol{y} + \langle c\boldsymbol{s}_1 \rangle$ // $\langle c\boldsymbol{s}_1 \rangle = \mathrm{NTT}^{-1}(\hat{c} \circ \hat{\boldsymbol{s}}_1)$
  $\boldsymbol{r}_0 \Leftarrow \mathrm{LowBits}(\boldsymbol{w} - \langle c\boldsymbol{s}_2 \rangle)$ // $\langle c\boldsymbol{s}_2 \rangle = \mathrm{NTT}^{-1}(\hat{c} \circ \hat{\boldsymbol{s}}_2)$
  if $\|\boldsymbol{z}\|_\infty \geq \gamma_1 - \beta$ **and** $\|\boldsymbol{r}_0\|_\infty \geq \gamma_2 - \beta$ **then**
    $(\boldsymbol{z}, \boldsymbol{h}) \Leftarrow \perp$
  else
    $\langle c\boldsymbol{t}_0 \rangle = \mathrm{NTT}^{-1}(\hat{c} \cdot \hat{\boldsymbol{t}}_0)$
    $\boldsymbol{h} \Leftarrow \mathrm{MakeHint}(-\langle c\boldsymbol{t}_0 \rangle, \boldsymbol{w} - \langle c\boldsymbol{s}_2 \rangle + \langle c\boldsymbol{s}_1 \rangle)$
    if $\|c\boldsymbol{t}_0\|_\infty \geq \gamma_2$ **or** the number of 1 in $\boldsymbol{h}$ is greater than ω **then**
      $(\boldsymbol{z}, \boldsymbol{h}) \Leftarrow \perp$
    end if
  end if
  $\kappa \Leftarrow \kappa + l$
end while
return $\sigma = (\tilde{c}, \boldsymbol{z}, \boldsymbol{h})$

Function Verify(pk, M', σ)
$\hat{\boldsymbol{A}} \in R_q^{k\times l} \Leftarrow \mathrm{ExpandA}(\rho)$
$tr \Leftarrow H(pk, 64)$
$\mu \in \{0,1\}^{512} \Leftarrow H(tr \parallel M')$
$c \in R_q \Leftarrow \mathrm{SampleInBall}(\tilde{c})$
$\boldsymbol{w}_1' \Leftarrow \mathrm{UseHint}_q(\boldsymbol{h}, \mathrm{NTT}^{-1}(\hat{\boldsymbol{A}} \circ \mathrm{NTT}(\boldsymbol{z}) - \mathrm{NTT}(c) \circ \mathrm{NTT}(\boldsymbol{t}_1 \circ 2^d)), 2\gamma_2)$
return $\|\boldsymbol{z}\|_\infty < \gamma_1 - \beta$ **and** $\tilde{c} = H(\mu \parallel \boldsymbol{w}_1')$

(1) A polynomial arithmetic module based on a multipath delay commutator (MDC) architecture that supports multiple parameter sets. This module employs an eight-stage pipelined BFU array to meet the requirements for various types of coefficient operations, including NTT, inverse NTT (INTT), accumulated modular addition, and modular multiplication. More importantly, tailored to the characteristics of high-dimensional vectors/matrices in the Dilithium algorithm, the pipelined design architecture can continuously perform arithmetic operations, significantly increasing the data throughput of the entire module.

(2) Targeting the specific characteristics of the input and output data for the core sampling

operations and hash operations in the Dilithium algorithm, two distinct sampling modules are designed. The functional module dedicated to sampling operations is reasonably optimized, supplemented by sub-modules for data encoding and decoding, responsible for data formatting and alignment. This approach achieves a doubling of sampling operation efficiency with lower resource consumption.

(3) By using the parameter set corresponding to Security Level 5 as the benchmark, a unified storage unit array is designed. The combination rules and storage depth for polynomial coefficients were determined by the chip select signals input from the top-level module. Based on a reconfigurable design philosophy, this meets the storage requirements for polynomial vectors/matrices across all parameter sets with higher resource utilization.

(4) By analyzing the sub-operation types within the core computational steps of the signing process, they are decomposed and recombined according to a fine-grained reorganization design strategy. This effectively increases the total number of polynomial coefficients processed per input to the arithmetic array, thereby logically reducing the ineffective latency per individual coefficient from a hardware perspective. Secondly, the parallelism of both the top-level sampling modules and the arithmetic array modules is maximized as much as possible. This further enhances the utilization of clock cycle resources within the overall hardware architecture and improves the signature operation efficiency, provided no data conflicts are introduced.

3.1 Polynomial arithmetic module design

As shown in Figure 1, the polynomial arithmetic module is designed and implemented based on a delay-steering architecture. Its core part consists of an arithmetic array composed of radix-2 BFUs, supplemented by 40 shift registers and 20 multiplex selectors (delay register and multiplex selector, DM). Therefore, it can also be referred to as a systolic array arithmetic unit. As mentioned earlier, the NTT process in the Dilithium algorithm is a standard eight-level iterative computation, which corresponds one-to-one with the arithmetic array of the hardware

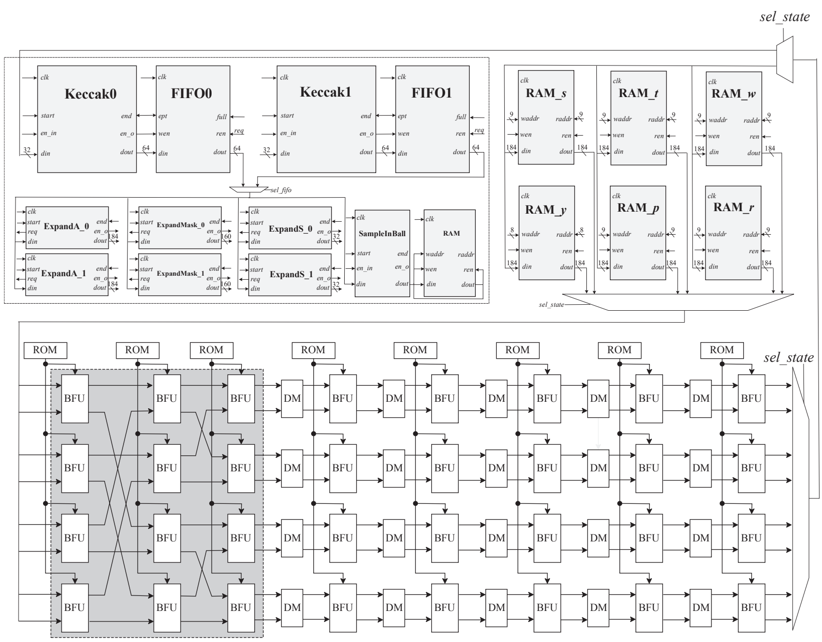

Figure 1 Top-level architecture of signature algorithm hardware design

module. Simultaneously, adjacent arrays, aided by registers and multiplex selectors, achieve correct operand matching, effectively hardwiring each stage of the NTT/INTT computation using hardware resources. Consequently, the input of the arithmetic module can be continuously filled with data, capable of accepting up to eight operands per clock cycle. Under this multi-stage pipelined design structure, it enables continuous computation of high-dimensional polynomial vectors/matrices. Based on the hardware structure of the aforementioned arithmetic module, the total time consumption consists of two parts: the clock cycles required to fill the input with data, which is proportional to the total amount of data involved in a single computation; and the latency clock cycles of the computation process itself, which is related to the internal data path and computational depth of the module within the arithmetic module. For a specific type of operation, the latency is determined by the number of BFUs and their associated delay-steering units involved in the coefficient operation, making this latency a fixed value. Given the parameter $n = 256$, a complete set of polynomial coefficients can be input in 32 clock cycles. If a certain type of operation is performed on K sets of polynomial coefficients with a latency of D cycles, the clock cycle utilization can be calculated as $32K/(32K + D)$. Analysis readily shows that as the number of polynomials K involved in the computation increases, the effective time utilization gradually converges towards 1. At the hardware level, this process can be interpreted as when the number of polynomials involved in the computation increases, the fixed latency is effectively amortized over a larger data volume, thereby naturally reducing the latency per computed data item and simultaneously increasing the module's throughput. It is important to note that the hardware structural characteristics of the polynomial arithmetic module described above, while improving the efficiency of specific polynomial operations, also substantially influence the final selection of the top-level hardware design from a macro perspective. This point will be detailed in the timing control logic section (Section 3.7).

3.2 Compact BFU design and implementation

Figure 2 shows the hardware architecture of the core component of the polynomial arithmetic module, namely the butterfly arithmetic unit. It adopts a dual-path design structure accommodating both CT and GS butterfly operations. It incorporates two levels of adders, one multiplier, and one reduction unit and includes a dedicated register with a delay depth of seven cycles, used to normalize the timing differences of the dual-input data streams at the hardware level. The designed low-latency and compact butterfly arithmetic unit can meet the requirements for various types of operations, with the specific operation instance determined

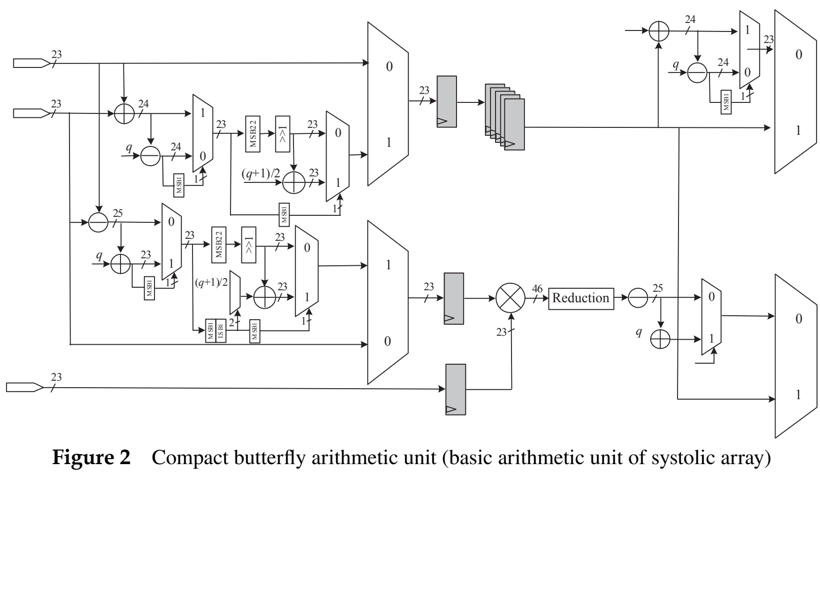

Figure 2 Compact butterfly arithmetic unit (basic arithmetic unit of systolic array)

by the chip select signal. Specifically, when the chip select signal is set to 1, it corresponds to the INTT operation instance. The input ports, from top to bottom, are $(coff_0, coff_1, tw)$. The values of the two output ports are $(coff_0 + coff_1)/2 \pmod q$ and $(coff_0 - coff_1)w/2 \pmod q$, respectively. When the chip select signal is set to 0, it can correspond to three operation instances: modular addition, modular multiplication, and forward NTT operation. It is important to note that polynomial modular addition and modular multiplication operations only utilize the eight BFUs in the first two columns of the systolic array. Therefore, disregarding hardware latency, completing one set of these operations consumes 32 clock cycles, which is the same as the input filling time for the NTT operation. This design architecture provides the hardware foundation for subsequent top-level timing optimization.

During computation, it is essential to ensure the final output is reduced to the range $[0, q-1]$. Common reduction algorithms include the Montgomery algorithm and the Barrett algorithm[42, 43]. In the hardware design proposed in this paper, the Barrett reduction algorithm offers two significant advantages by comparison: firstly, the result of the Montgomery algorithm lies in the interval $(-q, q)$, which implies an extra sign bit is needed for each coefficient at the hardware level. In contrast, the result of the Barrett algorithm lies in the interval $(0, q)$, with no negative numbers, making it more suitable for uniformly using unsigned numbers in the hardware design. Secondly, the Barrett reduction algorithm's result includes both the reduced result res and the quotient value quo. Considering the presence of the operation $\mathrm{Decompose}_q(r, \alpha)$ in the Dilithium algorithm, whose core function is to calculate the high and low bits of r, specifically, $r_0 := r \bmod {}^{\pm}\alpha$ and $r_1 := (r - r_0)/\alpha$, the results correspond precisely to the residue res and the quotient quo, respectively. Based on these two analyses, we chose to design the reduction module based on the Barrett algorithm. As shown in Figure 3, according to the layout structure of the delay registers, the reduction unit features a three-stage pipeline. Considering the input and output delays, one complete reduction process requires five clock cycles. This is also the main reason why the upper data path of the BFU requires delay registers for timing normalization. In the first stage of computation, the modulus q for the Dilithium algorithm is decomposed based on the minimum Hamming weight principle, resulting in $2^{23} - 2^{13} + 1$. On this basis, the two multiplication operations in the Barrett algorithm are replaced using addition and subtraction, thereby saving DSP resources. The core of the second stage is to correct the preliminary reduction result. The correction is based on the first three bits of the reduction result, corresponding to eight possible cases, and the corrected result resides in a larger range $[0, 2^{24}-1]$. In the final third stage, data within the range $[2^{23}, 2^{24}-1]$ undergoes

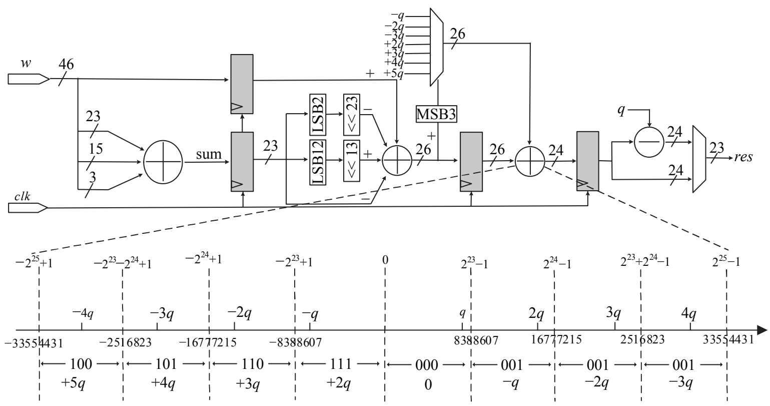

Figure 3 Improved reduction arithmetic hardware design based on the Barrett algorithm (including correction logic)

another modular subtraction operation to obtain the final correct reduction result. As mentioned earlier, the functional module corresponding to the operation $\mathrm{Decompose}_q(r, \alpha)$ is isomorphic to the three-stage hardware architecture shown in Figure 3. The difference lies in the presence of two modulus values here: 190,466 and 523,776. Their respective decomposed forms are $2^{18} - 2^{16} - 2^{13} + 2^{11}$ and $2^{19} - 2^{9}$. Furthermore, to save hardware resources, a division-free design approach is adopted when implementing the reduction module corresponding to the modulus 8,380,417.

3.3 Multifunctional permutation network design

As analyzed previously, the input of the systolic array arithmetic unit is order-sensitive, meaning the correctness of the computation results strongly depends on the matching of input data. However, the initial order of the sampled polynomial coefficients, the input order for computation, and the output order after computation are not entirely consistent. Therefore, to ensure the correctness of polynomial operations, a reordering operation on the data matching relationship at the input or output of the systolic array unit is required. We define the order of the sampled polynomial coefficients as the initial order. Based on the characteristics of the core polynomial operation types, a multifunctional permutation network containing four operation modes is designed and implemented, as shown in Figure 4. The subscript of coefficient x uses hexadecimal notation to represent the coefficient index in the polynomial. Specifically, the first permutation mode converts the initial order into the input order for the NTT operation, as shown in Figure 4(a). Each correct set of input coefficients is distributed across eight different address positions; thus, one correct set of operands can be obtained every eight clock cycles. The second permutation mode converts the output order of the NTT operation back to the initial order and performs data rearrangement before the start of the multiply-accumulate operation, as shown in Figure 4(b). The first 8 data points in the initial order are distributed across four NTT operation result sets; therefore, one correct set of operands can be obtained every 4 clock cycles. During the computation of vector $\boldsymbol{t}$, the output of the $\mathrm{INTT}(\boldsymbol{A}\boldsymbol{s}_1)$ operation needs to undergo modular addition with the corresponding coefficients in vector $\boldsymbol{s}_2$. Considering that the polynomial coefficients in vector $\boldsymbol{t}$ need to be in the initial order for the subsequent $H(\rho \parallel \boldsymbol{t}_1)$ operation, we chose to convert the output order of the INTT operation to the initial order, defining this as the third permutation mode, as shown in Figure 4(c). Furthermore, since the results of PWMs $(c\boldsymbol{s}_1, c\boldsymbol{s}_2, c\boldsymbol{t}_0)$ reside in the NTT domain, they need to be converted from the NTT output order

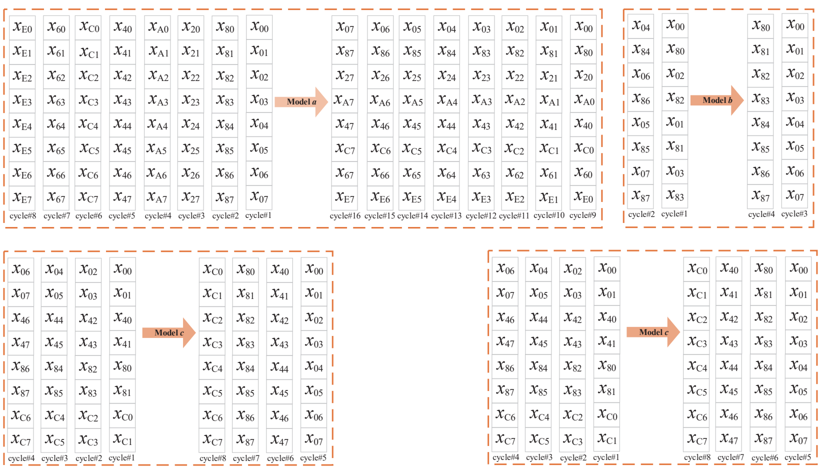

Figure 4 Permutation network operation mode

to the INTT input order before participating in the next operation. This process is defined as the fourth permutation mode, as shown in Figure 4(d).

Since the first permutation mode requires the highest number of delay cycles, the design of the multifunctional permutation network was based on this case. The required hardware resources could simultaneously accommodate the other three cases. Based on the above analysis, the reordering network in this paper primarily consists of two sets of registers. Each set contains eight long registers with a bit width of 184 bits, used for data buffering at the input and output of the systolic array unit. Simultaneously, the buffered operands are sequentially subjected to delay alignment operations according to their register index values. After a fixed number of clock cycles, this ensures the generation of eight coefficient sets that meet the computational requirements. By using these two register sets alternately to perform the above operations, a pipelined operation mode for both polynomial computation and the corresponding reordering operation is ultimately achieved.

3.4 Dual-core hash module design

Generating random bit streams is a prerequisite operation for sampling algorithms, and its performance directly impacts the efficiency of subsequent sampling modules. As mentioned earlier, the Dilithium algorithm uses two computational instances, SHAKE-128 and SHAKE-256, to generate random bit streams. The three most critical sampling operations among these are ExpandS, ExpandMask, and ExpandA, which correspond to the polynomial coefficient sampling processes for vector $\boldsymbol{s}$, vector $\boldsymbol{y}$, and matrix $\boldsymbol{A}$, respectively. The remaining computational processes are primarily responsible for generating data digests or random seeds. We conducted an in-depth study and summarized the characteristics of these two types of operations, arriving at the following conclusion: the input parameter length for the sampling algorithms ExpandS, ExpandA, and ExpandMask is shorter than the block length of the XOF computational instance's input data. This implies that during coefficient sampling, the hash module only requires input data at the initial stage. In subsequent iterative steps, there is no operation that involves XOR-ing the result from the previous stage with new input data. This provides a fundamental condition for optimizing the hash module. Based on the above analysis, we comprehensively considere both hardware resources and algorithmic efficiency, designing two different dedicated hash modules: a general-purpose hash module and a sampling-optimized hash module, as shown in Figure 5.

The hardware architecture of the general-purpose hash module mainly consists of three long registers, each 1,600 bits in length. The input and output port widths are set to 64 bits, meeting the computational requirements for all XOF instances in the signature algorithm. In contrast, the hardware architecture of the sampling-optimized hash module contains only two long registers and is specifically used for the coefficient sampling operations in polynomial vector $\boldsymbol{s}$, vector $\boldsymbol{y}$, and matrix $\boldsymbol{A}$. The difference between them lies in the fact that in the general-purpose hash module, after every 24 rounds of the permutation function, the result is output and simultaneously stored in the third long register. It then receives an XOR operation with the input data stored in the first long register to generate the initial state data for the next round of iterative computation. This computational process corresponds to the $H()$ function. Since the input data bit width for the sampling algorithms ExpandS, ExpandA, and ExpandMask is much smaller than the maximum input length for a single XOF instance operation, the sampling-optimized hash module can perform the required functions without needing the third long register, thereby saving significant hardware resources in implementation. Furthermore, the two computational modules are designed based on a clock-sharing concept. The core long registers correspond to different sub-operations within the XOF instance computation process and share clock cycle resources with each other. During specific testing, sampling the polynomial coefficients for two matrices $\boldsymbol{A}$ or vectors $\boldsymbol{y}$ requires only 120 clock cycles, while the corresponding data for sampling the polynomial coefficients of vectors $\boldsymbol{s}$ requires 48 clock cycles.

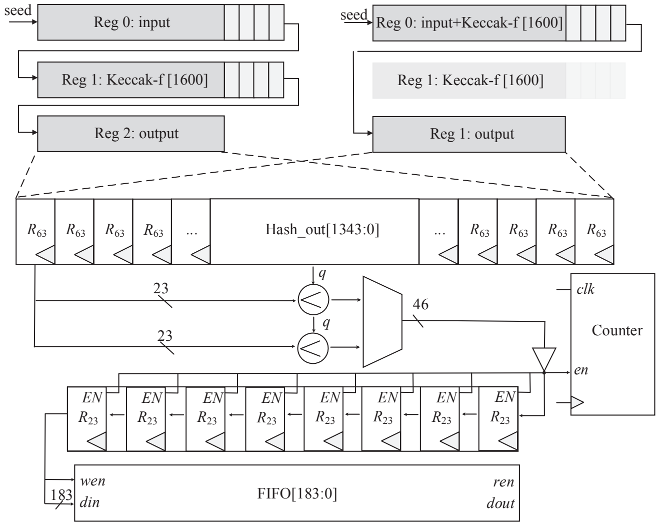

Figure 5 Hardware design for reject sampling based on dual Keccak modules

After the sampling operations are completed, the obtained polynomial coefficients require normalization at the hardware level to meet the requirements of the top-level pipeline design for both polynomial sampling and polynomial operations. We incorporated a FIFO unit with a depth of 32 and a bit width of 184 at the output of each of the two aforementioned sampling modules, used for buffering polynomial coefficients. Considering that the coefficients for every two 256-dimensional polynomials are generated synchronously, a dedicated functional control logic, centered around a single-bit indicator signal sel_fifo, is designed. This logic determines the timing for asserting the read signals of the two FIFO units, ensuring that the sampled polynomial coefficients participate in the next systolic array operations in the correct sequence. Specifically, when the number of polynomials in a vector or matrix is even, the read signals for FIFO0 and FIFO1 are asserted sequentially. The final polynomial coefficients involved in the computation come from the buffered data in the FIFO1 unit, as illustrated by vector $\boldsymbol{s}_2$. When the number of polynomials is odd, the timing logic for the FIFO read signals remains unchanged. However, the final polynomial coefficients involved in the computation come from the FIFO0 unit, and the sampled data in the FIFO1 unit is simultaneously released. Since obtaining a new set of polynomial coefficients from the sampling module takes at least 48 clock cycles, significantly longer than the time required to fully output the buffered data from a FIFO unit, timing conflicts and data overwriting are avoided. We achieve dual-parallel sampling operations with lower hardware resource consumption while maintaining a high clock frequency at the top-level hardware architecture, effectively improving the overall signature operation efficiency.

3.5 Reconfigurable storage array design

Based on the types of polynomial coefficients during the signature process, the storage space can be categorized into three cases at the hardware level:

(1) Polynomial coefficients related to public and private keys need long-term storage and participate in all operational processes; they should be allocated independent storage space.

(2) Polynomial coefficients in matrix $\boldsymbol{A}$ only participate in specific operations, and the total data volume is large; suitable timing logic should be designed based on an on-the-fly

concept, allowing the sampled coefficients to participate in computations directly without occupying additional storage space.

(3) Polynomial coefficients serving as intermediate values in the computational process should share a unified storage space with subsequent operational outputs.

The storage resource minimization design is implemented according to the above principles, as detailed in Figure 6. Considering the dependencies between the storage unit and the sampling units and arithmetic units at the hardware level and referencing the parameters of the buffer FIFO units, the storage unit bit width was also set to 184 bits, meaning a single address location can store eight polynomial coefficients. To optimize the storage space design, the characteristics of the FPGA chip IP core design need further consideration. During instantiation, 36 bits are taken as the baseline, corresponding to 0.5 BRAM units. If the eight polynomial coefficients at a single address are stored separately, it would consume at least 4 BRAM units, resulting in a storage resource utilization of 63%. We decompose a single address location into four parts of 36 bits and one part of 40 bits, totaling five segments. This allows the polynomial coefficient storage requirement to be met using only 2.5 BRAM units.

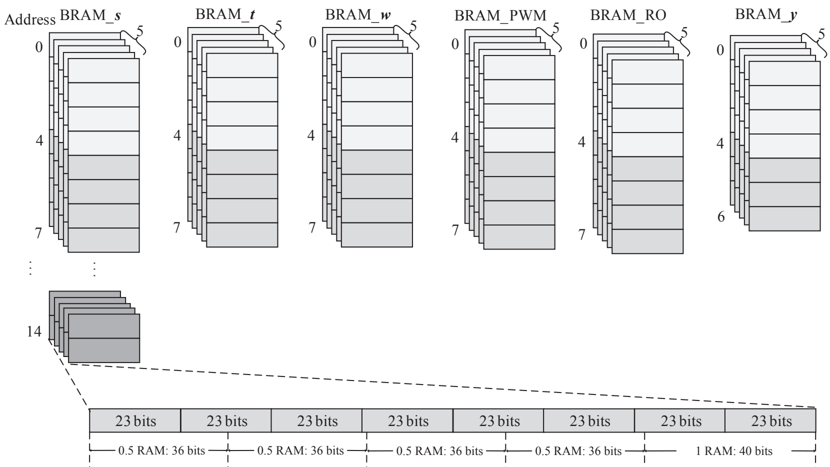

Figure 6 Reconfigurable storage unit array

Based on the above analysis, a reconfigurable storage unit array is designed based on the parameter set corresponding to security level 5, consisting of six parts: the polynomial coefficients of vector $\boldsymbol{s}_1$, vector $\boldsymbol{s}_2$, and vector $\boldsymbol{t}$ are uniformly and permanently stored in the BRAM_$\boldsymbol{s}$ and BRAM_$\boldsymbol{t}$ units shown in Figure 6, with depths set to 480 and 256, respectively, capable of storing up to 23 polynomial coefficients. The polynomial coefficients of vector $\boldsymbol{w}$ are stored in the BRAM_$\boldsymbol{w}$ unit, with a depth of 256, corresponding to storage for 8 polynomial coefficients. This space is shared with the intermediate values and output results of the operation $\boldsymbol{w}-c\boldsymbol{s}_2+c\boldsymbol{t}_0$, corresponding to the second case mentioned above. Furthermore, the polynomial coefficients of vector $\boldsymbol{y}$ exist in two forms: coefficients in the operation $\boldsymbol{Ay}$ are in the NTT domain, while coefficients in the operation $\boldsymbol{z} = \boldsymbol{y} + c\boldsymbol{s}_1$ are in the natural numbers domain. Therefore, independent storage space is also needed to store the initial values of the vector coefficients and the NTT results, corresponding to BRAM_$\boldsymbol{y}$ unit, and this space is shared with the operation result $\boldsymbol{z}$.

The BRAM_PWM unit is used to store intermediate values of multiplication operations. Since the polynomial coefficients in the three key multiplication processes $c\boldsymbol{s}_1$, $c\boldsymbol{s}_2$, and $c\boldsymbol{t}_0$ are all in the NTT domain and can directly participate in PWM, the storage space at the hardware level should correspond to the vector with the highest dimension, and its depth is set to 256. During the multiply-accumulate operation, only the first 32 address locations of the BRAM_PWM unit are used. Specifically, the multiplication and accumulation operations corresponding to $\boldsymbol{A}_{ji}\boldsymbol{y}_i + \boldsymbol{A}_{j(i-1)}\boldsymbol{y}_{i-1}$ are executed simultaneously. This process only requires storage space for one polynomial. Iterating l times yields a complete set of polynomial coefficients for vector $\boldsymbol{w}$. Since vector $\boldsymbol{w}$ has already been allocated independent storage space, the first 32 address locations in the BRAM_PWM unit can be released at this point. Considering the continuity of pipelined operations, after completing one round of computation, the data in the first 32 address locations of the BRAM_PWM unit is overwritten with zeros. Here, the first multiply-accumulate in each iteration can be interpreted as the operation $0 + \boldsymbol{A}_{j0}\boldsymbol{y}_0$. Performing this iterative process k times yields the complete output result $\boldsymbol{w} = \boldsymbol{A}\boldsymbol{y}$. Additionally, when implementing the specific functions of the permutation network corresponding to the systolic array unit described in Section 3.1, dedicated storage space, corresponding to BRAM_$\boldsymbol{y}$ unit, is required to temporarily store polynomial coefficients and complete the permutation order conversion. Its storage depth also corresponds to the highest dimension vector.

3.6 Read/write address mapping logic design

When designing the memory access logic for the BRAM array, two operational characteristics of the systolic array unit must be considered: polynomial operations require coefficient matching based on the multifunctional reordering network; the input and output of the arithmetic unit have fixed latency, meaning that, from the perspective of polynomial vectors, memory access processes and computation processes for the same set of coefficients might share clock cycles. To enhance the operational efficiency of the overall architecture while avoiding timing conflicts, we design four address mapping rules and two sets of synchronous memory access logic specifically for the two scenarios mentioned above.

The reordering network performs a directed permutation of the operand storage locations at the polynomial level, aiming to meet the computational requirements of the systolic array unit. Therefore, the functional implementation of the reordering network can be materialized as the read/write address mapping rules for polynomial coefficients within the storage unit array. Specifically, in the first reordering mode, each set of input operands is distributed across eight non-adjacent address locations. The initial read address mapping rule is (0, 16, 4, 20, 8, 24, 12, 28). Subsequent read data locations increment based on this pattern, eventually traversing the stored coefficients across all 32 address locations. In the second reordering mode, each set of operands corresponds to four different address locations. The initial write address mapping rule at the output end is (0, 8, 16, 24). At this point, the coefficients in the storage unit are arranged sequentially, facilitating synchronous read address and clock cycle counting for the subsequent multiply-accumulate operation $\boldsymbol{A}\boldsymbol{y}$. The other two reordering modes follow a similar design logic, aiming to satisfy the operational rules for the modular addition after INTT and the INTT operation after NTT. We pre-store the four dedicated read/write address mappings in a 160-bit register network. Correct read/write address information is then obtained based on the top-level chip select signal and bit left-shift operations each cycle, as shown in Table 1.

Considering the timing differences between the input and output ends of the systolic array arithmetic unit, synchronous operation of the read operation, reordering operation, and write operation can be achieved through reasonable functional design, thereby maintaining the uniformity of the overall pipelined computational structure. We design two sets of parallel read/write address generators for the storage unit array:

(1) A counter group (ctr_k, ctr_{k_v2}) for the core polynomial operations. Here, the main counter ctr_k is responsible for determining the total number of clock cycles for the input process. The auxiliary counter ctr_{k_v2}, based on the operation type and starting after a

fixed clock cycle delay relative to the main counter, generates the write addresses.

(2) Auxiliary operations, such as reordering and storage of sampled data, use the counter group $(raddr_{\mathrm{RO}}, waddr_{\mathrm{RO}})$ to generate read and write addresses, following the same design logic as the first counter group.

Figure 7 illustrates the specific synchronization rules for the simultaneous operation of these two address generators, using the core computational process $\boldsymbol{Ay}$ as an example. First, the polynomial coefficients of vector $\boldsymbol{y}$ are stored according to the first reordering mode introduced in Section 3.3. This process consumes $(l \times 32 + 4)$ clock cycles. It is evident that the PWM operation and the reordering operation temporally overlap in this phase. Based on the previous analysis, both sets of address generators need to be activated simultaneously to fulfill the requirements of these two distinct operations. Specifically, the working range of the $raddr_{\mathrm{RO}}$ counter is set to $(0, l \times 32)$, allowing it to traverse all address locations of the BRAM_$\boldsymbol{y}$ unit completely. It reads the data, performs the corresponding reordering operation, and generates the write addresses $waddr_{\mathrm{RO}}$ for the BRAM_RO unit based on the mapping rules from Table 1 and the clock delays. Moreover, after a complete polynomial for matrix $\boldsymbol{A}$ is sampled and ready, the main counter ctr_k starts and completes one cycle of 32 clock cycles, generating the read addresses for the BRAM_RO unit. Subsequently, the auxiliary counter ctr_{k_v2} starts after an 8-clock-cycle delay and generates 32 write addresses arranged in natural order, thereby completing one full PWM operation. It is important to note that the reordering operation runs continuously and starts at least 32 clock cycles before the commencement of the first PWM operation. This design prevents timing conflicts between the two processes and ensures the correctness of the computational results.

Table 1 Mapping relationship between input coefficient address bits and clock cycle offset

Operating mode	Cycle1	Cycle2	Cycle3	Cycle4	Cycle5	Cycle6	Cycle7	Cycle8
Post-sampling NTT	0/1/2/3	16/17/18/19	4/5/6/7	20/21/22/23	8/9/10/11	24/25/26/27	12/13/14/15	28/29/30/31
Post-NTT modular multiplication	0/2/1/3	4/6/5/7	8/10/9/11	12/14/13/15	16/18/17/19	20/22/21/23	24/26/25/27	28/30/29/31
Post-INTT modular addition	0/2/1/3	16/18/17/19	8/10/9/11	24/26/25/27	4/6/5/7	20/22/21/23	12/14/13/15	28/30/29/31
Post-NTT INTT	0/8/4/12	1/9/5/13	2/10/6/14	3/11/7/15	16/24/20/28	17/25/21/29	18/26/22/30	19/27/23/31

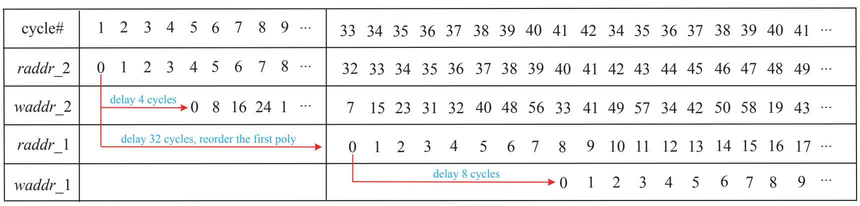

Figure 7 Synchronous memory access logic rules for multiply-accumulate operations

3.7 Segmented timing state design

Based on the design characteristics of the systolic array unit discussed in Section 3.1, it is known that, under the premise of fixed hardware latency, increasing the amount of data processed per operation can effectively improve clock cycle resource utilization. Building upon the operational features of the systolic array unit, we propose a segmented timing state control logic that use vectors rather than individual polynomials as the basic operational units and fully considers the specific process of the signature algorithm (cf. Figure 8). The core idea is to deconstruct the core computational flow of the signature algorithm at a fine granularity and, based on differences in operation types, reorganize them into multiple dedicated operation groups processed uniformly. This approach significantly increases the throughput of the core arithmetic unit, thereby enhancing the overall operational efficiency of the hardware architecture.

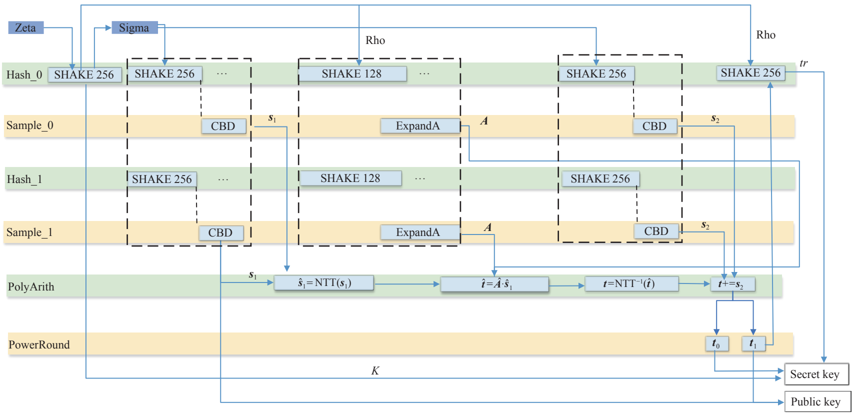

Figure 8 Timing control logic for the key generation algorithm

Applying this timing control design concept concretely to the signature process, we take the security level with the largest data volume (security level 5) as an example: first, the polynomial coefficients of the vector group $(\boldsymbol{s}_1, \boldsymbol{s}_2, \boldsymbol{t}_0)$ are uniformly fed into the systolic array unit for the forward NTT. The input phase consumes 732 clock cycles. Considering the fixed latency of 96 clock cycles, the effective time utilization rate for this operation is 88%. Second, sampling operations proceed synchronously to generate the polynomial coefficients for vector $\boldsymbol{y}$, which also undergo forward NTT using the systolic array unit. Here, the input phase consumes 224 clock cycles, resulting in an effective time utilization of 70%. This shows that the effective time utilization for vector group operations is significantly higher than that for individual vectors. Finally, a key analysis focuses on the three core operations $(\boldsymbol{y} + c\boldsymbol{s}_1, \boldsymbol{w} - c\boldsymbol{s}_2, \boldsymbol{w} - c\boldsymbol{s}_2 + c\boldsymbol{t}_0)$. Their characteristic is that they each consist of three parts: multiply-accumulate, INTT, and modular addition. By using the fine-grained decomposition and reorganization design strategy, three unified operation groups are obtained: $(c\boldsymbol{s}_1, c\boldsymbol{s}_2, c\boldsymbol{t}_0)$, INTT$(\sim)$, and $(+, -, -, +)$. The systolic array unit is used to perform PWM, INTT, and modular addition operations, corresponding to latencies of 8, 96, and 8 clock cycles, respectively. This timing design minimizes the number of activations of the core arithmetic unit and its associated fixed latency overhead. It also effectively distributes the ineffective waiting time inherent at the single-coefficient scale. Consequently, while ensuring the correctness of the pipelined operations, it greatly enhances the overall signature generation efficiency.

4 Performance Comparison

We employ the Verilog hardware description language to design and implement a unified circuit architecture capable of supporting digital signature operations for all three security levels of the Dilithium algorithm. The parameter set for the algorithm implementation is selected by adjusting the value of the chip select signal at the input port. The written Verilog code is simulated, synthesized, and place-and-routed using the Xilinx Vivado 2017.4 development suite. Practical deployment and testing are conducted on the Xilinx Artix-7 FPGA platform, specifically the xc7a200tfbg484-2 chip model. A comprehensive performance report is obtained, covering three aspects: clock cycle count, maximum clock frequency, and hardware resource consumption. The clock cycle count corresponds to the time required for the hardware circuit to generate a correct signature under optimal conditions, representing the maximum throughput rate. Hardware resources primarily focus on four common metrics: look-up table (LUT) for implementing basic logic functions, FFs related to data storage and signal delay, BRAMs for large-scale data storage and caching, and DSP slices for multiplication operations.

4.1 Hardware resource consumption analysis

Table 2 presents the hardware resource usage of each sub-functional module. The polynomial arithmetic module, with the systolic array at its core, consumes 51% of the LUT resources and 54% of the FF resources. Since the multiplier unit uses an 18-bit baseline, the PWM of 23-bit polynomial coefficients in the signature algorithm corresponds to two DSP units. Therefore, the entire systolic array module shown in Figure 1 consumes 64 DSP units in total. An IP core configuration prioritizing area saving can conserve 50% of the DSP units but simultaneously consumes more LUT resources. In practical application environments, a more suitable design approach can be selected based on specific requirements. Based on the analysis of the test platform used in this paper, the number of DSP units included in the Artix-7 series FPGA chips is sufficient to meet the multiplier resource demands of the systolic array. The parallel sampling unit occupies 32% of the LUT resources and 26% of the FF resources at the hardware level. Comparing the two modules reveals that the resource consumption of the sampling-optimized Keccak core is significantly less than that of the general-purpose Keccak core, saving 16% in LUT resources and 25% in FF resources. Ref. [26] proposed a single-core cascaded hash module design, which also doubles the sampling rate but consumes 4,577 FF resources, similar in magnitude to the general-purpose Keccak core. In contrast, the optimized sampling-optimized Keccak core consumes only 3,308 FF resources, representing approximately 28% savings in register resources. To achieve two permutation function iterations per clock cycle, the cascaded architecture requires instantiating more combinational logic. Overall, the Keccak core designed in Ref. [26] consumes 15,784 LUT resources. Compared to this, the dual-core structure proposed in this paper saves approximately 23% in hardware resources. Furthermore, the design of the storage array's unit composition rules and data access logic also fully consideres the hardware characteristics of the IP cores. Each address location in a storage unit group can accommodate eight 23-bit polynomial coefficients, occupying only 2.5 BRAM units. This approach effectively conserves storage resource consumption while meeting reconfigurability requirements.

Table 2 Hardware resource occupancy of functional modules

Module	Hardware resources			
	LUT	FF	BRAM	DSP
Keccak_v1	6,569	4,414	0	0
Keccak_v2	5,502	3,308	0	0
Expand_s×2	113/45	46/46	0	0
ExpandA×2	179/116	367/367	0	0
ExpandMask×2	336/366	264/265	0	0
Encode×2	47/22	38/38	0	0
SampleInBall	40	62	0	0
RAM_C_TAU	0	0	0.5	0
NTT_core	19,664	16,588	18	64
FIFO 0	478	203	0	0
FIFO 1	478	203	0	0
FIFO 2	1,647	154	0	0
Combine	37,943	30,268	18.5	64

4.2 Comparative analysis of core arithmetic modules

We first deploy and test the core polynomial arithmetic module, comparing it against references including both pure hardware implementations and software-hardware co-designs. The specific details are shown in Table 3. The time component in the area-time product (ATP) calculation uniformly uses the sum of the clock cycles for the NTT operation and the PWM operation. Furthermore, by using the data from this paper as the baseline, the ATP data of the compared references were scaled proportionally to more clearly illustrate the differences. Overall, the hardware module designed in this paper shows significant advantages in computational performance. According to the ATP results, it also achieves a good balance in the consumption of both LUT and DSP hardware resources. Meanwhile, the NTT, PWM, and point-wise addition (PWA) operations in Ref. [33] correspond to three different hardware units. Therefore, if calculated according to the unified rule mentioned above, the actual ATP value for the core arithmetic module designed in Ref. [33] would be higher than the data shown in Table 3. Regarding BRAM data, different design philosophies lead to different storage resource consumption. The storage resources in our work are intended for all operational processes of the entire digital signature algorithm, whereas software-hardware co-designs can also utilize storage space in the soft core, such as the L2 cache mentioned in Ref. [32]. Additionally, we use a significant number of register resources to implement the multifunctional permutation network, ensuring computational correctness at the cost of excessive FF resource consumption, a process not present in other design schemes. Therefore, for fairness, we primarily list data comparisons for LUT and DSP hardware resources, which are more closely related to the polynomial arithmetic process and focus on comparative analysis with the latest pure hardware implementation[26] and the highest-performance software-hardware co-design[30].

Table 3 Performance comparison of polynomial arithmetic modules

Related work	Testing platform	Number of units	Area source			Cycle count		Area × time	
			LUT	BRAM	DSP	NTT/INTT	PWM	LUT	DSP
[26]	Artix-7	4	1,919	2	8	256	64	0.49	0.63
[25]	Zynq-7000	1	2,386	1	8	264	260	0.99	1.02
[35]	Zynq-7000	1	6,639	3	16	265	73	1.78	1.32
[24]	Artix-7	2	524	1	17	533/536	85	0.26	2.59
[33]	Artix-7	1	799	4.5	2	1,405	269	1.06	0.82
[30]	Artix-7	32	25,674	6	64	32	32	1.31	1.00
Our work	Artix-7	32	19,664	18	64	32	32	1.00	1.00

Both this work and Ref. [30] designed high-performance systolic array units based on pipelining, effectively increasing the polynomial computation rate. The difference lies in the fact that Ref. [30] designed the core arithmetic unit targeting both the Kyber and Dilithium algorithms. Consequently, it specifically designed the register network and data paths for the first three columns of BFUs to support the three-stage PWM operation corresponding to the TNTT algorithm. The comparison based on LUT ATP data indicates that this design mode enhances the scalability of the hardware module's computational process but also increases area consumption and, to some extent, reduces the hardware resource reuse rate. The core reasons are as follows: as described in Section 3.2, since Dilithium's parameter set supports the standard NTT process, the PWM and NTT operations at the hardware level meet the requirements for isomorphic design. In contrast, the additional hardware resources consumed by the first three columns of specialized BFUs in Ref. [30] only come into play during the PWM operations of the Kyber algorithm. Furthermore, judging from the instantiated MLWE structures of the two algorithms, the computational load of the Kyber algorithm is much lower than that of the Dilithium algorithm. Under similar sampling rate conditions, the performance improvement offered by the core polynomial arithmetic unit differs significantly between the two algorithms. This implies that a

4 × 8 structured systolic array unit exceeds the normal acceleration requirements of the Kyber algorithm and is more suitable as a specialized arithmetic module for the Dilithium algorithm. In comparison, Ref. [26] has the lowest ATP value, indicating a more balanced comprehensive performance of its core arithmetic module. The main reason is that the authors performed a specialized design for the Dilithium algorithm, using four BFUs to construct a pipelined core arithmetic unit. Secondly, we analyze the data dependency between the arithmetic module and the sampling module, designing a cascaded Keccak core to generate the random bitstream required for sampling operations. This roughly matches the input/output rates of the polynomial sampling module and the arithmetic module, reducing the ineffective waiting time of the core computations at the state machine level, which also means hardware resources were effectively utilized in most clock cycles.

The aforementioned two pieces of references are compared with this work. According to the fundamental RTL design principle of "speed-area trade-off", the pure hardware design proposed in Ref. [26] better balances resource consumption in terms of both area and time, offering better comprehensive performance and certain cost advantages in practical deployment. The core arithmetic module designed in Ref. [30] offers better flexibility and scalability, suitable for more diverse and resource-insensitive application scenarios. In contrast, the design scheme proposed in this paper holds a clear advantage in computational efficiency, better fulfilling the requirements of high-performance computing application scenarios.

4.3 Comparative analysis of comprehensive performance

Table 4 provides a comparison between the implementation scheme proposed in this paper and related works, including both pure hardware implementations and software-hardware co-designs, along with detailed comparative analysis results. To more fairly and accurately reflect the performance advantages of the proposed scheme, the performance data of all pure hardware implementation schemes are visualized in Figure 9, where the vertical axis unit is μs. As mentioned in Section 1, although Ref. [25] is a software-hardware co-design based on the Zynq platform, the core computational process of the signature algorithm is entirely implemented on the PL side, with the PS side only responsible for initial signal control and final data output. Therefore, the test data from Ref. [25] is also included in Figure 9. Furthermore, Ref. [30] mentioned in Table 3 deployed and tested the core polynomial arithmetic module on the Artix-7 platform. However, the complete implementation scheme proposed in that work adopted a software-hardware dual-core design based on the Rocket Core and was deployed and tested using TSMC technology, which differs significantly from other implementation schemes. Therefore, we do not consider including Ref. [30] in the comparative analysis of Table 4.

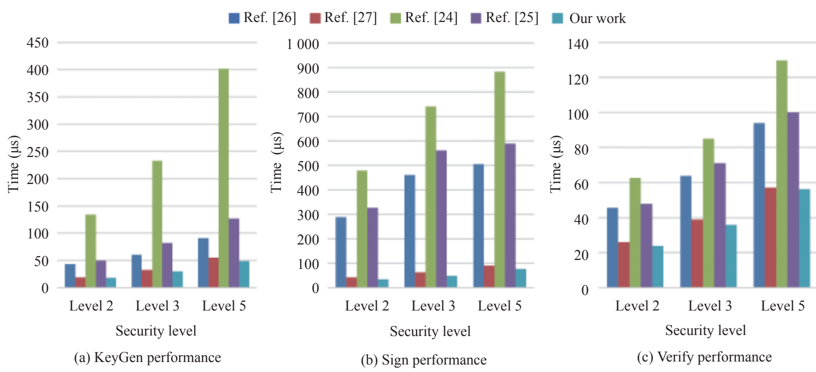

Figure 9 Performance comparison of design schemes

Table 4 Overall performance comparison of hardware design schemes

Refs.	LUT	FF	BRAM	DSP	f (MHz)	KeyGen		Sign		Verify		Security level
						t (μs)	cycles	t (μs)	cycles	t (μs)	cycles	
						17.9	2,348	34.4	4,508	23.8	3,123	2
Ours[a,1]	37,943	30,268	18.5	64	131	29.4	3,864	49.2	6,458	36	4,718	3
						48.9	6,401	76	9,958	56.2	7,763	5
						43.1	4,172	289.9	28,091	45.6	4,422	2
[26][a,1]	29,998	10,366	11	10	96.9	60.4	5,851	461.3	44,706	63.8	6,181	3
						90.5	8,765	505.6	48,996	93.9	9,039	5
						19	4,875	43	10,945	26	6,582	2
[27][b,1]	53,907	28,435	29	16	256	32	8,291	63	16,178	39	9,724	3
						55	14,037	95	24,358	57	13,642	5
	24,320	9,668	15	45	140	134	18,761	478.3	66,966	62.6	8,770	2
[24][a,1]	29,987	11,274	23	45	142	233.1	33,102	740.3	10,5129	85.1	12,084	3
	42,860	14,136	33	45	127	401.4	50,982	883	11,2145	129.6	16,462	5
	18,558	7,432	17	10	159	49	7,757	327	52,038	48	7,675	2
[25][c,2]	19,614	8,466	21	10	159	82	12,982	561	89,213	71	11,232	3
	20,973	9,677	28	10	159	127	20,189	589	93,708	100	15,875	5
[28][2]	—	—	—	—	72	2,325.5	167 433	8,816.2	634,763	3,187.2	229,481	2
						3,101	223,272	11,328.2	815,636	3,836.4	276,221	3
[29][d,2]	2,620	—	16	6	100	—	—	12,600	—	9,940	—	3
						731	438,698	2,829	1,697,566	904	542,212	2
[32][c+d,2]	6,639	1,660	3	16	600	1,330	798,111	4,685	2,990,703	1,466	879,854	3
						2,085	1,251,278	5,615	3,369,190	2,298	1,378,741	5
						1,100		2,300		1,100		2
[33][c,2]	13,128	11,556	14	4	150	1,500	—	3,100	—	1,600	—	3
						2,200		4,500		2,300		5

Notation: Superscript letters a, b, c, and d correspond to four test platforms: a: Artix-7; b: VirtexUltraScale $+$; c: Zynq; d: PolarFire; Superscript numbers 1 and 2 correspond to two design schemes: 1: Pure hardware design; 2: Software-hardware co-design

Ref. [26] designed a specialized hardware circuit architecture for the third-round Dilithium algorithm parameter sets, supporting digital signature operations under all three security levels. Similar to this work, its core polynomial arithmetic module is also designed based on a delay-steering structure, using BRAM units to achieve inter-stage data delays. Each BFU handles two adjacent stages of the NTT computation, and thus only four butterfly arithmetic units are instantiated. This circuit structure saves hardware resources, especially register consumption, to a certain extent, but it also limits the efficiency of polynomial operations: only one polynomial coefficient can be input per clock cycle, and one complete NTT operation consumes 296 clock cycles, including a fixed latency of 40 clock cycles. We adopt a highly parallel systolic array design architecture, significantly increasing the module's data throughput. Completing a single NTT operation requires only 32 clock cycles. Simultaneously, supplemented by the segmented timing state control logic, the core computational steps of the signature algorithm are restructured into three parts: PWM, INTT, and modular addition. These parts are processed uniformly using vector groups as the basic unit, which maximizes the advantages of the high throughput of the systolic array arithmetic unit, thereby enhancing the operational efficiency of the hardware architecture. Overall, the design proposed in Ref. [26] consumes 29,998 LUTs, 10,366 FFs, 11 BRAMs, and 10 DSPs, showing some advantage in area resources, but its clock cycle consumption is significantly higher than the scheme proposed in this paper. Additionally, the maximum clock frequency achieved in this work is also superior to that in Ref. [26]. The primary reason for this lies in the design of the sampling module. As mentioned earlier, although a single-core hardware architecture based on a cascaded design can also improve sampling efficiency, it requires designing more complex combinational logic and completing two consecutive permutation function iterations within a single clock cycle. This greatly extends the clock setup time T_{su} and hold time T_h, thereby limiting the increase in clock frequency. In contrast, we adopt a dual-core design approach and optimize the Keccak core specifically for the characteristics of different sampling operations, achieving a higher clock frequency while meeting parallel sampling requirements. Compared to Ref. [26], the overall operational efficiency of the design proposed in this paper is improved by factors of 1.4/7.4/0.9, 1.1/8.3/0.7, and 0.8/5.6/0.6 for the three security levels, respectively. Further analysis of the performance comparison results shows that the efficiency improvement is most pronounced for the signature generation process, followed by key generation, while the performance gain for signature verification is the lowest. The main reasons are related to the operation types and vector dimensions: the signature generation process involves the largest number of polynomials. The segmented timing state control logic, based on fine-grained optimization, combined with the highly parallel systolic array unit, can effectively improve overall computational efficiency. Therefore, compared to the other two parts, signature generation achieves higher performance improvement. Furthermore, compared to key generation, the signature verification process requires an additional $H(\mu \parallel \boldsymbol{w}_1')$ operation. Since the dual-core hash module in this design cannot accelerate the $H(\mu \parallel \boldsymbol{w}_1')$ operation, whereas the cascaded Keccak core in Ref. [26] can still perform parallel computation in this case, the performance improvement effect is lower for the signature verification process, which has a higher density of hash operations.

Ref. [27] implemented a polynomial arithmetic unit design with a 2×2 architecture. It utilized the data buffering capability of FIFOs to achieve operand matching and implemented the NTT operation based on an iterative memory access architecture. This design can input four polynomial coefficients simultaneously per clock cycle, with each BFU handling two adjacent stages of the NTT computation. Completing one forward or INTT operation consumes 64×4 clock cycles. Compared to the systolic array arithmetic unit in this work, it has some advantages in terms of hardware resource occupancy, but there is a significant gap in operational efficiency. The number of clock cycles consumed for the core signature operation is more than double that of this work. Ref. [27] deployed and tested the scheme on the high-performance Virtex UltraScale + platform, and it achieved a higher clock frequency. However, overall, the design scheme in this paper still holds certain advantages in terms of operational efficiency, with time consumption being 92%/80%/91.5%, 90.6%/78.1%/92%, and 87.5%/83%/98.5% of that in Ref. [27]. Since Ref. [27] used three Keccak cores to handle different types of operations and did not propose module optimization techniques similar to those in this work, it consumed more hardware resources. According to the comparative data provided in Table 4, its consumption of LUT resources and BRAM resources is 1.4 times and 1.5 times that of this work, respectively.

Ref. [24] also proposed a hardware design supporting the third-round Dilithium algorithm parameters. The distinction is that the three parameter sets correspond to three different circuit structures, resulting in lower hardware resource compatibility and reuse rates compared to our work and Ref. [26]. Although Ref. [24] specifically designed separate NTT and MACC modules for polynomial NTT and multiplication operations, respectively, there exist certain timing differences between the different functional modules. The clock cycles required to complete one forward NTT, INTT, PWM, and PWA are 533, 536, 85, and 75, respectively. Our work, based on the high-performance systolic array unit, uniformly implements all these arithmetic processes. This not only improves computational efficiency but also standardizes the clock cycle counts for different operation types, thereby providing feasibility for the design and optimization of the top-level timing control state machine. Overall, for the three parameter sets of Dilithium and using the same hardware test platform, the hardware design proposed in this paper improves the operational efficiency for key generation, signature generation, and signature verification processes by factors of 6.6/12.9/1.6, 7.0/14.0/1.3, and 7.3/10.6/1.3, respectively.

Refs. [25, 28, 29, 32, 33] proposed various software-hardware co-design schemes for the Dilithium algorithm. The core idea is to combine the efficiency of unified hardware design with the flexibility of software implementation. They design dedicated hardware accelerators for complex computational processes like polynomial multiplication, while invoking these hardware functional modules and managing the algorithm logic at the software level. Ref. [28] utilized the RISC-V instruction set for its design, supporting the parameter sets from the NIST second-round Dilithium algorithm. Since this scheme only implemented two dedicated functional modules (NTT and Keccak), its overall hardware resource footprint is relatively small. Using instruction set language to call hardware functional modules can effectively accelerate parts of the computation; however, compared to pure hardware designs, the overall degree of parallelism is lower, limiting the potential for further enhancing the algorithm's overall performance. Ref. [29] designed a dedicated polynomial arithmetic module based on the Zynq-7020 platform, requiring 1,170 clock cycles to complete one NTT operation. This demonstrates a lower performance improvement effect compared to the systolic array arithmetic unit in this work.

Refs. [25, 32] are two different software-hardware co-design schemes developed by the same authors. Ref. [33], like Ref. [25], implemented the complete Dilithium algorithm on the Zynq-7000 platform but employed a different design approach. Given the interrelated nature of the design schemes in these three references, a unified comparative analysis was conducted. Firstly, Ref. [25] adopted a full-hardware development approach, where all signature algorithm functions are implemented at the hardware level. The software part is only responsible for basic enable signals and control logic. This design pattern effectively reduces the interaction frequency between the software and hardware platforms, thereby enhancing the overall operational efficiency of the algorithm. Furthermore, we designed a dedicated radix-4 polynomial arithmetic module, where each NTT operation fixedly consumes 264 clock cycles. In contrast, the systolic array unit designed in this work, operating under pipelined conditions, sees its per-operation latency gradually decrease as the volume of processed data increases, better aligning with the dataset characteristics of the Dilithium algorithm. Overall, the design scheme in this paper still holds a significant performance advantage, improving operational efficiency by factors of 1.7/8.5/1.0, 1.8/10.4/0.9, and 1.6/6.75/0.7 for the three parameter sets, respectively. In comparison, the design in Ref. [32] features a higher degree of decoupling between software and hardware. The hardware side is primarily responsible for accelerating polynomial operations. Considering communication overhead and overall scalability, the more time-consuming SHA-3 function and corresponding sampling operations were implemented on the software side and optimized using instruction set languages, ultimately completing the design and implementation for both the Kyber and Dilithium algorithms. Additionally, the authors conducted three types of performance tests: the first is pure software single-core execution, the second is single-core execution in software-hardware co-design, and the third is multi-core execution in software-hardware co-design (where multi-core implies doubled sampling rates). Table 4 presents the test data under the highest-performance multi-core condition. Comparison with Ref. [25] shows that although the multi-core scheme in software-hardware co-design can support multiple algorithms, its overall performance decreases significantly. Finally, Ref. [33] also adopted a decoupling philosophy for its software-hardware co-design. It implemented the polynomial arithmetic module and the SHAKE module on the PL side of the Zynq-7000 platform, while the PS side handles function calls and algorithm logic. As mentioned in Section 4.2, Ref. [33] designed three separate hardware modules for NTT/INTT operations, PWM operations, and PWA operations, respectively, and did not adopt the hardware time-sharing reuse concept mentioned in Ref. [38]. Consequently, it consumed more hardware resources compared to Ref. [32]. A comprehensive comparative analysis of the aforementioned three design schemes with this work reveals that as the degree of hardware implementation increases, the overall operational efficiency of the algorithm also improves. The core reasons are twofold:

(1) The parallel computation in pure hardware designs is manifested at both the functional

module and the overall architecture levels, resulting in a higher degree of parallelism compared to other design approaches.

(2) As indicated in Refs. [32, 33], the data interaction process between software and hardware will consume a significant number of clock cycles, which subsequently impacts the maximum achievable frequency.

Therefore, design schemes based on the software-hardware co-design approach generally exhibit lower overall performance than pure hardware design schemes.

5 Summary

We proposed a novel pure hardware implementation scheme for the NIST third-round lattice digital signature algorithm Dilithium. Based on a reconfigurable design philosophy, a unified circuit architecture supporting key generation, digital signature generation, and signature verification under all three parameter sets was designed and implemented. The proposed fine-grained parallel segmented timing control logic effectively matched the high-throughput advantages of the systolic array unit. Simultaneously, a new optimized design scheme featuring dual Keccak cores was developed according to the characteristics of the sampling operations. This enabled the final hardware design architecture to possess the dual advantages of high performance and high resource reuse rate. Our work provides valuable insights and reference for advancing the engineering and practical application process of lattice-based cryptographic schemes.

References

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

[2] Shor PW. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Review, 1999, 41(2): 303–332. [doi: 10.1137/S0036144598347011]

[3] Shor PW. Algorithms for quantum computation: Discrete logarithms and factoring. Proc. of the 35th Annual Symp. on Foundations of Computer Science. Santa Fe: IEEE, 1994. 124–134. [doi: 10.1109/SFCS.1994.365700]

[4] Grover LK. A fast quantum mechanical algorithm for database search. Proc. of the 28th Annual ACM Symp. on Theory of Computing. 1996. 212–219.

[5] National Institute of Standards and Technology. Post-quantum cryptography—Call for proposals. 2017. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/Call-for-Proposals

[6] Chinese Association for Cryptologic Research. Public key algorithms selected to the second round competition of national cyptographic algorithm competitions. 2019. https://sfjs.cacrnet.org.cn/site/term/list_77_1.html

[7] National Institute of Standards and Technology. Post-quantum cryptography—Round 3 submissions. 2020. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions

[8] Kumari S, Singh M, Singh R, Tewari H. Signature based Merkle hash multiplication algorithm to secure the communication in IoT devices. Knowledge-based Systems, 2022, 253: 109543. [doi: 10.1016/j.nosys.2022.109543]

[9] Engelbert D, Overbeck R, Schmidt A. A summary of McEliece-type cryptosystems and their security. Journal of Mathematical Cryptology, 2007, 1(2): 151–199. [doi: 10.1515/JMC.2007.009]

[10] Courtois N, Klimov A, Patarin J, Shamir A. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. Proc. of the 2000 Int'l Conf. on the Theory and Applications of Cryptographic Techniques. Belgium: Springer, 2000. 392–407. [doi: 10.1007/3-540-45539-6_27]

[11] Peng C, Chen JH, Zeadally S, He DB. Isogeny-based cryptography: A promising post-quantum technique. IT Professional, 2019, 21(6): 27–32. [doi: 10.1109/MITP.2019.2943136]

[12] Nejatollahi H, Dutt N, Ray S, Regazzoni F, Banerjee I, Cammarota R. Post-quantum lattice-based cryptography implementations: A survey. ACM Computing Surveys (CSUR), 2019, 51(6): 129. [doi: 10.1145/3292548]

[13] Avanzi R, Bos J, Ducas L, Kiltz E, Lepoint T, Lyubashevsky V, Schanck JM, Schwabe P, Seiler G, Stehlé D. CRYSTALS-Kyber: Algorithm specifications and supporting documentation (version 3.01). 2021. https://pq-crystals.org/kyber/data/kyber-specificationround3-20210131.pdf

[14] Bai S, Ducas L, Kiltz E, Lepoint T, Lyubashevsky V, Schwabe P, Seiler G, Stehlé D. CRYSTALS-Dilithium algorithm specifications and supporting documentation (version 3.1). 2021. https://pq-crystals.org/ dilithium/data/dilithium-specification-round3-20210208.pdf

[15] Fouque PA, Hoffstein J, Kirchner P, Lyubashevsky V, Pornin T, Prest T, Ricosset T, Seiler G, Whyte W, Zhang ZF. Falcon: Fast-Fourier lattice-based compact signatures over NTRU (Specification v1.2). NIST PQC Round 3 Submission, 2020. https://falcon-sign.info/falcon.pdf

[16] National Institute of Standards and Technology. PQC standardization process: Announcing four candidates to be standardized, plus fourth round candidates. 2022. https://csrc.nist.gov/News/2022/pqc-candidates-to-be-standardized-and-round-4

[17] National Institute of Standards and Technology. Module-lattice-based digital signature standard. 2024. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.204.pdf

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

[19] Kannwischer MJ, Rijneveld J, Schwabe P, Stoelen K. pqm4: Testing and Benchmarking NIST PQC on ARM Cortex-M4. Cryptology ePrint Archive, 2019/844, 2019.

[20] Kim Y, Song JY, Youn TY, Seo SC. Crystals-Dilithium on ARMv8. Security and Communication Networks, 2022, 2022: 5226390. [doi: 10.1155/2022/5226390]

[21] Zheng JY, He F, Shen SY, Xue CX, Zhao YL. Parallel small polynomial multiplication for Dilithium: A faster design and implementation. Proc. of the 38th Annual Computer Security Applications Conf. Austin: ACM, 2022. 304–317. [doi: 10.1145/3564625.3564629]

[22] Soni D, Basu K, Nabeel MM, Karri R. A hardware evaluation study of NIST post-quantum cryptographic signature schemes. Proc. of the 2nd PQC Standardization Conf. National Institute of Standards and Technology, 2019. https://api.semanticscholar.org/CorpusID:198939541

[23] Ricci S, Malina L, Jedlicka P, Smékal D, Hajny J, Cibik P, Dzurenda P, Dobias P. Implementing CRYSTALS-Dilithium signature scheme on FPGAs. Proc. of the 16th Int'l Conf. on Availability, Reliability and Security. Vienna: ACM, 2021. 1–11. [doi: 10.1145/3465481.3465756]

[24] Land G, Sasdrich P, Güneysu T. A hard crystal-implementing Dilithium on reconfigurable hardware. Proc. of the 20th Int'l Conf. on Smart Card Research and Advanced Applications. Lübeck: Springer, 2021. 210–230. [doi: 10.1007/978-3-030-97348-3_12]

[25] Wang TF, Zhang C, Cao P, Gu DW. Efficient implementation of Dilithium signature scheme on FPGA SoC platform. IEEE Trans. on Very Large Scale Integration (VLSI) Systems, 2022, 30(9): 1158–1171. [doi: 10.1109/TVLSI.2022.3179459]

[26] Zhao CK, Zhang N, Wang HN, Yang BH, Zhu WP, Li ZD, Zhu M, Yin SY, Wei SJ, Liu LB. A compact and high-performance hardware architecture for CRYSTALS-Dilithium. IACR Trans. on Cryptographic Hardware and Embedded Systems, 2022, 2022(1): 270–295. [doi: 10.46586/tches.v2022.i1.270-295]

[27] Beckwith L, Nguyen DT, Gaj K. High-performance hardware implementation of CRYSTALS-Dilithium. Proc. of the 2021 Int'l Conf. on Field-programmable Technology (ICFPT 2021). Auckland: IEEE, 2021. 1–10. [doi: 10.1109/ICFPT52863.2021.9609917]

[28] Banerjee U, Ukyab TS, Chandrakasan AP. Sapphire: A configurable crypto-processor for post-quantum lattice-based protocols. IACR Trans. on Cryptographic Hardware and Embedded Systems, 2019(4): 17–61. [doi: 10.13154/tches.v2019.i4.17-61]

[29] Zhou Z, He D, Liu Z, Luo M, Choo KKR. A software/hardware co-design of Crystals-Dilithium signature scheme. ACM Trans. on Reconfigurable Technology and Systems, 2021, 14(2): 11. [doi: 10.1145/3447812]

[30] Zhao YF, Xie RQ, Xin GZ, Han J. A high-performance domain-specific processor with matrix extension of RISC-V for module-LWE applications. IEEE Trans. on Circuits and Systems I: Regular Papers, 2022, 69(7): 2871–2884. [doi: 10.1109/TCSI.2022.3162593]

[31] Xin GZ, Han J, Yin TY, Zhou YC, Yang JW, Cheng X, Zeng XY. VPQC: A domain-specific vector processor for post-quantum cryptography based on RISC-V architecture. IEEE Trans. on Circuits and Systems I: Regular Papers, 2020, 67(8): 2672–2684. [doi: 10.1109/TCSI.2020.2983185]

[32] Wang TF, Zhang C, Zhang XL, Gu DW, Cao P. Optimized hardware-software co-design for Kyber and Dilithium on RISC-V SoC FPGA. IACR Trans. on Cryptographic Hardware and Embedded Systems, 2024, 2024(3): 99–135. [doi: 10.46586/tches.v2024.i3.99-135]

[33] Mao GY, Chen DL, Li GY, Dai WC, Sanka AI, Koç CK, Cheung RCC. High-performance and configurable SW/HW co-design of post-quantum signature CRYSTALS-Dilithium. ACM Trans. on Reconfigurable Technology and Systems, 2023, 16(3): 44. [doi: 10.1145/3569456]

[34] Aikata A, Mert AC, Jacquemin D, Das A, Matthews D, Ghosh S, Roy SS. A unified cryptoprocessor for lattice-based signature and key-exchange. IEEE Trans. on Computers, 2023, 72(6): 1568–1580. [doi: 10.1109/TC.2022.3215064]

[35] Mert AC, Karabulut E, Öztürk E, Savaş E, Aysu A. An extensive study of flexible design methods for the number theoretic transform. IEEE Trans. on Computers, 2022, 71(11): 2829–2843. [doi: 10.1109/TC.2020.3017930]

[36] Yaman F, Mert AC, Öztürk E, Savaş E. A hardware accelerator for polynomial multiplication operation of CRYSTALS-KYBER PQC scheme. Proc. of the 2021 Design, Automation & Test in Europe Conf. Exhibition (DATE). Grenoble: IEEE, 2021. 1020–1025. [doi: 10.23919/DATE51398.2021.9474139]

[37] Zhao XY, Liang ZC, Hu Y, Geng HX, Zhao YL. NTT architecture research and its FPGA hardware optimization implementation. Chinese Journal of Computers, 2023, 46(12): 2670–2686 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2023.02670]

[38] Xing YF, Li SG. A compact hardware implementation of CCA-secure key exchange mechanism CRYSTALS-KYBER on FPGA. IACR Trans. on Cryptographic Hardware and Embedded Systems, 2021, 2021(2): 328–356. [doi: 10.46586/tches.v2021.i2.328-356]

[39] Hu Y, Zhao XY, Liu YX, Zhao YL. Hardware implementation of lattice-based key encapsulation mechanism algorithm OSKR/OKAI. Chinese Journal of Computers, 2023, 46(6): 1156–1171 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2023.01156]

[40] Mookherjee S, DeBrunner L, DeBrunner V. A low power radix-2 FFT accelerator for FPGA. Proc. of the 49th Asilomar Conf. on Signals, Systems and Computers. Pacific Grove: IEEE, 2015. 447–451. [doi: 10.1109/ACSSC.2015.7421167]

[41] Dang VB, Mohajerani K, Gaj K. High-speed hardware architectures and FPGA benchmarking of CRYSTALS-Kyber, NTRU, and Saber. IEEE Trans. on Computers, 2023, 72(2): 306–320. [doi: 10.1109/TC.2022.3222954]

[42] Barrett P. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In: Odlyzko AM, ed. Proc. on Advances in Cryptology—CRYPTO 1986. Berlin: Springer, 1987. 311–323. [doi: 10.1007/3-540-47721-7_24]

[43] Montgomery PL. Modular multiplication without trial division. Mathematics of Computation, 1985, 44(170): 519–521. [doi: 10.1090/S0025-5718-1985-0777282-X]

Yue Hu, Ph.D. His research interests include post-quantum cryptography and cryptographic engineering.

Xuyang Zhao, Ph.D. candidate. His research interests include post-quantum cryptography, cryptographic engineering, and software/hardware co-design.

Wei Wang, Ph.D. candidate. Her research interests include lattice cryptography theory and its applications.

Jieyu Zheng, Ph.D. candidate. Her research interests include post-quantum cryptography and cryptographic engineering.

Qian Yuan, master's degree candidate. His research interest is cryptography engineering.

Yafang Yang, Ph.D. Her research interests include cryptography and its applications.