A Lightweight CRYSTALS-Dilithium Post-quantum Cryptography Signature for FPGA-Based Edge Computing

Huy Minh Dung Do[1,2], Duc Hoang Pham[1,2], Thien Hai Ha[1,2], and Cuong Pham-Quoc[1,2(✉)]

[1] Ho Chi Minh City University of Technology (HCMUT), Ho Chi Minh City, Vietnam
{dung.do05050,hoang.pham2004,hai.hathien,cuongpham}@hcmut.edu.vn
[2] Vietnam National University - Ho Chi Minh City (VNU-HCM), Ho Chi Minh City, Vietnam

Abstract. In this golden time of edge computing, while more IoT devices and distributed services are increasingly processing security-sensitive data close to the data source, efficient authentication has become a critical requirement under strict latency, power, and resource constraints. At the same time, advances in quantum computing threaten the widespread deployment of public-key cryptography, motivating the transition to post-quantum digital signatures, such as CRYSTALS-Dilithium, which has been standardized by NIST as ML-DSA (Module-Lattice-Based Digital Signature Algorithm). However, practical ML-DSA deployment on edge platforms remains challenging due to compute- and memory-intensive kernels, including NTT/INTT-based polynomial arithmetic, SHAKE-based hashing, and signing-time rejection sampling, as well as non-negligible data movement overhead in SoC systems. This paper presents a lightweight FPGA-based ML-DSA signature accelerator tailored for FPGA-based edge computing, following a HW/SW co-design approach that offloads arithmetic-dominant kernels to programmable logic while retaining protocol orchestration and standard-compliant serialization in software. We implement the proposed design on the Xilinx Kria KV260 edge platform and evaluate it in terms of hardware utilization, timing, signing and verification latency, throughput, and power/energy consumption. Experimental results demonstrate that the FPGA-accelerated design enhances end-to-end performance compared to a software-only baseline on the embedded ARM processor, while maintaining a compact resource footprint and energy efficiency suitable for power-sensitive edge deployments. The proposed architecture is modular and can be integrated with other edge security functions, enabling practical post-quantum authentication for IoT gateways and edge services.

Keywords: FPGA · Hardware accelerator · Post-quantum cryptography · ML-DSA · CRYSTALS-Dilithium · Edge Computing · IoT

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2027
O. Gervasi et al. (Eds.): ICCSA 2026 Workshops, LNCS 16763, pp. 333–350, 2027.
https://doi.org/10.1007/978-3-032-30530-5_21

1 Introduction

In the era of edge computing, as IoT systems and AI-enabled services continue to move computation closer to data sources [10,25], security and privacy have become critical concerns due to the reliance on distributed devices operating under strict resource and power constraints. At the same time, advances in quantum computing pose a fundamental threat to widely deployed public-key cryptography. In particular, Shor's algorithm makes RSA and elliptic-curve cryptography vulnerable once large-scale quantum computers become practical [21], raising serious concerns about long-lived data and the authentication mechanisms that underpin modern networks and services [8]. This challenge is amplified by the "harvest now, decrypt later" paradigm, where adversaries can collect protected communications today in anticipation of future decryption capabilities [1,15,16]. In response, NIST has standardized a first set of post-quantum cryptographic algorithms, including ML-DSA (CRYSTALS-Dilithium) for digital signatures [3,17].

Although ML-DSA provides a promising path toward quantum-resistant authentication, deploying it efficiently on edge platforms remains challenging. Dilithium relies on lattice-based hardness assumptions, primarily Module Learning With Errors and Module Short Integer Solution [12,19], which are widely regarded as resistant to classical and quantum adversaries [4,18]. However, practical implementations must execute compute and memory-intensive kernels, including NTT-based polynomial arithmetic, SHAKE-based hashing and rejection sampling, and must do so while meeting real-time constraints and tight energy budgets [17,26]. These requirements are particularly relevant in edge-centric security settings, where digital signatures are needed to provide integrity, authenticity, and non-repudiation [6], and where security workloads are often offloaded from constrained devices to nearby edge nodes (e.g., security agents) to improve scalability [9,20].

Field-Programmable Gate Arrays (FPGAs) offer an attractive solution to address these challenges by combining reconfigurability with energy-efficient parallelism. This makes FPGAs particularly suitable for edge platforms that must deliver predictable performance under tight power and resource budgets, while still retaining the flexibility to adapt to evolving cryptographic standards and deployment requirements [13]. In this paper, we present *A Lightweight CRYSTALS-Dilithium Post-Quantum Cryptography Signature for FPGA-based Edge Computing*. Instead of treating the edge node as a generic compute accelerator, we focus on the digital-signature workload itself and design a compact Dilithium (ML-DSA) signature engine tailored to edge constraints. The proposed design focuses on the practical sign and verification at the edge, targeting low latency and resource efficiency while preserving the security properties required by the standardized algorithm [17]. To this end, we focus on the dominant computations in Dilithium, including NTT-based polynomial arithmetic and SHAKE-based hashing, and expose a hardware/software interface that supports integration into edge applications and security offload settings [26].

The proposed FPGA-based signature engine is evaluated with various parameters, including hardware utilization, power efficiency, and signature performance (latency and throughput). The results indicate that our approach is well-suited for security-sensitive edge environments such as IoT networks, industrial monitoring, and real-time data acquisition systems, where quantum-resistant authentication must be delivered within limited resources [10].

The main contributions of our paper can be summarized in threefold.

1. We propose a lightweight FPGA-based CRYSTALS-Dilithium (ML-DSA) digital signature design for FPGA-based edge computing, focusing on resource-efficient and low-latency signing and verification.
2. We demonstrate the proposed design with a prototype on the Xilinx Kria KV260 edge computing platform and report end-to-end performance in a realistic SoC-FPGA setting.
3. We present and analyze experimental results in terms of hardware utilization and signature performance to facilitate reproducible comparisons and future optimization studies for post-quantum signature deployment at the edge.

The rest of the paper is structured as follows. Section 2 presents the background on the work related to CRYSTALS-Dilithium/ML-DSA and FPGA-based post-quantum systems. We introduce our proposed FPGA-based Dilithium signature design in Sect. 3. Section 4 discusses our prototype with Xilinx Kria KV260 edge computing. Experiments with the prototype platform and different configurations are analyzed in Sect. 5. Finally, Sect. 6 concludes our paper.

2 Preliminaries and Related Work

In this section, we first summarize the background of post-quantum digital signatures and CRYSTALS-Dilithium (standardized as ML-DSA in FIPS 204) that are relevant to FPGA-based edge deployments. We highlight the underlying hardness assumptions and the main computational kernels that dominate practical implementations. We then analyze related work on FPGA acceleration of ML-DSA, with emphasis on SoC-FPGA co-design considerations such as workload partitioning and data movement.

2.1 CRYSTALS-Dilithium (ML-DSA) Preliminaries

CRYSTALS-Dilithium is a lattice-based digital signature scheme selected in the NIST Post-Quantum Cryptography standardization effort and standardized as the Module-Lattice-Based Digital Signature Standard (ML-DSA) under FIPS204 [3,17]. Digital signatures remain a core mechanism for ensuring data authenticity, integrity, and non-repudiation in distributed systems, including edge and IoT environments where devices frequently exchange security-critical messages. FIPS 204 specifies three sets of standardized parameters, namely ML-DSA-44, ML-DSA-65 and ML-DSA-87, corresponding to increasing security strengths and implementation costs [17]. In many edge use cases that require

long-term trust in signed data, the highest-security parameter set is attractive, but it also imposes the heaviest arithmetic workload and the pressure of the memory bandwidth [17].

From a security point of view, ML-DSA targets strong unforgeability in chosen-message attacks, based on the hardness of the module-lattice assumptions [17]. In particular, FIPS 204 relates the security of ML-DSA to the hardness of Module Learning With Errors (MLWE) and a structured MSIS-type problem called SelfTargetMSIS. Informally, a successful forgery would imply an efficient solver for these underlying module-lattice problems [12,19]. The credibility of LWE-like assumptions is supported by worst-case to average-case reductions that connect average-case problems encountered by an attacker to worst-case lattice problems such as GapSVP and SIVP, with subsequent work strengthening the classical hardness picture for structured settings [4,12,18,19].

From an implementation perspective, ML-DSA is dominated by structured polynomial arithmetic. Computations are carried out on the polynomial ring $R_q = \mathbb{Z}_q[x]/(x^{256} + 1)$, and efficient polynomial multiplication is based on the Number-Theoretic Transform (NTT) and its inverse [17]. This NTT-centric structure appears repeatedly in key generation, signing, and verification, for example when computing commitments of the form $w = Ay$ using NTT-domain multiplication. In addition, the signing procedure contains a rejection sampling loop that repeatedly generates a signature candidate until norm constraints are met; this step is essential to prevent leakage of secret information through out-of-bound outputs [17]. ML-DSA also uses SHAKE128 and SHAKE256 extensively for matrix expansion, sampling, and challenge generation, and supports a pre-hash interface (HashML-DSA) for deployments that sign message digests rather than raw messages [17].

These characteristics motivate hardware acceleration on edge-class SoC-FPGA platforms, where compute resources and energy budgets are limited, but tight latency requirements are common [10,25]. A practical design pattern is HW/SW co-design: arithmetic-intensive, data-parallel kernels (NTT/INTT, pointwise multiplication, modular arithmetic) are mapped to the programmable logic, while control-heavy tasks (protocol orchestration, parameter checks, serialization, and rejection sampling decision logic) remain on the processing system. This split is typically paired with AXI-based integration, using memory-mapped control paths for configuration and DMA streaming for bulk polynomial data transfers, while on-chip BRAM acts as a low-latency scratchpad. To keep the design lightweight and timing-robust on FPGA fabric, stream-oriented interfaces with moderate data widths, for instance, transferring seeds or coefficients in 64-bit chunks, are often preferable to very wide buses that increase routing congestion and critical path length.

2.2 FPGA-Based Dilithium Related Work

FPGA-based acceleration of CRYSTALS-Dilithium has been actively studied because lattice-based signatures impose substantially higher arithmetic and memory costs than classical schemes, which makes software-only deployment

difficult under edge-class constraints. Across prior implementations, the dominant runtime is typically concentrated in structured polynomial arithmetic (especially NTT/INTT and pointwise multiplication) and SHAKE-based hashing, while system-level efficiency is often bounded by memory bandwidth and the overhead of moving polynomial data between the processing system and programmable logic. Consequently, most FPGA-oriented studies can be understood through three recurring design goals: increasing NTT throughput, reducing data-movement overhead, and selecting a practical HW/SW partition that preserves flexibility without sacrificing end-to-end performance.

Therefore, a large part of the literature focuses on NTT-centric optimization, since NTT/INTT is repeatedly invoked in key generation, signing, and verification. Beckwith *et al.* present a high-performance FPGA implementation of CRYSTALS-Dilithium and demonstrate that carefully engineered transform pipelines and polynomial arithmetic datapaths can achieve very low latency compared to earlier FPGA efforts [2]. In a similar spirit, Kundi *et al.* propose a high-performance NTT hardware accelerator intended to support both ML-KEM and ML-DSA, reflecting the broader trend of designing reusable NTT engines that can serve multiple NIST PQC primitives [11]. At the framework level, unified NTT multiplication architectures have also been explored to reduce duplicated hardware and improve portability across platforms, which is attractive in edge deployments where resource budgets are tight and multiple PQC services may coexist [13]. These works collectively indicate that the transform datapath is necessary but not sufficient: the realized system benefit depends on whether the design can keep the NTT pipeline continuously fed and can manage coefficient storage and reordering without introducing stalls.

Beyond arithmetic throughput, several studies show that data movement can become the primary bottleneck once polynomial kernels are accelerated. Carril *et al.* investigate hardware acceleration for high-volume operations of CRYSTALS-Kyber and CRYSTALS-Dilithium and emphasize an execution model that leverages batch processing to maximize effective memory throughput, together with hardware optimization of selected algorithmic components [5]. This direction highlights that host–accelerator interaction, buffering strategy, and transfer granularity may dominate end-to-end latency when the accelerator is invoked frequently and that throughput-oriented designs can benefit substantially from amortizing setup and transfer costs across many requests. While such batch-centric acceleration is well matched to high-volume services (for example, gateway-style nodes that process many signature requests), it also underlines a key challenge for edge settings: if the target is low latency for single signing or verification operations, the communication path and invocation overhead must be explicitly co-optimized with the compute core.

A third line of work addresses precisely this co-optimization through HW/SW partitioning on SoC-FPGA platforms. Mao *et al.* present a SW/HW configurable high-performance co-design of CRYSTALS-Dilithium, proposing a hardware accelerator integrated into a flexible SoC architecture and emphasizing a parameterized design that enables runtime configuration across Dilithium

modes [14]. Their results reinforce that accelerator effectiveness depends not only on fast NTT and hashing modules, but also on how control-dominant tasks and data marshaling are handled in software, and on how efficiently the PS–PL interface supports repeated movement of polynomial vectors. Related SoC-oriented efforts similarly report that careful workload partitioning and integration choices are necessary to obtain meaningful application-level speedups, particularly when standard-compliant encoding/decoding and signing-specific control flow (including rejection checks) remain in the software domain [23,24].

Overall, existing FPGA-based Dilithium research demonstrates strong kernel-level acceleration, yet it also exposes a practical gap that is central to edge computing: peak-throughput or high-volume designs do not automatically translate into lightweight, low-overhead deployments on compact SoC-FPGA platforms. NTT-focused engines [2,11] improve the arithmetic backbone, high-volume accelerators [5] clarify the importance of memory throughput and invocation overhead, and HW/SW co-designs [14] show that end-to-end gains hinge on a disciplined partitioning and a communication-efficient data path. These observations motivate the approach of this paper, which targets a lightweight ML-DSA signature engine for FPGA-based edge computing by jointly optimizing the NTT-centric datapath and the system integration costs that govern practical signing and verification latency under constrained resources.

3 Proposed FPGA-Based Dilithium Core Architecture

In this section, we introduce our FPGA-based CRYSTALS-Dilithium signature accelerator for edge computing. We design the architecture following a hardware acceleration paradigm, so that the proposed core can be integrated with either hard processors (e.g., ARM cores in SoC-FPGAs) or soft general-purpose processors to support practical signing and verification in embedded and edge applications. The design focuses on a lightweight integration model, where compute-intensive kernels such as NTT/INTT-based polynomial arithmetic and SHAKE-based hashing are mapped to the programmable logic, while control-oriented tasks and protocol orchestration are handled by software to preserve flexibility and standard compliance [17].

3.1 Overview Architecture

Figure 1 shows the general architecture of the proposed lightweight CRYSTALS-Dilithium post-quantum signature core designed for FPGA-based edge computing. The core communicates with a host processor through a communication infrastructure (e.g., AXI4 or shared memory) to transfer input data (entropy seeds, public keys, secret keys, messages, and parameters) and output results (generated keys, signatures, and verification decisions). A central "Internal BRAM" is utilized for storing and exchanging intermediate polynomials and data between all modules.

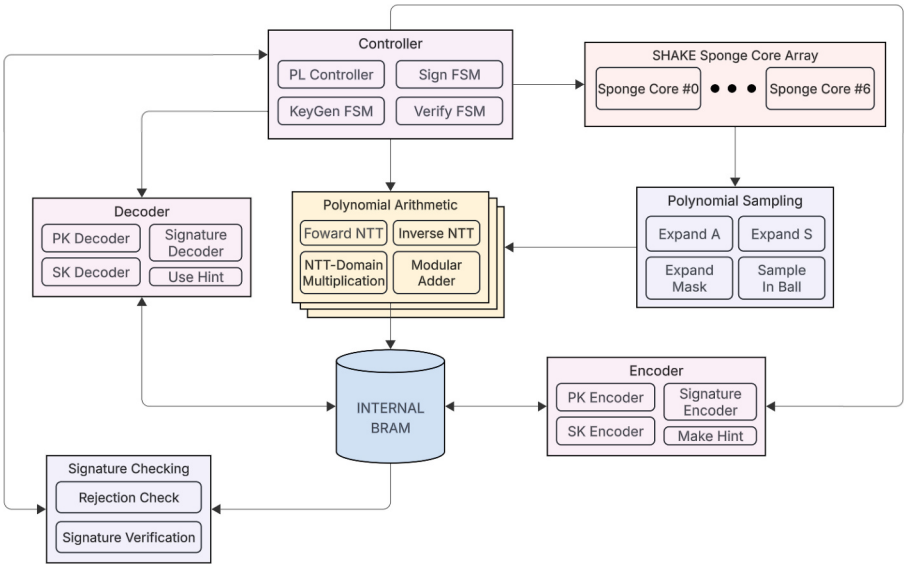

Fig. 1. The overview of the lightweight CRYSTALS-Dilithium Post-Quantum Cryptography Signature for FPGA-based Edge Computing.

The overall operation of the accelerator is managed by a global controller, which includes dedicated finite-state machines (FSMs) for the KeyGen, Sign, and Verify phases. Additional supporting modules, such as encoder/decoder units and signature check logic, handle the data formatting, compression, and validation steps required by the specification.

Among the processing components, two modules play a central role in accelerating the computationally intensive operations of the Dilithium algorithm: the polynomial arithmetic module and the polynomial sampling module.

The polynomial arithmetic module implements the core polynomial operations required by the scheme, including forward and inverse Number Theoretic Transform (NTT/INTT), modular addition and pointwise polynomial multiplication in the NTT domain.

The polynomial sampling module generates the pseudorandom polynomials required by the protocol. The implementations of the sampling procedures were defined in the Dilithium specification (FIPS 204), which derive structured randomness using extendable-output functions based on SHA-3. The detailed architectures of the polynomial arithmetic and polynomial sampling modules are discussed in the following subsections.

3.2 NTT Module

The proposed NTT core adopts a 2×2 radix-2 butterfly architecture, where two consecutive radix-2 butterfly units are arranged to process four input coefficients

in a pipelined manner. As illustrated in the architecture, the first stage consists of two parallel radix-2 butterflies that operate on pairs of inputs (x_1, x_2) and (x_3, x_4). Before entering the butterfly computation, the second operand of each pair is multiplied by the corresponding twiddle factor ζ^i. Each butterfly then performs modular addition and subtraction, generating two intermediate outputs according to Eq. 1.

$$\begin{aligned} t &= \zeta^i \cdot b \\ a' &= a + t \pmod{q}, \qquad b' = a - t \pmod{q}. \end{aligned} \tag{1}$$

The outputs of these two butterflies are then forwarded to the next stage, forming the 2×2 radix-2 structure. In the second stage, another pair of radix-2 butterflies receives the intermediate results from the first stage and performs additional modular operations using updated twiddle factors ζ^{2i+1} and ζ^{2i+2}. This cascading structure allows for four coefficients to be cooperatively processed, enables high-throughput processing, and reduces the complexity of the control. Figure 2 illustrates the NTT processing.

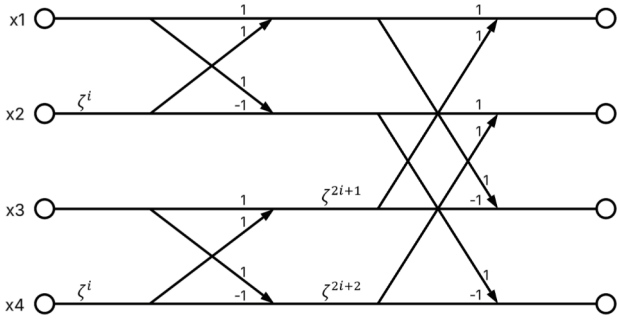

Fig. 2. Proposed 2×2 radix-2 butterfly architecture for the NTT core.

Based on the NTT computation principle, the architecture of the proposed NTT core is designed using a pipelined processing model to improve performance, as presented in Fig. 3. The computation is organized into several pipeline stages including modular multiplication, modular reduction, and butterfly operations. Pipeline registers are inserted between functional units to maintain continuous data flow and increase the operating frequency of the system.

The input coefficients are first received through the `Data in` interface and stored in pipeline registers. In the first computation stage, the input data is multiplied by the corresponding twiddle factor ζ provided by the input coefficient. The multiplication results are then passed through a modular reduction block to compute the modulo result q, ensuring that the values remain within the finite field used by the NTT. After modular reduction, intermediate results are forwarded to the butterfly computation stage, which performs modular addition and subtraction operations. The architecture then repeats a similar sequence

of operations in the subsequent stage, including multiplication, modular reduction, and butterfly computation, corresponding to the cascaded radix-2 butterfly structure.

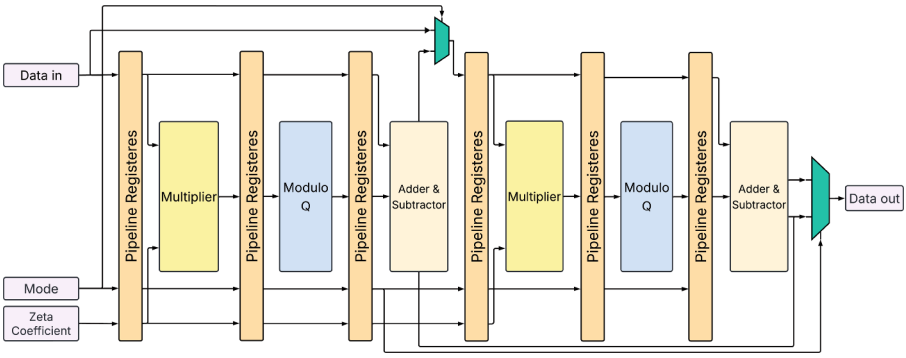

Fig. 3. The pipeline architecture for 2×2 radix-2 butterfly.

Multiplexers are placed at specific points in the data path to support different operation modes and to select appropriate intermediate results during the computation process. The pipeline registers between stages ensure that each functional block operates concurrently, allowing multiple data samples to be processed simultaneously in different stages of the pipeline. Finally, the processed coefficients are forwarded through the output multiplexer and written to the `Data out` interface.

Figure 4 shows the architecture of the polynomial processing block. The Computation Controller coordinates the overall operation, including input handling, data flow control, and output generation. An Address Generation Unit produces the RAM write addresses and twiddle-factor addresses needed for each processing stage. These addresses are used to access Zeta ROM, which stores the twiddle coefficients required by the NTT computation. The selected coefficients are then fed into the RADIX 2×2 block, where four butterfly units perform the core polynomial arithmetic. For inverse NTT processing, the intermediate results are forwarded to the Post-Inv NTT block, which applies the scaling constant, then passes the result to the Parallel-in Serial-out block for data formatting and serialization before being written back or delivered to the output interface. Through this organization, the architecture defines a clear data-path for polynomial computation, where the controller coordinates memory accesses, coefficient retrieved from the twiddle-factor ROMs, and the execution of arithmetic operations in the butterfly units.

3.3 Polynomial Sampling Module

The polynomial sampling module is responsible for generating the polynomials required by the Dilithium algorithms. As illustrated in Fig. 5, the module

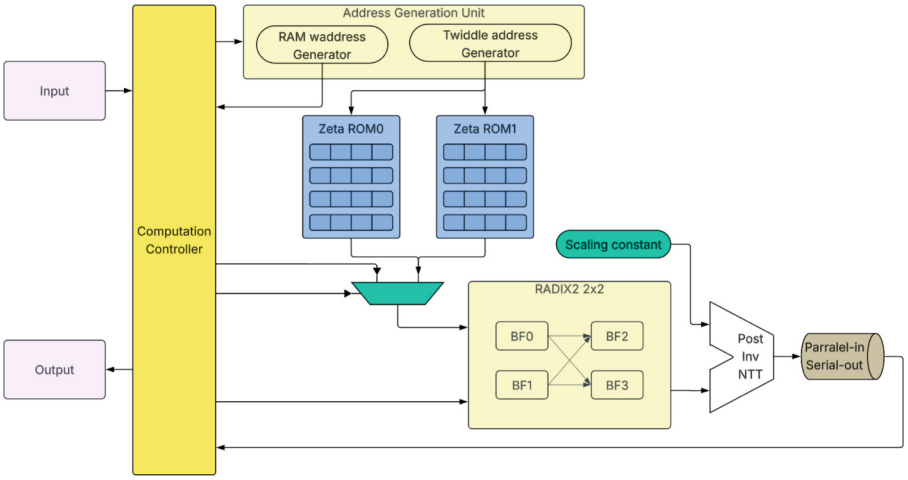

Fig. 4. The architecture of the whole operation module.

consists of three main components: the sampling controller, the SHAKE sponge engine, and the coefficient sampling unit. These components collectively implement the sampling procedures used in the ExpandA, ExpandS, ExpandMask, and SampleInBall operations.

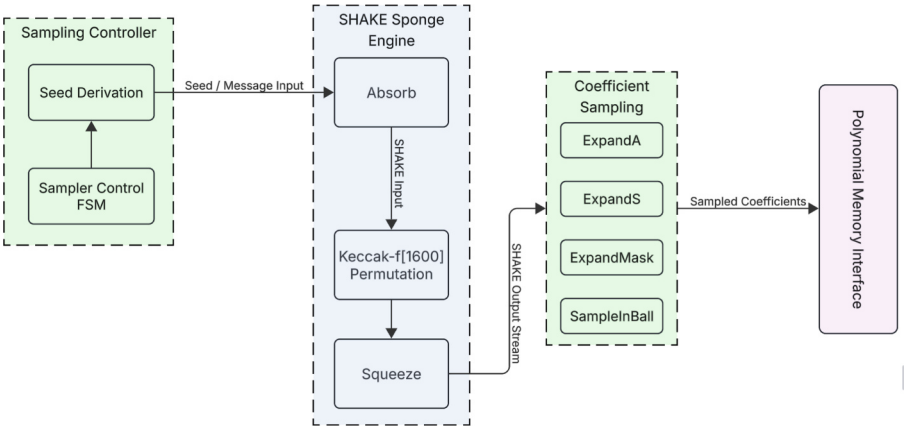

Fig. 5. The architecture for the polynomial sampling module.

The Sampling Controller operates the overall sampling process. It includes a Seed Derivation unit and a Sampler Control finite-state machine FSM. The Seed Derivation unit prepares the deterministic seeds required by the SHAKE engine based on the input parameters, such as the public seed, secret seed, message digest, and nonce values specified in the Dilithium algorithms. These seeds

are forwarded to the SHAKE engine during the absorption phase. The sampler control FSM then selects and activates the appropriate sampling procedure depending on the current algorithm phase (e.g., matrix expansion during key generation or mask sampling during signing).

The pseudorandom streams used for coefficient generation are produced by a SHAKE sponge engine based on the Keccak-f[1600] permutation. The engine follows the standard sponge construction with absorb, permutation, and squeeze phases. After absorbing the input, the permutation is applied to update the internal state, and the squeeze phase generates a pseudorandom output bitstream. This stream serves as the entropy source for the coefficient sampling unit. In the proposed implementation, the engine can be instantiated as SHAKE128 or SHAKE256.

The coefficient sampling unit converts the SHAKE output stream into polynomial coefficients according to the Dilithium specification. The ExpandA operation deterministically expands a seed into public matrix A. The ExpandS operation generates secret vector polynomials with small coefficients, while ExpandMask produces masking polynomials used during the signing process. The SampleInBall procedure generates a sparse challenge polynomial with a fixed Hamming weight. The sampled coefficients are written to the polynomial memory interface, allowing subsequent modules such as the polynomial arithmetic unit to access the generated data.

4 FPGA Edge Computing Prototype

This section presents the prototype FPGA-based edge computing platform developed for integrating a CRYSTALS-Dilithium post-quantum cryptographic accelerator on the Xilinx Kria KV260 platform. The prototype is implemented on the Xilinx Zynq UltraScale+ MPSoC, where the system is divided into two main parts: the Processing System (PS) and the Programmable Logic (PL). The accelerator is described in Verilog HDL and synthesized using Xilinx Vivado, while the embedded ARM Cortex-A53 processor in the PS serves as the host processor to control and interact with the hardware accelerator rewrite this paragraph with the same length but change the expression way.

Figure 6 illustrates the overall architecture of the proposed system. In the Processing System, the ARM Cortex-A53 processor running PYNQ is responsible for parameter initialization, user interaction, data pre-processing, memory read/write operations, and validation of hardware outputs against a golden software model. The processor communicates with the accelerator in the PL through the M_AXI_HPM0_FPD AXI master interface, which is connected to AXI SmartConnect. Inside the PL, the AXI SmartConnect links both the Dilithium accelerator core and the AXI BRAM Controller, while the dual-port RAM acts as a shared on-chip memory buffer between the processor and the accelerator.

The Dilithium accelerator core consists of several functional modules. An AXI slave interface provides the control and status interface between the host processor and the accelerator. A Control FSM coordinates the execution flow

of the core and dispatches operations to the cryptographic engines. The core includes dedicated KeyGen, Sign, and Verify engines, which implement the main Dilithium operations. These engines rely on an Encoder/Decoder block for packing and unpacking keys, signatures, and intermediate data, a Polynomial Arithmetic Unit for polynomial and modular arithmetic operations, and a Polynomial Sampling block for generating the random and structured polynomials required by the Dilithium algorithm.

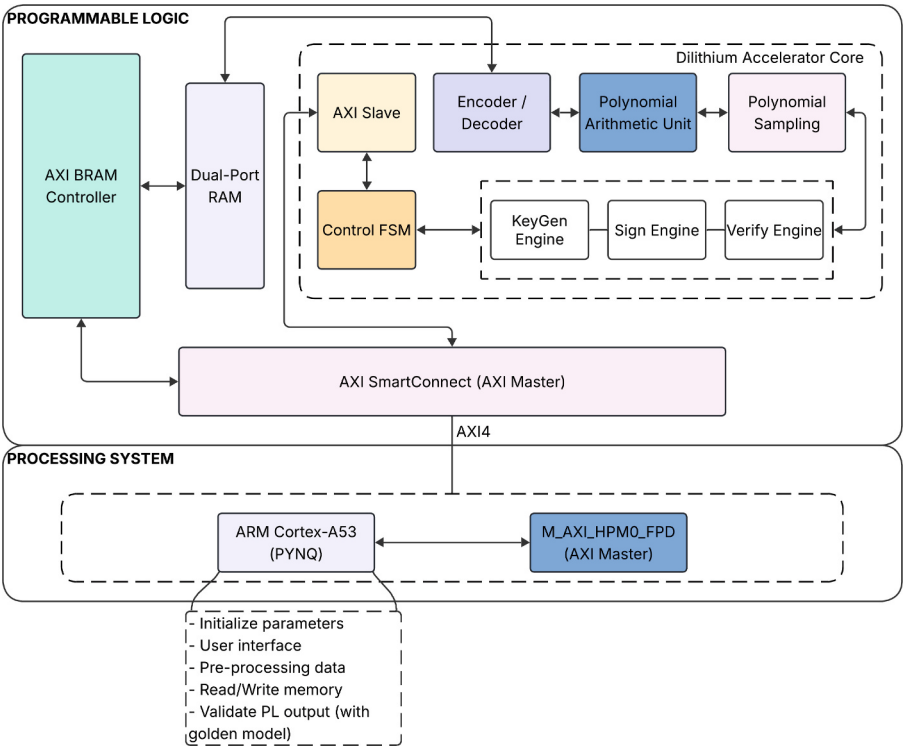

Fig. 6. The prototype system with Xilinx Kria KV260 board.

During operation, the host processor initializes the required parameters and input data, stores them in the shared BRAM, and triggers the accelerator through the AXI slave control interface. The Control FSM then activates the corresponding KeyGen, Sign, or Verify engine, while the arithmetic, encoding/decoding, and sampling modules support the requested computation. After execution is completed, the generated outputs are stored back into memory and read by the host processor for post-processing or correctness verification.

5 Experiments

This section presents experiments to evaluate the effectiveness of the CRYSTALS-Dilithium accelerator core and the FPGA-based prototype systems. The experiment results are then examined, comparing the system's performance with the accelerator against using only software on the ARM processor. In addition, power consumption will be evaluated, analyzed, and compared.

5.1 Synthesis Results

The CRYSTALS-Dilithium accelerator core, developed by Verilog-HDL, and the entire system are synthesized and configured on the Kria board by Xilinx Vivado 2024.2. The results of the synthesis reveal that the system can operate at 133 MHz and requires 3.71 W of total power consumption. Table 1 presents the usage of hardware resources for the system.

Table 1. Hardware resources usage for the proposed system

Resources	Amount	Percentage	Available
Look-up tables (LUTs)	97,215	83%	117,120
Flip-flop (FFs)	64,913	27.71%	234,240
DSP	48	3.85%	1,248
Block RAMs (BRAMs)	89	61.8%	144
$F_{max} = 133$ MHz		$P = 3.71$ W	

As presented in Table 1, the proposed system uses 97,215 LUTs (83% of the available resources) and 64,913 FFs (27.71%), indicating that the design is primarily logic-intensive. The relatively high utilization of LUT results mainly from the implementation of polynomial arithmetic units and multiple SHAKE sponge cores used for polynomial sampling. In contrast, the usage of DSP blocks and BRAMs remains moderate. This shows that the proposed architecture efficiently leverages FPGA logic resources while maintaining reasonable consumption of dedicated hardware blocks.

5.2 Simulation Results

Figure 7 shows the timing waveform of the butterfly core 1× 1 operating in *FORWARD_NTT* mode. When `validi` is asserted, the input coefficients and twiddle factors are processed through three pipeline stages: twiddle multiplication, modular reduction, and butterfly addition/subtraction. After the pipeline latency, the output coefficient pairs are produced sequentially, and `valido` indicates valid output data.

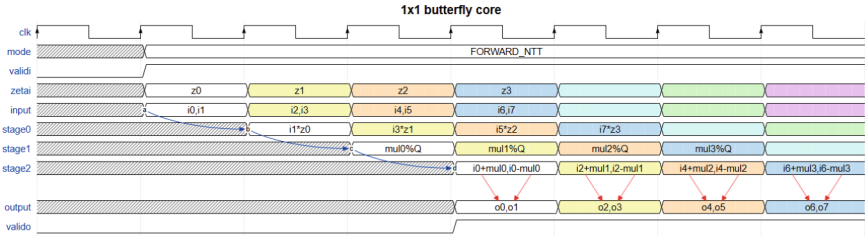

Fig. 7. The waveform for simulation 1×1 butterfly core.

Following the timing behavior of the 1×1 butterfly core shown previously, Fig. 8 presents the waveform of the proposed 2×2 radix-2 butterfly core operating in *FORWARD_NTT* mode. In this architecture, the computation is performed in two butterfly stages. When `validi` is asserted, the input coefficients and the corresponding twiddle factors are first processed by two parallel butterflies (BF0 and BF1), producing intermediate results t_0, t_1, t_2, t_3. Before being transferred to the second stage, the intermediate outputs are rearranged by swapping the positions of t_2 and t_3 to match the required input ordering of the following butterfly units. The reordered values are then processed by butterflies BF2 and BF3 using the corresponding twiddle factors. Due to the pipelined structure, the intermediate results appear first in the output of the first stage, followed by the final output coefficients generated by the second stage. The signals `valido_bf01` and `valido_bf23` indicate the availability of valid intermediate and final results, respectively.

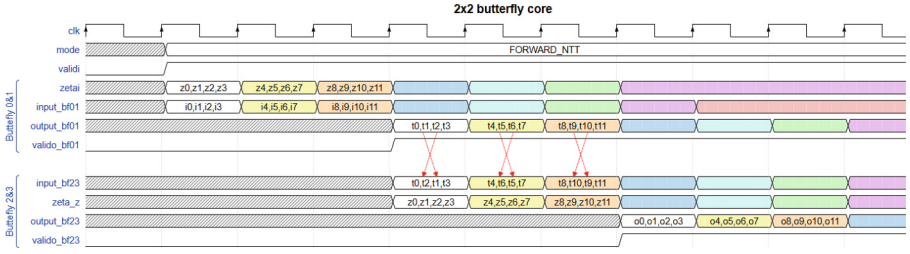

Fig. 8. The waveform for simulation 2×2 butterfly core.

For polynomial sampling, Fig. 9 illustrates the timing waveform of the module `ExpandA`, chosen as a representative example among the four polynomial generation procedures. The SHAKE engine continuously produces pseudorandom outputs (shake_valid, shake_data), which are processed to generate candidate coefficients. Due to the rejection sampling procedure, invalid candidates are discarded, causing the coefficient index (coeff_idx) and the packing counter (coeff_cnt) to temporarily stall. Valid coefficients are accumulated and four coefficients are packed into a single memory word before being written into polynomial

memory when write_en is asserted. After all coefficients of the current polynomial are generated, the polynomial index (poly_idx) increases and the sampling process continues for the next polynomial.



Fig. 9. The waveform for simulation ExpandA module.

5.3 Performance Evaluation and Comparison

We evaluate the performance of the proposed processor using message sizes ranging from 32 bytes to 20 kB for the signing and verification procedures. The evaluation compares two implementations: (1) A hardwired ARM core operating at 1.33 GHz running the reference Dilithium implementation written in C (Software); and (2) the proposed accelerated system, which integrates a dedicated CRYSTALS-Dilithium accelerator core (FPGA).

Table 2 summarizes the execution time of the proposed accelerator system compared to a software implementation running on an ARM processor operating at 1.33 GHz. The proposed design achieves speedups of up to 3.66×, 4.56× and 4.18× for the key generation, signing and verification phases, respectively.

The main performance improvement originates from the hardware acceleration of polynomial arithmetic operations, including number theoretic transform (NTT), inverse NTT, and polynomial multiplications. These operations dominate the computational workload in Dilithium and benefit significantly from the parallel datapath implemented in the accelerator. For evaluating polynomial arithmetic modules, a standard 1×1 polynomial matrix is used as the baseline input to measure the latency of NTT, INTT and polynomial multiplication operations. Despite the substantial acceleration of polynomial arithmetic, the overall speedups of the main phases remain moderate. This is primarily because several components of the Dilithium algorithm, such as the rejection sampling procedures and the SHAKE hash function, are less amenable to hardware acceleration while accounting for a considerable fraction of the total execution time.

Table 3 shows that the proposed NTT architecture achieves competitive latency compared to recent FPGA-based implementations. In particular, this work obtains 1.93 μs for both NTT and INTT, which is faster than most previous designs on Zynq-7000, Artix-7, and ZCU-series platforms in the literature and is close to the best reported results. For polynomial-wise multiplication, the proposed design achieves 0.48 µs, outperforming the compared works reporting

Table 2. Execution time (μs) of each scheme

Scheme	FPGA		Software
	Exec. time	Speed-up	
ExpandA	238.71	2.79×	665.26
ExpandS	47.96	2.68×	128.66
ExpandMask	27.98	3.45×	96.63
SampleInBall	1.48	3.29×	4.87
Foward NTT	1.93	11.79×	22.76
Inverse NTT	1.93	13.56×	26.18
Polynomial Multiplication	0.48	11.10×	5.33
Key Generation	398.61	3.66×	1461.53
Sign	1356.41	4.56×	6181.28
Verify	392.83	4.18×	1642.41

this metric. These results indicate that the proposed architecture provides an effective balance between lightweight implementation and high computational efficiency for edge-oriented Dilithium acceleration.

Table 3. Performance comparison of NTT design with other implementations

Work	Year	Platform	Latency (μs)		
			NTT	INTT	PWM
Gaoyu Mao [14]	2023	Zynq-7000	8.17	8.17	1.51
Tengfei Wang [24]	2024	PolarFire	1.96	1.96	0.54
Stefano Di Matteo [7]	2024	ZCU104	3.91	3.91	-
Suraj Mandal [13]	2024	Zynq 7000	3.67	3.67	0.92
Suraj Mandal [13]	2024	Artix-7	3.75	3.75	0.94
Suraj Mandal [13]	2024	ZCU102	2.48	2.48	0.62
Rishabh Shrivastava [22]	2025	ZCU104	1.56	1.56	-
This Work	**2026**	**Kria KV260**	**1.93**	**1.93**	**0.48**

6 Conclusion

This paper presented a lightweight FPGA-based hardware accelerator for the CRYSTALS-Dilithium post-quantum digital signature scheme targeting edge computing environments. The proposed architecture integrates a dedicated Dilithium accelerator core within the programmable logic of the Xilinx Kria

KV260 platform, while the ARM Cortex-A53 processor in the processing system acts as the host controller. Using hardware parallelism and optimized polynomial arithmetic units, the proposed design accelerates key cryptographic operations, including key generation, signing, and verification. Experimental results demonstrate that the hardware accelerator significantly improves execution efficiency compared with software-based implementations while maintaining a compact resource footprint suitable for edge devices. These results confirm the feasibility of deploying post-quantum digital signature mechanisms on FPGA-based edge platforms to enhance security in resource-constrained environments.

Acknowledgement. This research is funded by Vietnam National University - Ho Chi Minh City (VNU-HCM) under grant number: A2025-20-01.

References

1. Baseri, Y., Chouhan, V., Ghorbani, A., Chow, A.: Evaluation framework for quantum security risk assessment: a comprehensive strategy for quantum-safe transition. Comput. Secur. **150**, 104272 (2025)
2. Beckwith, L., Nguyen, D.T., Gaj, K.: High-performance hardware implementation of crystals-dilithium. In: 2021 International Conference on Field-Programmable Technology (ICFPT), pp. 1–10. IEEE (2021)
3. Boutin, C.: NIST Releases First 3 Finalized Post-Quantum Encryption Standards (2024). https://www.nist.gov/news-events/news/2024/08/nist-releases-first-3-finalized-post-quantum-encryption-standards
4. Brakerski, Z., Langlois, A., Peikert, C., Regev, O., Stehlé, D.: Classical hardness of learning with errors. In: Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing, pp. 575–584 (2013)
5. Carril, X., Kardaris, C., Ribes-GonzáLez, J., Farràs, O., Hernandez, C., Kostalabros, V., González-Jiménez, J.U., Moreto, M.: Hardware acceleration for high-volume operations of crystals-kyber and crystals-dilithium. ACM Trans. Reconfigurable Technol. Syst. **17**(3), 1–26 (2024)
6. Chen, L., Moody, D., Regenscheid, A., Robinson, A.: Digital signature standard (2023)
7. Di Matteo, S., Sarno, I., Saponara, S.: Cryphtor: a memory-unified ntt-based hardware accelerator for post-quantum crystals algorithms. IEEE Access **12**, 25501–25511 (2024)
8. Halak, B., et al.: A security assessment tool for quantum threat analysis (2024). arXiv:2407.13523 arXiv preprint
9. Hsu, R.-H., Lee, J., Quek, T.Q.S., Chen, J.-C.: Reconfigurable security: edge-computing-based framework for iot. IEEE Network **32**(5), 92–99 (2018)
10. Jin, X., Katsis, C., Sang, F., Sun, J., Kundu, A., Kompella, R.: Edge security: Challenges and issues. arXiv preprint arXiv:2206.07164 (2022)
11. Kundi, D.E.S., Mera, J.M.B., Strub, P.Y., Hutter, M.: High-performance ntt hardware accelerator to support ml-kem and ml-dsa. In: Proceedings of the 2024 Workshop on Attacks and Solutions in Hardware Security, pp. 100–105 (2024)
12. Langlois, A., Stehlé, D.: Worst-case to average-case reductions for module lattices. Des. Codes Crypt. **75**(3), 565–599 (2014). https://doi.org/10.1007/s10623-014-9938-4

13. Mandal, S., Roy, D.B.: Kid: a hardware design framework targeting unified ntt multiplication for crystals-kyber and crystals-dilithium on fpga. In: 2024 37th International Conference on VLSI Design and 2024 23rd International Conference on Embedded Systems (VLSID), pp. 455–460. IEEE (2024)
14. Mao, G., Chen, D., Li, G., Dai, W., Sanka, A.I., Koç, Ç.K., Cheung, R.C.: High-performance and configurable sw/hw co-design of post-quantum signature crystals-dilithium. ACM Trans. Reconfigurable Technol. Syst. **16**(3), 1–28 (2023)
15. Mascelli, J., Rodden, M.: "Harvest Now Decrypt Later": Examining Post-Quantum Cryptography and the Data Privacy Risks for Distributed Ledger Networks (2025)
16. Moody, D., Perlner, R., Regenscheid, A., Robinson, A., Cooper, D.: Transition to post-quantum cryptography standards. Tech. rep, National Institute of Standards and Technology (2024)
17. NIST: FIPS 204: Module-Lattice-Based Digital Signature Standard. Tech. rep., U.S. Department of Commerce (2024). https://doi.org/10.6028/NIST.FIPS.204, Federal Information Processing Standards Publication 204
18. Peikert, C.: A decade of lattice cryptography. Found. Trends Theoret. Comput. Sci. **10**(4), 283–424 (2016)
19. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. J. ACM (JACM) **56**(6), 1–40 (2009)
20. Sha, K., Yang, T.A., Wei, W., Davari, S.: A survey of edge computing-based designs for iot security. Digital Commun. Networks **6**(2), 195–202 (2020)
21. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th Annual Symposium on Foundations of Computer Science, pp. 124–134. IEEE (1994)
22. Shrivastava, R., Ratnala, C.P., Puli, D.M., Banerjee, U.: A unified hardware accelerator for fast fourier transform and number theoretic transform. In: ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 1–5. IEEE (2025)
23. Wang, T., Zhang, C., Cao, P., Gu, D.: Efficient implementation of dilithium signature scheme on fpga soc platform. IEEE Trans. Very Large Scale Integr. VLSI Syst. **30**(9), 1158–1171 (2022)
24. Wang, T., Zhang, C., Zhang, X., Gu, D., Cao, P.: Optimized hardware-software co-design for kyber and dilithium on risc-v soc fpga. IACR Trans. Cryptographic Hardware Embedded Syst. **2024**(3), 99–135 (2024)
25. Xue, T., et al.: Edge Computing for IoT: Novel Insights from a Comparative Analysis of Access Control Models. arXiv preprint arXiv:2405.07685 (2024)
26. Zheng, J., Zhu, H., Song, Z., Wang, Z., Zhao, Y.: Optimized vectorization implementation of crystals-dilithium. IEEE Trans. Dependable Secure Comput. (2025)