IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 22 November 2024, accepted 14 December 2024, date of publication 20 December 2024,
date of current version 30 December 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3520592

RESEARCH ARTICLE

High-Efficiency Multi-Standard Polynomial Multiplication Accelerator on RISC-V SoC for Post-Quantum Cryptography

DUC-THUAN DAM[1,2], (Graduate Student Member, IEEE), TRONG-HUNG NGUYEN[1], (Graduate Student Member, IEEE), THAI-HA TRAN[1,2], (Graduate Student Member, IEEE), DUC-HUNG LE[3,4], (Member, IEEE), TRONG-THUC HOANG[1], (Member, IEEE), AND CONG-KHA PHAM[1], (Senior Member, IEEE)

[1]Department of Computer and Network Engineering, The University of Electro-Communications (UEC), Tokyo 182-8585, Japan
[2]Department of Communications, Faculty of Radio-Electronics Engineering, Le Quy Don Technical University (LQDTU), Hanoi 11917, Vietnam
[3]Department of Electronics, Faculty of Electronics and Telecommunications, University of Science, Ho Chi Minh City 72711, Vietnam
[4]Vietnam National University Ho Chi Minh City, Ho Chi Minh City 71300, Vietnam

Corresponding author: Duc-Hung Le (ldhung@hcmus.edu.vn)

This work is funded by Vietnam National University Ho Chi Minh City (VNU-HCM) under grant number DN2024-18-02.

ABSTRACT Number Theoretic Transform (NTT) enables speeding up polynomial multiplications, thereby accelerating the implementation of lattice-based post-quantum cryptography (PQC) algorithms. Currently, the standardized PQC algorithms FIPS 203 (CRYSTALS-Kyber), FIPS 204 (CRYSTALS-Dilithium), and the one in the process of being standardized FIPS 206 (FALCON) all use the NTT to perform polynomial multiplication. This paper proposes a high-speed, low-complexity, and run-time configurable accelerator that supports all three standards. Firstly, we propose a unified design using four parallel radix-2 butterflies targeting a high-speed polynomial multiplier. With a unified design, the accelerator performs NTT, inverse NTT (INTT), point-wise multiplication (PWM), and matrix-vector polynomial multiplication. Secondly, we propose a compact, configurable reordering unit for effective coefficient processing in high-parallelism. As a bonus, the required memory size is minimal, and the memory access pattern is straightforward. Finally, we present a RISC-V SoC architecture with a loosely coupled accelerator through register-map communication and the data flow to accelerate NTT-based operation in software. The FPGA implementation results show that the achieved speed for NTT/INTT/PWM executions is 224/224/64 clock cycles (CCs) for Kyber, 512/512/128 CCs for Dilithium, 576/576/128 CCs for FALCON-512, and 1280/1280/256 CCs for FALCON-1024, respectively. The Area×Time Product (ATP) results also show superiority over other algorithm-specific and configurable designs, achieving improvement up to 82%, 63%, 79%, and 50% for Kyber, Dilithium, FALCON-512, and FALCON-1024, respectively. The SoC implementation results show that the NTT-based operations have improved by up to 5.29×, 27.49×, 56.79×, and 58.91× in software; and speed-up up to 10.53×, 9.81×, 9.57×, and 9.99× for the considered algorithms compared to previous SW/HW works on RISC-V platforms.

INDEX TERMS RISC-V, number theoretic transform, polynomial multiplication, post-quantum cryptography, software/hardware, FIPS 203, FIPS 204, FIPS 206.

I. INTRODUCTION

Since introducing Shor's algorithm [1] in 1994, the risk of public key encryption algorithms being attacked by quantum computers has become increasingly evident. Therefore, the

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

TABLE 1. The specifications and status of PQC lattice-based cryptography standards by NIST.

Standards	Status	Algorithms	n	q	$\lceil \log_2(q) \rceil$
FIPS203	Released	Kyber	256	3329	12
FIPS204	Released	Dilithium	256	8380417	23
FIPS206	Draft	FALCON - 512	512	12289	14
		FALCON - 1024	1024	12289	14

American National Institute of Standards and Technology (NIST) has initiated a standardization process to standardize algorithms against attacks by quantum computers, also known as post-quantum cryptography (PQC) [2]. Currently, NIST has released the Module-Lattice-Based Key-Encapsulation Mechanism Standard (ML-KEM) based on the CRYSTALS-Kyber algorithm with Federal Information Processing Standards (FIPS) 203 [3], and the Module-Lattice-Based Digital Signature Standard (ML-DSA) based on the CRYSTALS-Dilithium algorithm with FIPS 204 [4].

The function of ML-KEM is to establish a shared secret key between two parties communicating through a public channel. The security of ML-KEM is based on the presumed hardness of the Module Learning with Errors (MLWE) problem, which is a generalization of the Learning with Errors (LWE) problem. ML-KEM involves three primary operations: key generation, encapsulation, and decapsulation. Each function uses calculations in the NTT domain with a matrix of polynomials, whose size is determined by the security level. Dilithium is a digital signature algorithm. The security of Dilithium is based on MLWE and Module Short Integer Solution (MSIS). Similarly to Kyber, the key generation, signing, and verification of Dilithium are also based on Point-wise Multiplication (PWM).

The FIPS 206 standard for lattice-based digital signatures is based on the FALCON algorithm [5]. The security of FALCON is based on the hardness of the Short Integer Solution (SIS) problem over the Number Theory Research Unit (NTRU) lattices. The advantage of Dilithium is more straightforward over the integer domain. FALCON is faster for verification and works with both integers and complex numbers. However, some internal algorithms in FALCON's public key generation and key pair generation also use Number Theoretic Transform (NTT) to speed up implementation. Table 1 shows the status and related parameters of LBC standards, in which two out of three were released. The bottleneck of the lattice-based algorithm is polynomial multiplication, which is also the most time-consuming process. The most classical method is ordinary multiplication using a schoolbook algorithm with $O(n^2)$ complexity, where n is the degree of the polynomial. Several methods reduce the complexity for low-degree cases using Karatsuba and Ofman [6] and Toom [7], Cook [8] algorithms. Using NTT effectively reduces the computational complexity of polynomial multiplication to $O(n \log n)$ for the high-degree polynomials.

Researchers generally apply three methods to accelerate NTT operations, including hardware (HW), software (SW), and software/hardware (SW/HW). Hardware designs are applied to achieve optimal performance. These are typically implemented on Field-Programmable Gate Array (FPGA) and Application-Specific Integrated Circuit (ASIC) platforms [9]. The advantage of the HW method is achieving very high performance, which is reflected in the execution speed of the operation. Currently, many different architectures are being applied to accelerate NTT, which are often used for specific algorithms. However, the drawbacks of this method are the long development period and fixed parameters, which lead to limited flexibility. Software is a suitable acceleration method to improve design flexibility and shorten implementation time. Therefore, the SW method can be applied to many different algorithms [10], [11]. Algorithms can be easily implemented on Central Processing Units (CPUs) or Graphic Processing Units (GPUs) for [12], [13] system designs. ARM-Cortex-M3 and M4 are the primary platforms for software development and optimization in embedded devices [14].

System-on-a-Chip (SoC) on embedded systems can handle time-consuming computations by optimizing the instruction set architecture (ISA) [15]. Custom instructions are created to suit the target algorithms, and then these instructions are added to the processor's instruction set to be called when executing the software. This approach is widely used in embedded systems and open source processor families such as the fifth Reduced Instruction Set Computer (RISC-V) with their openness and modular mindset ISA [16]. The advantages of this method are flexibility and short implementation time. Application-Specific Instruction set Processors (ASIP) are a co-design approach that can take advantage of RISC-V. ASIP tightly coupled accelerators are integrated with the processor to reduce communication overhead [17]. However, this architecture requires the processor to support extended instructions.

Another way to combine them is to loosely coupled accelerators without affecting the processor. This makes it easier to attach and detach accelerators. RISC-V cores are not explicitly designed for PQC, so loosely coupled accelerators are reasonable. In this approach, an accelerator is designed on a hardware platform with appropriate interfaces. The accelerator is then loosely coupled to the processor and controlled by the processor through software. This way, the accelerator performs the time-consuming computation, and the processor executes the remaining operations. With the goal of an efficient, high-speed, and flexible design, we aim to propose a configurable accelerator for polynomial multiplication that optimizes performance and area on a RISC-V SoC.

This paper proposes a RISC-V SoC architecture with a coupled accelerator for polynomial multiplication in lattice-based cryptography (LBC) algorithms. The contributions can be summarized as follows.

1) We propose a run-time configurable, high-efficiency, and low-complexity multiplication accelerator for

the PQC standards based on Kyber, Dilithium, and FALCON algorithms. A configurable reordering unit (CRU) is designed to perform the reordering coefficients after each stage without increasing the initial time of the pipeline. The CRU enables low-complexity memory access patterns and eliminates (RAW) conflicts with a minimum memory size of $1n$ and a straightforward control method. We also design a configurable processing element with four radix-2 BUs parallel architecture to accelerate the NTT-based operations. Our accelerator supports NTT/INTT/PWM and matrix-vector polynomial multiplication in a unified design.

2) The results of the Artix-7 and Virtex-7 implementations outperform previous studies. The ATP results show a remarkable improvement compared to state-of-the-art designs. The improvement is up to 82% for Kyber, 63% for Dilithium, 79% for FALCON-512, and 50% for FALCON-1024. This advantage is achieved thanks to the effective resource reuse for NTT-based operations and straightforward control procedures. In addition, applying pipelines allows the design to execute multiple polynomials simultaneously and supports performing matrix-vector multiplication. Specifically, the accelerator requires 224, 512, 576, and 1280 clock cycles (CCs) for NTT/INTT and requires 64, 128, 128, and 256 CCs for PWM in Kyber, Dilithium, FALCON-512, and FALCON-1024 algorithms, respectively.
3) We propose a RISC-V SoC architecture with a loosely coupled accelerator using a Memory-Map Input Output (MMIO) interface. We also propose a data flow to eliminate the cumulative addition delays when performing polynomial multiplication. The system supports multiplications of up to 4×4 polynomials in a single execution. Similarly, we sequentially process multiplications of up to 2×2 for Dilithium, FALCON-512, and 1×1 for FALCON-1024. The results show that the SoC with accelerator significantly improves speed for NTT/INT up to 58.91× and polynomial multiplication up to 13.17×, compared to other RISC-V-based results.

The rest of the paper is organized as follows: Section II presents related background. Section III describes the system, focusing on the accelerator, the SoC architecture, and the data flow. Section IV presents the implementation results and software evaluation, comparing the result with other works. Finally, Section V concludes the paper.

II. RELATED WORKS

The core element to perform polynomial multiplication is NTT. Currently, several designs for accelerating configurable NTT have been proposed. Ye et al. in [19] proposed a pipeline architecture with serial butterfly units (BUs) to speed up computation but at the expense of areas and accumulated latency. The execution between stages is not consecutive due to waiting for the temporary data in memory, which increases the total latency. The reordering is done in memory, increasing the memory size required. Memory conflict is always the bottleneck in parallel NTT execution architectures. Read-after-write (RAW) conflict occurs when reading data from an address where writing data to it is not completed.

To avoid RAW conflict, Mu et al. in [28] proposed an algorithm to access memory efficiently with high complexity. Li et al. in [23] used 3 Random Access Memories (RAM) for each BU, which leads to the amount of RAM required increasing as the number of BUs increases. This approach also requires an appropriate address-generation algorithm. Authors in [18] also introduce configurable architectures and support multiple parameter sets. These designs use multiple BUs that support iterative NTT for acceleration and multiple RAM for data storage, which makes resource utilization unoptimal. Ping-pong buffer is a simple method that is used in some designs to simplify memory access patterns. However, this method leads to double the memory ($2n$) with n as the degree of the polynomial. Liu et al. in [35] has reduced the memory to $1.5n$, but the memory access is still complicated. Another design in [22] avoided using memory but used a register set to store coefficients after each computation, but the number of registers also increased and behaved similarly to a single memory. Thus, a design that ensures a small memory size while avoiding RAW conflicts with simple memory accesses is essential.

Many BU architectures are applied to NTT implementations. Popular designs use a radix-2 BU architecture with a pipeline architecture that processes each pair of coefficients in series [36], [37]. This architecture allows for resource savings but limits execution speed. Single-path Delay Feedback (SDF) and Multi-path Delay Commutator (MDC) are common architectures. MDC architectures often encounter delays between stages, while SDF requires a group of BUs and memories. Implementations that process multiple pairs of coefficients in parallel use 2, 4, 8, 16, and 32 parallel BU architectures [23], [38]. This method allows for faster execution but entails complex dataflow and memory access requirements. The radix-4 BU [39], and split-radix BU [27] architecture are also applied but increase control and memory access complexity. To ensure applicability to considered algorithms and to balance overhead and speedup, we use a radix-2 architecture with 4 parallel BUs following the iterative architecture.

Several studies have proposed accelerator implementations on the SoC. Designs in [40], [41], and [42] implemented Kyber, Dilithium and NewHope algorithms on the ARM-Cortex-M3/M4, PQRISCV and Freedom E310 platforms. The authors proposed a small architecture that implements each modular operation and assigns corresponding instructions. However, these implementations are limited in speed. Alkim et al. [31] proposed a polynomial arithmetic implementation for Kyber and NewHope on the VexRISCV

TABLE 2. Feature profiling of NTT acceleration designs using SW, HW, and SW/HW methods.

Works	Platforms	Methods	Kyber	Dilithium	FALCON		Const. time	PWM	Vec-mat polyMul	Mem. size	RTC	Simple R/W
					512	1024						
[18]	FPGA/ASIC	HW	●	●	●	●	○	●	○	-	●	○
[19]	FPGA	HW	●	○	●	●	●	○	○	$5n-10$	○	●
[20]	FPGA	HW	●	○	○	○	○	●	○	$2n$	○	●
[21]	FPGA/RISC-V	SW/HW	●	●	○	○	●	●	●	$2n$	●	●
[22]	FPGA/ASIC	HW	●	○	○	○	○	○	○	-	○	●
[23]	FPGA	HW	●	●	○	●	●	●	○	-	○	○
[24]	FPGA/ASIC	HW	●	●	○	○	●	●	○	$1n$	●	○
[25]	FPGA	HW	●	○	○	○	●	○	○	$2n$	○	●
[26]	FPGA/ARM	SW/HW	○	●	○	○	●	●	●	-	○	○
[27]	FPGA	HW	●	○	○	○	●	●	○	-	○	○
[28]	FPGA	HW	●	●	●	●	●	●	○	$1n$	○	○
[29]	FPGA	HW	○	○	○	●	●	○	○	$2n$	●	○
[30]	FPGA/RISC-V	SW/HW	●	●	●	●	-	○	○	-	○	-
[15]	FPGA/RISC-V	SW/HW	●	●	○	○	-	○	○	-	○	-
[31]	FPGA/RISC-V	SW/HW	●	○	○	○	○	●	○	-	○	-
[32]	RISC-V	SW/HW	●	○	○	○	○	○	○	-	○	-
[33]	FPGA/RISC-V	SW/HW	●	○	○	○	-	●	○	-	○	-
[34]	ASIC/RISC-V	SW/HW	●	●	○	○	●	●	●	-	●	-
This work	FPGA/RISC-V	SW/HW	●	●	●	●	●	●	●	$1n$	●	●

● Supported ○ Not supported RTC: Run-Time Configurable

platform and FPGA. The author also proposed a custom instruction set extension for finite field arithmetic for small fields. The SW/HW design in [15] is implemented on a 64-bit RISC-V platform CVA6. The design for each operation is also introduced, then attached to the processor and assigned to the corresponding instructions. Wygrzvwalski et al. in [32] implemented an accelerator on a RISC-V core with a loosely coupled method with no ISA. They used control/status registers to control the operation of this accelerator. Matteo et al. in [21] also implemented the SW/HW design with an NTT-based accelerator on a RISC-V SoC with targeted algorithms Kyber and Dilithium.

The related research shows that SW/HW design solutions accelerate NTT-based computations when applied to current PQC standardized algorithms. Designs that support a lot of tunable parameters are often very resource-intensive, and resource-efficient designs often have a speed penalty. The profiling data on NTT acceleration on different platforms and methods supporting the algorithm under consideration is shown in Table 2, where **Simple R/W** represents sequential memory access. We can see that each design has different goals based on the advantages of each platform. Currently, there are not many SW/HW designs that can support all three selected standardized standards. The design needs to ensure flexibility and fast software deployment time while ensuring high-speed execution of NTT-based operations with reasonable overhead.

III. BACKGROUND

A. NUMBER THEORETIC TRANSFORM

The two selected LBC algorithms are based on Module-LWE (Kyber, Dilithium), with the commonality being that they all have to perform polynomial multiplication on

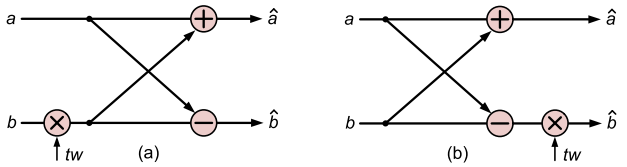


FIGURE 1. The CT Butterfly (a) and GS Butterfly (b) structures.

polynomial ring $\mathbb{Z}_q$. In FALCON, NTT helps to implement faster public key generation and key pair generation. NTT is effective for multiplying two polynomials because it turns ordinary multiplication into PWM and reduces the computational complexity from $O(n^2)$ to $O(n \log n)$. On the polynomial ring $\mathbb{Z}_q$, there exists a primitive $n-$th root of unity ω_n, meaning $q \equiv 1 (\text{mod } n)$. Lyubashevsky et al. [43] introduced the Negative-Wrapped Convolution (NWC) to avoid zero-padding when executing NTT. This method requires q to be satisfying $q \equiv 1 (\text{mod } 2n)$, so that exist $\psi^2 = \omega$ being a primitive $2n-$th root of unity. Thus, the NTT/INTT operations are presented as Eq. 1 and Eq. 2:

$$\widehat{a}_j = \sum_{i=0}^{n-1} \psi^i \omega^{ij} a_i \mod q, j = 0, 1, \ldots, n-1 \quad (1)$$

and

$$a_i = n^{-1} \sum_{j=0}^{n-1} \psi^{-j} \omega^{-ij} \widehat{a}_j \mod q, i = 0, 1, \ldots, n-1 \quad (2)$$

Then, the NTT-based polynomial multiplication is defined as in Eq. 3, where $\circ$ denotes PWM in the NTT domain.

$$y = INTT^{\psi^{-1}}(NTT^{\psi}(a) \circ NTT^{\psi}(b)) \quad (3)$$

Algorithm 1 Barrett Modular Reduction [48]

Input: $c : 0 \leq c < q^2$
Output: $c \pmod q$
1: $k = \lceil \log_2(q) \rceil$ ▷ Smallest k
2: $\mu = \lfloor 4^k/q \rfloor$ ▷ Precomputed
3: $t = c - \left\lfloor \frac{c \times \mu}{4^k} \right\rfloor \times q$
4: **if** $t < q$ **then**
5: **return** t
6: **else**
7: **return** $t - q$
8: **end if**

The divide-and-conquer technique reduces complexity and speeds up the NTT process. Based on that, the calculation of n points can be divided into $n/2$ points. If the number of points n is a power of 2, the entire process can be divided into two input transformations; then, we have an architecture known as a BU. There are two well-known BUs for fast radix-2 algorithms, Cooley-Tukey (CT) [44] and Gentleman-Sande (GS) [45] with the main difference being the position of modular multiplication as shown in Fig. 1 The NTT iterative algorithm allows multiple uses of BU to execute the transform. Let n be a polynomial degree; $n/2$ butterfly operations will be needed for each NTT/INTT stage. Several architectures using parallel BUs have been proposed to increase calculation performance. However, the cost is increased in the hardware area, more complex memory access patterns, and susceptibility to memory conflicts [29]. CT architecture with normal order input and the BU output order will be a bit reversed and opposite for GS. If we use a CT or GS type for the NWC-based NTT/INTT unified architecture, the architecture will require pre-processing and post-processing, leading to costly bit reversed operation. The authors in [46] and [47] proposed a combined architecture with CT for NTT and GS for INTT. Our BU design uses CT/GS architecture for NTT/INTT implementation to simplify the control process while saving hardware area.

Modular reduction operations in a BU include addition, subtraction, and multiplication, in which polynomial multiplication is the most time-consuming operation.

Several algorithms to speed up the implementation of modular reduction are Montgomery [49], Barrett [48], and K-RED [50]. K-RED is suitable for multiplying a number by a constant. To perform modular multiplication for NTT/INTT/PWM, we use Barrett modular reduction since the two input factors are arbitrary. The Barrett algorithm was introduced to speed up modular reduction multiplication, which is described in Algorithm 1. Let a, b be two integers satisfying $0 \leq a, b < q$, and let c be the product of two numbers $c = a \times b$. Therefore, c will be in the range $[0, q^2)$. Barrett's algorithm is applied to reduce c to the range $[0, 2q)$, and the final result is achieved simply with one subtraction. There will be different constants k and μ for each algorithm, which is described in Section IV.

B. RISC-V SYSTEM-ON-A-CHIP

RISC-V ISA follows the load-and-store architecture. Configure modification is implemented by using extensions, with the essential part being the integer (I) instruction set, followed by multiplication (M), atomic architecture (A), floating point (F), and other extensions. The RISC-V instruction set is defined on a 32-bit or 64-bit address space, and based on the combination of extensions, there are different variations of the RISC-V architecture. Rocket Chip [51] is one of the SoCs built on Rocket core, one of the most popular RISC-V CPUs in the community. It also includes other components of an SoC, such as memory and peripherals, and is interconnected via buses.

The tilelink protocol is an effective method to extend the Rocket SoC architecture. Rocket chip SoC supports both Memory-Map Input Output and Rocket Custom Coprocessor (RoCC) in implementing an accelerator. MMIO is also called a Tilelink-Attached accelerator, and RoCC is known as a tightly-coupled one. To design the accelerator for NTT-based multiplication, we used a loosely coupled accelerator on a RISC-V SoC using the MMIO approach. Firstly, we designed a high-speed, configurable, low-resource, low-complexity accelerator. Then, we connected this accelerator to the RISC-V processor via PBUS. Our research uses the RISC-V rv32imac CPU, which supports integer and multiplication. We also use the *riscv gcc toolchain* to compile the C program to the binary file, which runs on the CPU.

IV. PROPOSED SoC ARCHITECTURE

This section presents the proposed accelerator architecture in detail, following the top-down manner. Firstly, we introduce the Rocket SoC architecture with a coupled accelerator. Then, we describe the accelerator a rchitecture with a configurable processing element (CPE) for accelerating NTT/INTT/PWM and matrix-vector multiplication. Next, we analyze the CRU architecture with a detailed example. Finally, we propose a data flow in software to exploit the accelerator's operation.

A. ROCKET RISC-V SoC

We propose a RISC-V SoC architecture with an accelerator that acts as a peripheral on PBUS. This architecture includes a Rocket Core rv32imac and other system components as depicted in Fig. 2. SoC configuration includes 4 KB Instruction cache, 4 KB Data cache, and 256 MB memory. In addition, there are other basic components on PBUS, such as SPI for booting and loading SW programs from memory cards, UART for reporting results during the evaluation process, and boot ROM containing the startup program. Tilelink is a popular bus protocol in the RISC-V community, with features described in [52]. In this design, we use Tilelink Un-cached Lightweight to implement the accelerator. The accelerator is attached to PBUS through communication between the Primary interface on PBUS and the Secondary interface on the accelerator through two channels: Channel A (CPU sends a request) and Channel D (Accelerator responds).

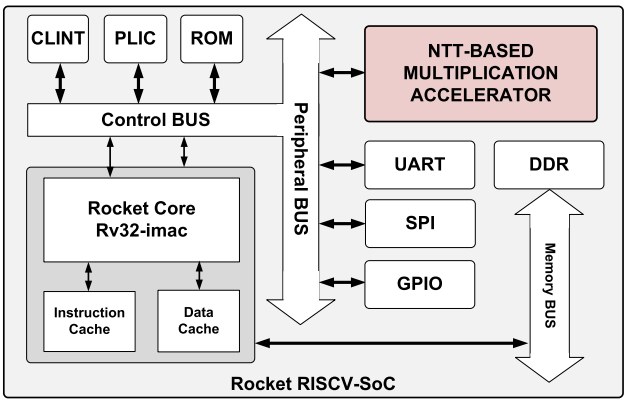


FIGURE 2. Proposed SoC architecture with multiplication accelerator.

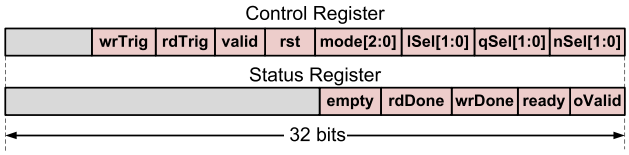


FIGURE 3. The system control and status registers.

We write a wrapper in Scala language using the Chisel library to use this interface, which supports control through Tilelink interfaces.

The CPU controls the accelerator through the **mode** signal to select the working mode, the **nSel** and **qSel** signals to select the parameter set (n, q) and **lSel** to select the polynomial number to be processed. The CPU writes these values to the control register to select the operating mode of the acceleration speed. Fig.3 describes the accelerator's control and status registers. On the register, there are signals to control the operation of the accelerator, including **rst** to reset, the **valid** signal to start the calculation process, and **wrTrig/rdTrig**, which are triggers to perform reading/writing data between the CPU and the accelerator. In addition, there are also signals indicating the status of the accelerator on the status register. The fields in the status register are used to signal the states of the accelerator during execution.

B. CONFIGURABLE POLYNOMIAL MULTIPLICATION ACCELERATOR

Previous configurable accelerator designs used BRAMs to store data and many BUs to execute NTT. The design in [18] can support several algorithms but utilizes a lot of system resources, leading to an increase in size as the number of BUs increases. Other designs in [23] and [29] use a group of BRAMs to store polynomial coefficients to increase execution speed but require complex memory access patterns. The accelerator in [19] uses one BU to save the area and performs reordering on BRAMs to simplify memory access, which leads to larger size memory required. It requires the size of $2n$ for coefficient storage and $2n - 8$ for reordering. To reduce memory usage while ensuring a simple memory access pattern, we designed a configurable accelerator, as shown in Fig. 4.

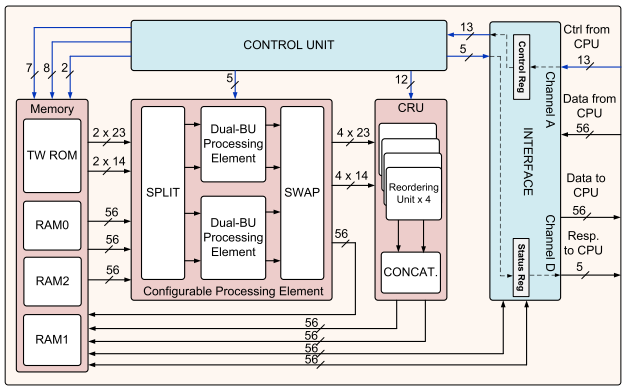


FIGURE 4. Proposed NTT-based multiplication configurable accelerator top-view architecture.

Our design follows previous architectures but solves the issues of memory access and saves resources. The multiplication accelerator consists of four main components. First, the Configurable Processing Element (CPE) performs NTT-based operations by inputting coefficients from RAMs and constants from ROM. Second, the configurable reordering unit (CRU) reorders the output of the CPE in the order required by the next stage. The output of the CRU is the reordered coefficients, which are then written into memory. The third component is the memory, with RAMs storing the coefficients and temporary results and ROM storing the twiddle factors. Finally, the Control Unit (CU) controls the entire accelerator operation based on the data in the control and status registers.

1) CONFIGURABLE PROCESSING ELEMENT

The proposed architecture of CPE, which are shown as a module in Fig. 4, includes two DPEs. To make the design configurable and low area, each BU in DPE is designed according to the CT/GS architecture for NTT/INTT to eliminate pre-processing, post-processing, and bit reversion. According to the algorithm parameters, each coefficient requires 12, 14, and 23 bits for Kyber, FALCON, and Dilithium, respectively. Since the DSP48E1 does not support 23×23 bit multiplication, to reuse resources, we design to use 2 DSP for performing two 14×14 bit multiplication or to perform one 23×23 bit multiplication. Fig. 6(a) illustrates the multiplier architecture. The 23×23 bit multiplication can be rewritten as $(23 \times 14_H) \ll 9 + (23 \times 9_L)$ where H and L represent the high and low significant bits of a 23-bit number, respectively.

Therefore, with two DPEs, we can process 4 pairs of coefficients for Kyber and FALCON in parallel or 2 pairs of coefficients for Dilithium with only 4 DSPs. The input size of the two DPEs now is $2 \times (6 \times 14)$ or $2 \times (3 \times 23)$ bits, including coefficients from RAMs and twiddle factors from ROMs. Therefore, each 56-bit value output from memory must be split into four 14-bit integers or two

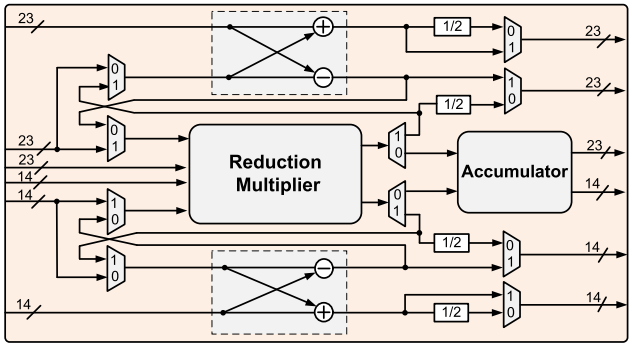


FIGURE 5. The proposed architecture of a dual-BU processing element.

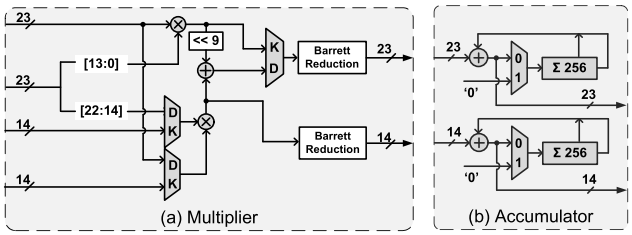


FIGURE 6. The proposed modular multiplier and accumulator architecture.

23-bit integers to get the corresponding coefficients of BUs. The splitting is taken from the lower bits. The selection of multiplier parameters is done by the multiplexer with the selection signal **qSel**. For modular multiplication, we use Barrett reduction with chosen constants k so that $2^k > q$. In this algorithm, we take the value $k = \lceil \log_2(q) \rceil$ as the smallest value of k. The value μ is a pre-computed value given by $\mu = \lfloor 4^k/q \rfloor$. We have the pair of values (k, μ) as (12, 5039), (14, 21843) and (23, 8396807) for the Kyber, FALCON, and Dilithium algorithms, respectively. The division by 4^k is a right shift of $2k$ bits, and multiplication by μ and multiplication by q is performed by addition and subtraction instead of multiplication. For example, in the case of Dilithium, $q = 8380417$, then $k = 23$, $\mu_D = 8396807$. The constant μ_D can be rewritten as $8396807 = 2^{23} + 2^{13} + 2^3 - 1$ and modulus q as $8380417 = 2^{23} - 2^{13} + 1$. So the above multiplications become Eq. 4.

$$\begin{aligned} c \times \mu_D &= (c \ll 23) + (c \ll 13) + (c \ll 3) - c \\ c \times q_D &= (c \ll 23) - (c \ll 13) + c \end{aligned} \tag{4}$$

For the parameter sets of Kyber and FALCON, the calculation is also performed similarly as presented in Eq. 5 and Eq. 6 with μ_K and μ_F are corresponding to Kyber and FALCON. The architecture of the Barrett reduction module is shown in Fig. 7. The 46-bit input c is the product of two 23-bit numbers, which cover different parameter sets. The output is selected by the **qSel** signal and the required number of bits c mod q is extracted for each algorithm.

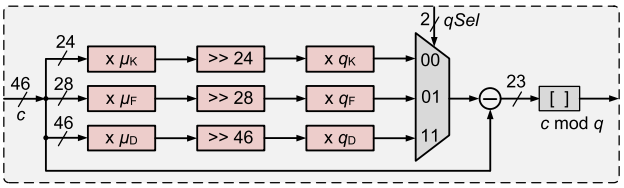


FIGURE 7. Barrett reduction module architecture.

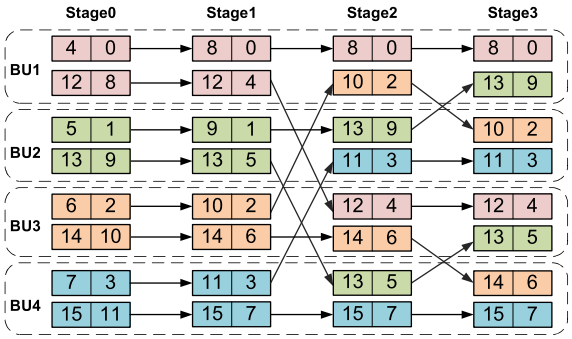


FIGURE 8. Illustration of coefficient swap with 16-point NTT.

$$\begin{aligned} c \times \mu_K &= (c \ll 12) + (c \ll 10) - (c \ll 6) - (c \ll 4) - c \\ c \times q_K &= (c \ll 11) + (c \ll 10) + (c \ll 9) + c \end{aligned} \tag{5}$$

$$\begin{aligned} c \times \mu_F &= (c \ll 14) + (c \ll 12) + (c \ll 10) \\ &\quad + (c \ll 8) + (c \ll 6) + (c \ll 4) + (c \ll 1) + c \\ c \times q_F &= (c \ll 13) + (c \ll 12) + c \end{aligned} \tag{6}$$

The values q, k, μ are selected appropriately via the multiplexers by the selection signals **nSel**, **qSel** in the control register. For INTT, the multiplication by n^{-1} is performed by dividing by 2 after each stage and reusing the twiddle factor for the INTT process. This allows the architecture to perform constant-time NTT/INTT. Fig. 5 demonstrates the architecture of a DPE with 2 BUs in parallel. The twiddle factors are loaded from ROM, and the operation mode NTT/INTT/PWM is selected through multiplexer using **mode** field in the control register. The output of each stage is passed through a swap circuit before being sent to the reordering unit. With a 4-BUs parallel architecture, the outputs at stages $\log(n) - 3$ and $\log(n) - 2$ will be swapped to ensure that the input at the next stage matches the corresponding DPE. Fig. 8 illustrates an example of swapping the outputs at the last two stages in the 16-point NTT case, with the numbers in the figure indicating the output index of each stage. With such a parallel architecture, the NTT/INTT process is performed iteratively, resulting in 224 CCs for Kyber, 512 CCs for Dilithium, 576 CCs for FALCON-512, and 1280 CCs for FALCON-1024, plus 4 CCs for initial pipeline filling.

For performing PWM with FALCON and Dilithium algorithms, the multiplication of two polynomials is performed by multiplying each pair of corresponding coefficients. For

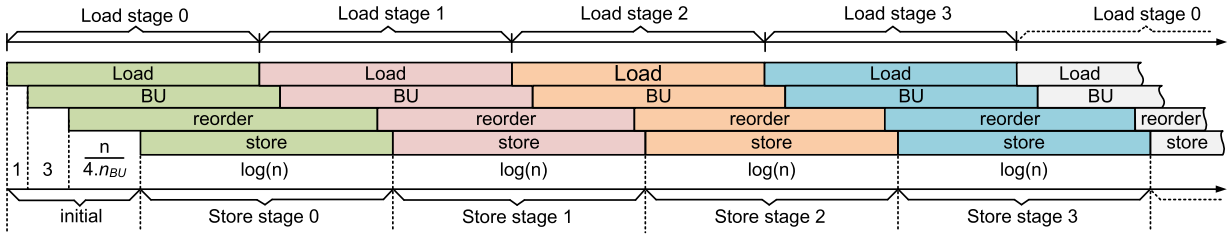


FIGURE 9. Accelerator pipelined NTT/INTT execution.

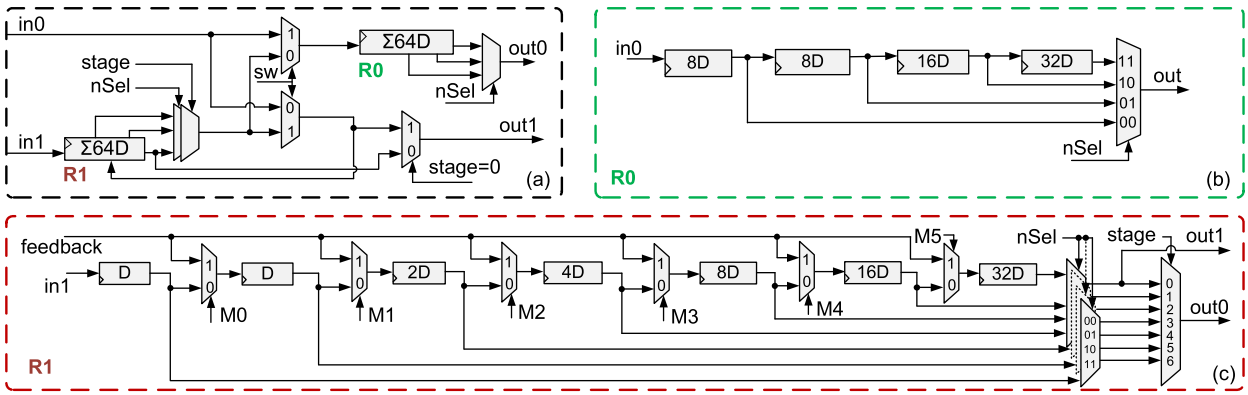


FIGURE 10. The proposed configurable reordering unit (a) with shift register R0 (b) and shift register with feedback R1 (c) architectures.

the Kyber algorithm, the multiplication requires 5 modular multiplications as described in Eq. 7

$$\begin{aligned}\widehat{y}_{2i} &= \widehat{a}_{2i} \cdot \widehat{x}_{2i} + \widehat{a}_{2i+1} \cdot \widehat{x}_{2i+1} \cdot \zeta^{2br(i)+1} \\ \widehat{y}_{2i+1} &= \widehat{a}_{2i} \cdot \widehat{x}_{2i+1} + \widehat{a}_{2i+1} \cdot \widehat{x}_{2i}\end{aligned} \tag{7}$$

where the indices $2i$ and $2i+1$ represent, respectively, the even and odd coefficients of the polynomials in the multiplication. According to [53], the Eq. 7 can be rewritten as Eq. 8 and Eq. 9:

$$\begin{aligned}\widehat{y}_{2i} &= (\widehat{a}_{2i} \cdot \widehat{x}_{2i}) + (\widehat{a}_{2i+1} \cdot \widehat{x}_{2i+1}) \cdot \zeta^{2br(i)+1} \\ &= p_0 + p_1 \cdot \zeta^{2br(i)+1} \qquad (8) \\ \widehat{y}_{2i+1} &= (\widehat{a}_{2i} + \widehat{a}_{2i+1}) \cdot (\widehat{x}_{2i} + \widehat{x}_{2i+1}) \\ &\quad - (\widehat{a}_{2i} \cdot \widehat{x}_{2i} + \widehat{a}_{2i+1} \cdot \widehat{x}_{2i+1}) \\ &= s_0 \cdot s_1 - (p_0 + p_1) \qquad (9)\end{aligned}$$

We use 4 DSPs of CPE to perform multiplication with the architecture in Fig. 6 (a) to implement PWM and architecture in Fig. 6 (b) to implement matrix-vector polynomial multiplication. The PWM implementation for Kyber is taken in 2 steps. The first step calculates the values of p_0, p_1, s_0 and s_1, and the second step calculates the final result. The multiplier is configured to perform this multiplication. For the accumulator, the register length will be $n/4$ for Kyber, FALCON, and $n/2$ for Dilithium. At the last pair of polynomials, the result will be taken directly; at the same time, the 0 bits will be filled in the register to serve the next multiplication. Finally, the products are concatenated in 56-bit form and written to memory. Therefore, with our architecture, PWM for a single polynomial pair would require $n/4$ CC for Kyber and FALCON and $n/2$ CC for Dilithium, respectively. Fig. 9 illustrates the pipeline operation of our design. The sequential execution includes **load**, **BU** execution, **reorder**, and **store**. The accelerator spends 1 CC to load the coefficient pairs and twiddle factors from memory, 3 CCs to execute the BU and $n/(4 \cdot n_{BU})$ CCs to reorder for the first stage. Finally, it takes 1 CC to store the result in memory. Thanks to the architecture of CRU with feedback path, the following stages are executed immediately without incurring any additional CCs delay, which increases latency in previous designs.

2) CONFIGURABLE REORDERING UNIT

To achieve the minimum memory size with a simple memory access pattern, we propose a configurable reordering unit (CRU) to reorder the output of the BU before storing it in RAMs. With an architecture consisting of 4 radix-2 BUs in parallel, there will be eight outputs to reorder at a time. There are 4 reordering units corresponding to 4 BUs. The output of the CRU will be concatenated together on a 56-bit output. Each 56-bit output will contain two 23-bit coefficients for the Dilithium algorithm or four 14-bit coefficients for the remaining algorithms. The reordering unit architecture is based on [38]. However, the initial time in the stages has been eliminated, maintaining the corresponding ordering delays for each algorithm as $n/32$ CCs for Kyber, $n/8$ CCs for

Dilithium, and $n/16$ CCs for FALCON. The CRU is a group of 4 reordering units (RUs).

The architecture of an RU is shown in Fig. 10 (a), with the main components consisting of multiplexers, a configurable shift register, and a configurable shift register with feedback. Fig. 10 (b) describes R0, which is a configurable parallel-in parallel-out (PIPO) register consisting of shift registers connected in series. The latency caused by the ordering is given by $n/(4 \cdot n_{BU})$, where n_{BU} is the number of parallel BUs used for that algorithm. This shift register has a total length of 64, with the output selected by the **nSel** signal. Shift register R1 is made up of a series of shift registers with length given by $1 + 1 + 2 + \cdots + n/(8 \cdot n_{BU})$. Shift register R1 also has a total length of 64, of which one additional feedback path keeps the result latency unchanged throughout the NTT/INTT process, as shown in Fig. 10 (c). The feedback position on R1 is selected by the control bits combination from M0 to M5, and at a time, there is only one M_i with the value of 1, meaning that only one feedback path is selected.

To explain the operation of this CRU, we perform an example with 16-point NTT. Fig. 11 shows the architecture of the CRU with the case $n = 16$. The reordering process with eight input pairs $(in0, in1)$ is assumed to be the CPE output, with the value inside each cell representing the index of the corresponding coefficient. In this case, registers R0 and R1 have a total length of 4, in which R1 has 2 feedback positions and is selected by $M0$ and $M1$. The output pair $(out0, out1)$ will be the pair of coefficients in the correct order needed to be stored in memory. In this example, since $n = 16$, it will take 4 stages to complete the NTT, and 8 CCs will be required to complete one stage. Fig.12 illustrates the waveform of the 16-point NTT in the example. The NTT process will be performed in 32 CCs. At clocks 0 to 3, four pairs of coefficients are loaded into registers R0 and R1. Then, at clocks 4 to 7, the first 4 output pairs of coefficients are pushed out in the desired order, and the next 4 pairs of coefficients are loaded. At the next stage, $(M0, M1) = (0, 1)$ selects the feedback path to R1, and two pairs of coefficients are pushed out every 2 CCs while maintaining the output delay constant. In this way, at each stage, the $(M0, M1)$ value will change to select the feedback location while keeping the ordering and total delay. The address is to be written in the memory of the reordered data, and the write enable signal is also generated. We can see that the output always has a delay of 4 CCs compared to the input. When the first polynomial completes its transformation, the coefficients of the following polynomial are loaded in (number in red colored), and the process continues. This architecture allows for a pipeline architecture without adding any initial latency. With this architecture, the memory access pattern is simple, with sequential reads and writes while avoiding RAW conflicts. Therefore, the memory requirement for each algorithm is always minimal $1n$. The INTT process is also performed similarly but in the reverse order of NTT.

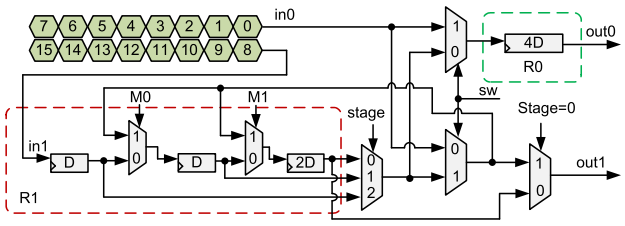


FIGURE 11. The reordering unit in case of 16-point NTT/INTT.

3) MEMORY

Memory optimization is one of the challenges in NTT-based accelerator design. Some designs use the ping-pong method to avoid RAW conflicts [21]. Although this method provides a simple memory access pattern, it doubles the memory size, making memory utilization only 50%. Other designs proposed conflict-free memory architecture to reduce the required memory size [35], [39]. The disadvantage is that memory access patterns are complex and increase as the design scales up. Our design has solved both problems. The minimum required memory size is only $1n$, and the memory access pattern is straightforward, even when the design is expanded. We also use this memory to exchange data with the CPU.

In this design, the memory stores the coefficients of the polynomial to be computed. Typically, a polynomial will need a minimum depth of memory to store coefficients is the number of coefficients, then we say the required memory will be $1n$. Thanks to a reordering unit, the size of RAM0 and RAM1 is $n/2$, and the total memory size for NTT is $n/2 + n/2 = 1n$, the minimum required size. To support 4 pairs of Kyber or FALCON coefficients and 2 pairs of Dilithium coefficients, the memory requires $14 \times 4 = 56$ bits wide to cover all cases. Thus, the depth needed for the memory is $1024/4 = 256$. This memory will be divided into 2 halves to ensure parallel execution. Therefore, the memory for NTT execution includes RAM0 and RAM1 of size 56×128. This memory can store 4 Kyber polynomials, 2 Dilithium polynomials, 2 FALCON-512 polynomials, and 1 FALCON-1024 polynomial.

The memory also includes a RAM2 memory to store the polynomials, which will be multiplied in PWM. This memory will contain polynomial matrix-vector on the NTT domain generated in internal algorithms. Since the design uses 4 multiplications, the width of this memory is also $14 \times 4 = 56$ to ensure parallel multiplication. The depth of RAM2 depends on the number of polynomials in the multiplied polynomial vector. In this design, we use a minimum depth of 256. Thus, RAM2 has a size of 56×256. Fig. 13 illustrates how data is arranged in one address of RAMs. For the Dilithium case, each address will contain 2 coefficients of the polynomial, represented by the red pairs (a_{2i+2}, a_{2i}), (a_{2i+3}, a_{2i+1}), $(\widehat{x}_{2i+1}, \widehat{x}_{2i})$ on RAM0, RAM1, RAM2 respectively. For Kyber and FALCON, each address contains 4 coefficients, as shown in the blue coefficients.

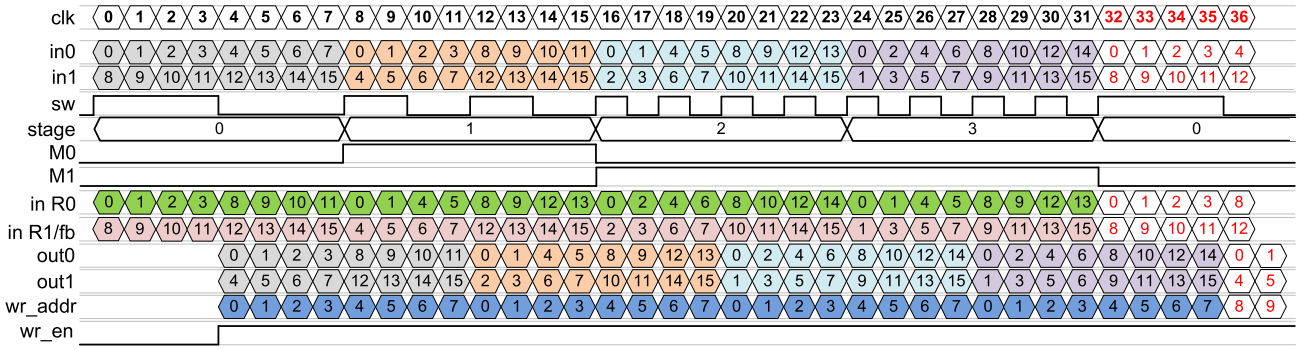


FIGURE 12. The waveform of reordering unit in case of 16-point NTT.

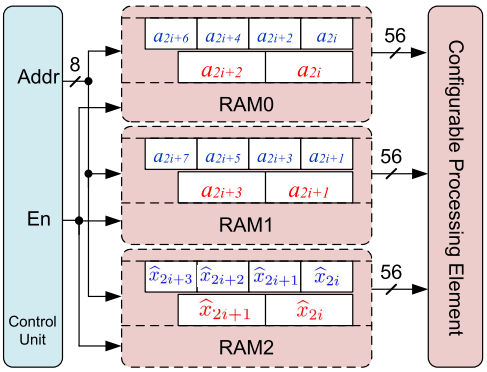


FIGURE 13. Data arrangement on RAMs for Dilithium (coefficients in red) and Kyber/FALCON (coefficients in blue).

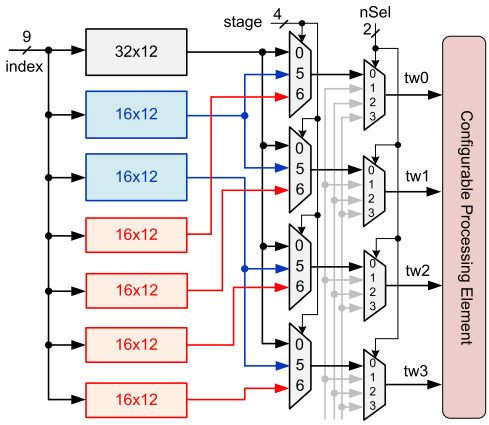


FIGURE 14. The shared and private ROMs in the case of Kyber.

We store the even coefficients on RAM0 and the odd coefficients on RAM1 for PWM without reordering.

This arrangement and the CPE architecture greatly simplify the memory access pattern. Accessing RAM0 and RAM1 is performed sequentially, with the same address for both reads and writes during the NTT/INTT operations. RAM2 is accessed while performing PWM. Accessing RAM2 is also simple, with the same read and write addresses. Depending on the execution mode, the result is written to RAM2 or RAM0 and RAM1. If only PWM is performed separately, it is written to RAM2; if INTT is performed immediately after PWM, the result is written to RAM0 and RAM1.

Based on the parallel 4 BUs architecture, the ROM for storing twiddle factors can be divided into shared and private memory areas to save resources. Typically, the simplest way is to store the twiddle factor for each calculation. This method causes resource waste because it stores the same value multiple times. The number of twiddle factors used for each parameter set is $n - 1$. With the 4 BUs parallel architecture, the twiddle factor values will be different in the last two stages, while the previous stages all share the same twiddle factor value. We propose an architecture that groups the ROMs containing twiddle factors into two parts: shared and private parts.

Fig. 14 illustrates how to divide the ROM area for the case $n = 128$, with the number of stages being $\log_2(128) = 7$ stages. At stage 0 to stage 4, the twiddle factor is loaded from the shared ROM (black colored); at stage 5, the twiddle factor is loaded from 2 shared ROMs (blue colored); and at the last stage, the twiddle for each BU is loaded from the private ROMs (red colored). In short, the shared ROM will contain the twiddle factor for the shared stages, and the private ROMs will store the private twiddle factor for the last stages. This solution helps reuse 50% of resources for twiddle factor storage. The twiddle factors are loaded by inputting the twiddle factor index and selecting the output according to the stage. With this approach, the amount of ROM memory needed to store the twiddle factor will always be minimal.

4) CONTROL UNIT

The Control Unit generates read/write addresses for the memory, generates indexes to load the twiddle factors, and generates control signals for the CRU and CPE based on the control register. The fields in the control register are shown in Table 3. With the proposed architecture, the control logic

TABLE 3. The control signal to select parameter sets and operation mode.

Algorithms	nSel	qSel	lSel	mode			
				NTT	INTT	PWM	polyMul
Kyber	00	00	00,01,10,11	001	000	011	111
Dilithium	01	11	00,01	001	000	011	111
FALCON-512	10	10	00,01	001	000	011	111
FALCON-1024	11	10	00	001	000	011	111

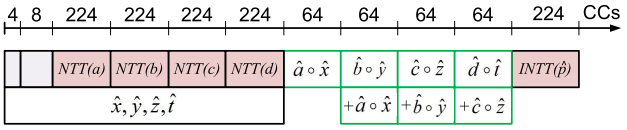


FIGURE 15. The data flow of NTT-PWM-INTT scenario for 2 vectors of 4 polynomials of Kyber.

is straightforward. We use low significant bits (LSB) of a simple counter to generate addresses and control signals for the entire architecture.

- Memory access control: The sequential addresses use $\log(n/(2 \cdot n_{BU}))$ LSB of the counter for generating the address for RAM0 and RAM1. The write address is delayed $n/(4 \cdot n_{BU})$ CCs after the read address. The read/write enabling signals are turned to follow the read/write address. When a polynomial finishes its NTT, the address for the following polynomial is the same as the previous polynomial but with an offset added. This offset is given by $(l-1) \times n$, where l is the order of the polynomial being processed. The indexes for loading the twiddle factors are also generated from $\log(n/n_{BU})$ LSB of the counter. RAM2 is accessed when performing PWM. When only multiplication is executed, the product will still be written back to RAM2 at the same read address. When INTT is performed immediately on the result of multiplication, this result will be saved to RAM0 and RAM1.
- The operations of the entire accelerator are controlled by employing the selection signals on the multiplexers. The **sw** signal in the CRU and the **swap** signal in the CPE are also changed based on the counter value of each corresponding **stage**. The generation of this signal is straightforward, based on the parameters selected by the CPU written in the control register.

C. PROPOSED SOFTWARE DATA FLOW

To take advantage of the pipeline capability of this architecture without increasing latency and memory capacity, we propose a data flow that speeds up the NTT-based operation on the SoC. Then, multiple polynomials are computed per execution. This aligns with LBC algorithms that require numerous polynomial multiplications depending on the security level [5], [54], [55]. Our design allows loading up to 4 polynomials for Kyber, 2 polynomials for Dilithium, 2 polynomials for FALCON-512, and 1 polynomial for FALCON-1024 to the accelerator per execution. In this way, we reduce the data transmission overhead and eliminate the impact of the number of CCs required to fill the pipeline and the CCs latency caused by reordering. For example, the number of the CCs needed to transform NTT/INTT for a polynomial for Kyber is $(224 \times 4 + 4 + 8)/4 = 227$ CCs, similarly required 530 CCs for Dilithium, 594 CCs for FALCON-512 and 1384 CCs for FALCON-1024. We propose two scenarios as follows:

- Performing a single polynomial multiplication before returning the data to the CPU. The accelerator will perform NTT first, then handle PWM for a single pair of polynomials. Lastly, the product is inverted into the normal domain by INTT. For polynomial multiplication, the accelerator requires 64, 128, 128, and 256 CCs for performing PWM for Kyber, Dilithium FALCON-512, and FALCON-1024, respectively. These processes are pipelined, so no additional initial cycles are required for PWM and INTT. Therefore, the number of CCs required will be the sum of the CCs of all three operations. As a result, single polynomial multiplication takes 515, 1170, 1298, and 2884 CCs for Kyber, Dilithium, FALCON-512 and FALCON-1024.
- Executing a matrix-vector polynomial multiplication in a single execution. In this scenario, a multiplication of a vector polynomial and a vector polynomial is performed. The product is cumulatively added before being INNT to obtain the result of matrix-vector multiplication. Fig. 15 illustrates the pipeline execution process for this scenario in the case of performing multiplication between 2 vectors of 4 polynomials. When the multiplication is performed, the polynomial vector to be multiplied $[\widehat{x}, \widehat{y}, \widehat{z}, \widehat{t}]$ is already available in RAM2. This is the vector created on the NTT domain. The accelerator initially takes $4 + 8 = 12$ CCs for the pipeline. Then, each polynomial of the input matrix-vector $[a, b, c, d]$ is transformed into the NTT domain. Following that, point-wise multiplication is performed. After each multiplication of a pair of polynomials, the result is cumulatively added to the product of the two previously calculated polynomials and we get $\widehat{p} = \widehat{a} \cdot \widehat{x} + \widehat{b} \cdot \widehat{y} + \widehat{c} \cdot \widehat{z} + \widehat{d} \cdot \widehat{t}$. Finally, $\widehat{p}$ is inverted to the normal domain before returning the result p to the main program. The required CCs is 1338, 1700, and 1874 for 4×4 multiplication in Kyber, 2×2 multiplication in Dilithium, and 2×2 multiplication in FALCON-512, respectively. It is worth noting that while the accelerator performs the calculation, the CPU can execute other software tasks depending on the algorithm. The CPU can then get the result when necessary.

V. EXPERIMENTAL RESULTS AND COMPARISON

A. IMPLEMENTATION RESULTS

We implemented the proposed accelerator on Xilinx Artix-7 (xc7a100tcsg324-1) and Virtex-7 (xc7vx485tffg1761-2) FPGA with the Xilinx Vivado tool version 2022.2. The

TABLE 4. The implementation results of proposed SoC on Arty-A7 100T FPGA board.

Components	Total LUT	FFs	BRAMs	DSPs
SoC	25319 (100%)	22164 (100%)	9.0	6
polyMul accelerator	3239 (12.79%)	2867 (12.93%)	3.0	4
⌊ CPE	1520	1732	0	4
⌊ CRU	1302	1149	0	0
⌊ Control Unit	18	32	0	0
⌊ Memory	120	0	3.0	0

accelerator consumes 3239 (3254) LUTs, 2867 (2869) FFs, 3 BRAMs, and 4 DSPs at an operating frequency of 115 (171) MHz and power consumption of 0.195 W (0.364 W) for Artix-7 (Virtex-7) platforms. The accelerator's operating frequency is not too high. However, the design goal is to attach to an SoC, which usually has a lower system frequency. Our RISC-V SoC is proposed to operate at 50 MHz. We can improve the accelerator's speed by optimizing the critical paths when working on higher-frequency platforms. On the Arty-A7 FPGA board, the proposed SoC with the configuration uses 25319 LUTs, 22164 FFs, 9 BRAMs, and 6 DSPs. Table 4 shows the results of the SoC resources consumed and occupied by the accelerator. The results show that with an overhead of nearly 13%, the results are remarkable in terms of improvement in computing speed. Then, we evaluate the resources and performance of the accelerator and the speed of pure software on the newly deployed SoC.

Table 5 statistics the number of required CCs when running with pure SW on the system, compared with the number of CCs needed when executing NTT-based operations with the accelerator (SW/HW). The number of CCs is counted when the CPU starts writing data into the memory until it completes NTT/INTT/PWM and returns the final result. We read the value in the *cycle* register at the start and end times, then take the difference to get the required result. The *cycle* register is a read-only register in the RISC-V, which is automatically incremented by one every system clock. Different sets of parameters n, q are investigated on SW and SW/HW. The results show that when running pure SW on SoC, the NTT/INTT/PWM processes take a huge number of CCs, which depends on the degree of polynomial n. When applying SW/HW, the number of required cycles is impressively reduced. The speedup ratio is calculated using the expression shown in Eq.10.

$$ratio_{speedup} = \frac{CC_w + CC_{execute} + CC_r}{CC_{ref}} \tag{10}$$

where CC_w and CC_r are CCs for writing and reading data by CPU, respectively. The $CC_{execution}$ is the CCs of specific operation: NTT, INTT, or polyMul, and the CC_{ref} is the CCs of referenced works. We can see that the speedup efficiency is up to 37.9×/ 38.6× for NTT/INTT in the case of Kyber with $n = 256$, $q = 3329$. The speedup improvement over the accelerator for the Dilithium and FALCON-512 algorithms is 40.4× and 47.8×, respectively. Because the higher the polynomial degree, the larger the number of CCs, the efficiency achieved when using SW/HW with larger n will be better, specifically improving up to 185.2 × /185.0× for FALCON-1024 algorithm with the parameter set $n = 1024$, $q = 12289$. PolyMul process includes NTT, then PWM and INTT. Instead of exchanging data repeatedly, the accelerator can perform the necessary calculations based on the results just calculated. As a result, polyMul achieved remarkable results. The speedup efficiency is 73.7×, 68.3×, 88.2×, and 331.3× for Kyber, Dilithium, FALCON-512, and FALCON-1024, respectively. This impressive result was achieved because the most significant computational parts were handled by the accelerator, and the software's task was to prepare input data and process output results.

B. COMPARISONS AND DISCUSSIONS

We report the implementation results of the design in Tables 6 and 7 and compare them with previous works. For a fair comparison, we report Area-Time Product as the product of total CCs × Equivalent Number of Slices (ENS), where ENS is the sum of the reported SLICES and the equivalent number of slices converted from BRAMs and DSPs. Total time T is the sum of CCs for NTT and INTT. For non-optimal designs, the CCs for INTT are usually larger than NTT because of the need to perform dividing by n at the last stage. Our design solves this problem by dividing by two after each stage when performing INTT. As a result, the CCs for NTT and INTT are equal. Since the studies do not report INTT CCs, we use the NTT CCs instead for fair comparison. We also use the common conversion ratio as described in [38] for the Artix-7 platform. Following that, we count 1 DSP = 100 SLICEs and 1 BRAM = 200 SLICEs. For the results that do not report SLICES, we calculate the number of slices from the number of LUTs and FFs, with the formula SLICEs = LUTs/4 + FFs/8. Thus, the equivalent conversion formula would be ENS = LUTs/4 + FFs/8 + BRAMs × 200 + DSPs × 100.

In our work, we designed a unified design to execute polynomial multiplication for multiple algorithms so that the hardware results for all algorithms are the same. Table 6 shows the hardware implementation results compared to previous works. In the considered algorithms, polynomial multiplication is usually performed between a transformed polynomial and a polynomial (matrix-vector polynomial) generated in the NTT domain. So that, the result of the polyMul execution is the number of CCs required to execute NTT, multiply a pair of polynomials, and INTT. The average CCs required for NTT/INTT are given by Eq. 11.

$$CC_{NTT} = (4 + \frac{n}{4 \cdot n_{BU}} + l \cdot \frac{n \log(n)}{2 \cdot n_{BU}})/l \tag{11}$$

The CCs for PWM is calculated by $CC_{PWM} = (2 \cdot n)/n_{BU}$ for Kyber and $CC_{PWM} = n/n_{BU}$ for Dilithium and FALCON. For reports that only have the CCs of the PWM, the result in the table will be the sum of the CCs of the NTT, PWM, and INTT reported.

TABLE 5. The required clock cycles of the proposed software.

Algorithms	n	q	NTT (CCs)		INTT (CCs)		polyMul* (CCs)	
			SW	SW/HW	SW	SW/HW	SW	SW/HW
Kyber	256	3329	157,535	4,156 (37.9×)	161,145	4,172 (38.6×)	328,680	4,457 (73.7×)
Dilithium	256	8380417	180,348	4,459 (40.4×)	185,152	4,475 (41.4×)	385,448	5,647 (68.3×)
FALCON-512	512	12289	405,320	8,472 (47.8×)	415,113	8,485 (48.9×)	863,254	9,788 (88.2×)
FALCON-1024	1024	12289	3,342,139	18,050 (185.2×)	3,346,139	18,086 (185.0×)	6,913,425	20,866 (331.3×)

* **polyMul** includes NTT, PWM and INTT

TABLE 6. The efficiency of the proposed accelerator compared to previous work.

Works	n_{BU}	Freq. (MHz)	BU Config.	Plat-forms	CCs			LUTs	FFs	BRAMs	DSPs	SLICEs	ENS*	ATP**
					NTT	INTT	polyMul							
Kyber, $n = 256, q = 3329, \lceil \log_2(q) \rceil = 12$														
[18]†	1	174	✓	V7	992	1184	3812	2128	1144	3	8	675	2,075	4,515,200 (+78%)
	8	186	✓	V7	138	176	572	11000	5422	64	12	3,428	17,428	5,472,314 (+82%)
[19]†	7	235	✓	V7	556	-	-	1362	1036	0.5	7	470	1,270	1,412,240 (+30%)
[20]	4	296	✕	V7	306	-	-	982	1151	10.5	5	389	2,989	1,829,498 (+46%)
[21]	1	118	✓	A7	933	933	2131	1243	562	3.5	11	381	2,181	4,069,746 (+76%)
[22]	1	187	✕	A7	896	1024	-	4731	3169	0	0	1,579	1,579	3,031,440 (+68%)
	1	212	✕	V7	896	1024	-	5109	3184	0	0	1,675	1,675	3,216,480 (+69%)
[23]	16	200	✓	X	100	-	-	24186	14756	24	48	7,891	17,491	3,498,200 (+72%)
[24]	2	263	✓	A7	448	448	1152	1315	1280	4.5	2	489	1,589	1,423,520 (+31%)
	4	200	✓	A7	224	224	576	3105	2389	4.5	4	1,075	2,375	1,063,944 (+7%)
[25]	1	278	✕	A7	306	-	-	1070	1071	10.5	5	401	3,001	1,836,842 (+46%)
This work	**4**	**115**	✓	**A7**	**227**	**227**	**515**	**3239**	**2867**	**3**	**4**	**1,168**	**2,168**	**984,329 (0%)**
	4	**171**	✓	**V7**	**227**	**227**	**515**	**3254**	**2869**	**3**	**4**	**1,172**	**2,172**	**986,145 (0%)**
Dilithium, $n = 256, q = 8380417, \lceil \log_2(q) \rceil = 23$														
[18]†	1	140	✓	A7	1052	1318	3688	2119	1058	3	8	662	2,062	4,886,940 (+53%)
	8	117	✓	A7	156	197	552	11000	5422	64	12	3,428	17,428	6,151,996 (+63%)
[23]	4	250	✓	X	284	284	998	5478	4955	6	12	1,989	4,389	2,492,881 (+8%)
	8	220	✓	X	156	156	550	11006	8791	12	24	3,850	8,650	2,698,917 (+15%)
	16	194	✓	X	113	113	385	26520	15171	24	48	8,526	18,126	4,096,561 (+44%)
[24]	2	263	✓	A7	1024	1024	2304	1315	1280	4.5	2	489	1,589	3,253,760 (+29%)
	4	200	✓	A7	512	512	1152	3105	2389	4.5	4	1,075	2,375	2,431,872 (+5%)
[26]	1	172	✕	A7	1405	1405	3079	799	971	4.5	2	321	1,421	3,993,361 (+42%)
[21]	1	131	✓	A7	1074	1074	2412	1302	571	3.5	9	397	1,997	4,289,288 (+46%)
This work	**4**	**115**	✓	**A7**	**530**	**530**	**1170**	**3239**	**2867**	**3**	**4**	**1,168**	**2,168**	**2,298,213 (0%)**
	4	**171**	✓	**V7**	**530**	**530**	**1170**	**3254**	**2869**	**3**	**4**	**1,172**	**2,172**	**2,302,453 (0%)**
FALCON-512, $n = 512, q = 12289, \lceil \log_2(q) \rceil = 14$														
[18]†	1	117	✓	A7	2334	2854	8072	2119	1058	3	8	662	2,062	10,697,656 (+76%)
	8	140	✓	A7	318	391	1100	11000	5422	64	12	3,428	17,428	12,356,275 (+79%)
[19]†	9	234	✓	V7	1074	-	-	2178	1694	3	8	756	2,156	4,631,625 (+44%)
[28]†	1	278	✓	V7	2311	-	5139	489	245	3	3	153	1,053	4,866,388 (+47%)
	2	248	✓	V7	1159	-	2579	1071	557	4	6	337	1,737	4,027,235 (+36%)
This work	**4**	**115**	✓	**A7**	**594**	**594**	**1298**	**3239**	**2867**	**3**	**4**	**1,168**	**2,168**	**2,575,733 (0%)**
	4	**171**	✓	**V7**	**594**	**594**	**1298**	**3254**	**2869**	**3**	**4**	**1,172**	**2,172**	**2,580,485 (0%)**
FALCON-1024, $n = 1024, q = 12289, \lceil \log_2(q) \rceil = 14$														
[19]†	10	232	✓	V7	2104	-	-	2151	1475	2.5	9	722	2,122	8,929,902 (+35%)
[29]†	4	213	✓	A7	1308	-	-	1161	967	3	12	411	2,211	5,784,303 (-1%)
	4	263	✓	V7	1294	-	-	2011	1070	5	12	637	2,837	7,340,862 (+20%)
[28]†	1	278	✓	V7	5127	-	11283	515	306	3	3	167	1,067	10,941,018 (+47%)
	2	256	✓	V7	2567	-	5651	1205	572	4	6	373	1,773	9,101,298 (+36%)
	4	189	✓	V7	1294	-	2849	1467	930	4.5	13	483	2,683	6,943,604 (+16%)
[23]	16	200	✓	X	343	-	-	22648	15030	24	48	7,541	17,141	11,758,554 (+50%)
This work	**4**	**115**	✓	**A7**	**1348**	**1348**	**2884**	**3239**	**2867**	**3**	**4**	**1,168**	**2,168**	**5,845,265 (0%)**
	4	**171**	✓	**V7**	**1348**	**1348**	**2884**	**3254**	**2869**	**3**	**4**	**1,172**	**2,172**	**5,856,049 (0%)**

✓: configurable designs; ✕: algorithm-specific designs, † Tunable q
A7: Arty- A7, V7: Virtex-7, X: Ultra Scale XCKU060

* The Equivalent Number of Slices ENS = LUTs/4 + FFs/8 + BRAMs×200 + DSPs×100.
** The Area-Time Product (ATP) is calculated as Total CCs (NTT + INTT) × ENS.

From reported data in area and latency, our design for the Kyber parameter set has improved from 7% to 82% in ATP compared to previous studies. The architecture in [18] is one of the configurable designs supporting the LBC

algorithms under consideration. This design uses multiple BU architectures to support multiple parameter sets. However, this approach leads to the high consumption of hardware resources, making the use of resources less efficient. This result shows that our architecture is 72% to 82% more efficient for Kyber in terms of ATP. Our design can accelerate many algorithms while using fewer resources. All system resources are reused efficiently, including DSPs, BRAMs, and functional modules such as adders, subtractors, modular reductions, and memories.

The designs in [18], [21], and [24] also support polynomial multiplication. However, resource optimization needs to improve, leading to low hardware efficiency. In [21], the design supports Kyber and Dilithium. The accelerator uses one BU, which leads to slower execution, but many BRAMs and DSPs are used for the architecture. The report shows that our design got ATP improved by 76% in Kyber compared to this design. The study [24] did not optimize memory usage, so it lost 7% to 31% in terms of ATP compared to us. Ye et al. in [19] proposed a pipeline architecture consisting of multiple consecutive BUs to execute NTT for Kyber and other algorithms. The drawback of this approach is the delay caused by the reordering unit, leading to the time efficiency not being able to compensate for the hardware loss. The results show that our design improves ATP by 30% compared to this design.

Although our accelerator supports many different parameter sets, our ATP results are still better than several algorithm-specific designs. Compared with the designs in [22] and [25], which are algorithm-specific designs, the reported results show that these architectures are slower than our architecture and consume more resources, where [25] uses a radix-4 BU. The results show that our architecture improved by 46% compared to the study. The architecture in [22] does not use BRAM but instead uses registers to store data. Therefore, our design improves ATP by 68% compared to this architecture.

For the Dilithium algorithm, our accelerator is also more efficient, 5% to 63%, than the designs for this algorithm. The design in [23] is a study that supports multiple sets of parameters. This design uses multiple BUs and RAMs to store coefficients and twiddle factors, making the memory access pattern more complicated to avoid memory conflicts. Furthermore, memory access requires waiting to read/write to RAM. Although this design supports the entire algorithm, when compared to the NTT-based executions, our design is entirely superior in terms of resource efficiency. Specifically, our ATP results improved from 8% to 44% for Dilithium. The study [26] supports the Dilithium algorithm in the SW/HW approach. The authors design each HW module individually and assign instructions to them. The resulting hardware implementation is 42% less efficient than our design.

For the FALCON algorithm, our design improves by 79% and 50% over previous studies for FALCON-512 and FALCON-1024, respectively. The designs in [19] and [28] both support FALCON with $n = 512$ and $n = 1024$. However, we can see that these designs are less efficient 44% and 47% compared to ours. The architecture in [18] only supports FALCON-512 but, with low hardware utilization, results in 79% ATP inferior to our design. The accelerator in [29] gives comparable results to ours in ATP, but it did not support INTT and multiplication. The design in [23] supports FALCON-1024 and has low latency for NTT but utilized more resources, thus resulting in 50% ATP inferior to ours. Table 7 compares the number of CCs for performing NTT/INTT/polyMul on the proposed SoC with other RISC-V platforms. The CCs for polyMul execution are counted since the data is sent to the accelerator until the result is returned to the CPU. For the cycle data reported separately, the CCs for polyMul are calculated as the sum of the CCs of the processes.

For CRYSTALS algorithms (Kyber and Dilithium), compared with the SW executions on ARM-Cortex [40], [41], [42], our proposed solution improves up to 1.93×/2.06×/5.04× in latency of NTT/INTT/polyMul operations for Kyber and improve 1.81×/1.88×/2.82× in latency of NTT/INTT/polyMul operations for Dilithium. It should be noted that this software is optimized to achieve the most significant speed. In the study [56], the authors implemented NTT with different parameter sets using multiple methods, including SW, High-Level Synthesis (HLS), and HW. The authors also investigated various platforms, including FPGA, CPU, MCU, and ASIC. Our results are compared with RISC-V implementations using the instruction set, and the

TABLE 7. Comparison of clock cycles for NTT process by SW and SW/HW method on RISC-V Platform.

Works	Platforms	Methods	Clock Cycles		
			NTT	INTT	polyMul*
Kyber, $n = 256, q = 3329$					
[40]	ARM-Cortex M4	SW	6,847 (1.65×)	6,975 (1.67×)	-
[41]	ARM-Cortex M3	SW	8,026 (1.93×)	8,594 (2.06×)	22,471 (5.04×)
	Freedom E310	SW	15,888 (3.82×)	15,719 (3.77×)	40,920 (9.18×)
	PQRISCV	SW	21,975 (5.29×)	23,666 (5.67×)	58,709 (13.17×)
[42]	ARM-Cortex M4	SW	5,992 (1.44×)	6,282 (1.51×)	13,887 (3.12×)
[30]	Pico RV32 rv32im	SW/HW	43,756 (10.53×)	-	-
[31]	VexRISCV rv32im	SW/HW	6,868 (1.65×)	6,367 (1.53×)	15,630 (3.51×)
[15]	CVA6	SW/HW	18,488 (4.45×)	18,488 (4.43×)	-
[32]	RISCV core	SW/HW	7,188 (1.73×)	-	-
[33]	Hbird E203 core	SW/HW	4,189 (1×)	3,481 (0.83×)	10,927 (2.45×)
TW	**Rocket rv32imac**	**SW/HW**	**4,156**	**4,172**	**4,457**
Dilithium, $n = 256, q = 8380417$					
[42]	ARM-Cortex M4	SW	8,093 (1.81×)	8,415 (1.88×)	18,463 (2.82×)
[30]	Pico RV32 rv32im	SW/HW	43,756 (9.81×)	-	-
[15]	CVA6	SW/HW	18,554 (4.16×)	21,375 (4.78×)	-
[56]	Pico RV32 rv32im	SW	122,600 (27.49×)	-	-
	BOOM r64gc	SW	17,419 (3.91×)	-	-
TW	**Rocket rv32imac**	**SW/HW**	**4,459**	**4,475**	**5,647**
FALCON-512, $n = 512, q = 12289$					
[56]	Pico RV32 rv32im	SW	481,096 (56.79×)	-	-
	BOOM r64gc	SW	68,456 (8.08×)	-	-
[30]	Pico RV32 rv32im	SW/HW	81,114 (9.57×)	-	-
TW	**Rocket rv32imac**	**SW/HW**	**8,472**	**8,485**	**9,788**
FALCON-1024, $n = 1024, q = 12289$					
[56]	Pico RV32 rv32im	SW	1,063,310 (58.91×)	-	-
	BOOM r64gc	SW	97,563 (5.41×)	-	-
[30]	Pico RV32 rv32im	SW/HW	180,327 (9.99×)	-	-
TW	**Rocket rv32imac**	**SW/HW**	**18,050**	**18,086**	**20,866**

* The polyMul includes one NTT, one PWM, and one INTT

results show that our architecture improves up to 27.49× in speed for Dilithium.

For the SW/HW method, the proposed SoC also significantly improves the speed for NTT/INTT/polyMul. Specifically, the accelerator achieves 1.73× to 10.53× speed up for NTT/INTT with Kyber [32], [30]. Our design also gets up to 9.81× faster in NTT for Dilithium. The study [32] shows the speed-up by not using the ISA extension. With our implementation, we can speed up 1.73× the NTT execution by using accelerators as a peripheral, executing in parallel with other tasks on the CPU. The implementation of [33] shows that NTT/INTT is comparable to ours, but our polyMul computation achieves a speed-up. This is achieved by directly fetching data from dedicated memory and applying the accelerator's pipeline architecture. As a result, the polyMul in Kyber is improved by 2.45×.

Evaluation for FALCON, our method also gives better results, even better than 64-bit SoCs in [56], with speed-up NTT up to 8.08× and 5.41× for FALCON-512 and FALCON-1024, respectively. Most HW/SW solutions are implemented by improving the architecture of an operation, such as modular addition/subtraction [31], modular multiplication, or vector-matrix multiplication. When applied to FALCON, our architecture can speed up 9.57× over FALCON-512 and 9.99× over FALCON-1024. The results show that our proposed SW/HW method allows for a significant improvement in implementing NTT-based operations, contributing to the general progress of implementing LBC algorithms. Also, only a few designs support NTT/INTT and vector-matrix polynomial multiplication for PQC standards.

VI. CONCLUSION

This paper proposes a high-efficiency, run-time configurable polynomial multiplication accelerator supporting currently standardized PQC algorithms, including FIPS 203 (Kyber), FIPS 204 (Dilithium), and underdevelopment standard FIPS 206 (FALCON). This accelerator performs NTT-based operations with a simple control procedure while avoiding RAW memory conflict. The FPGA implementation results show that the proposed design has achieved hardware efficiency regarding ATP. Compared to state-of-the-art studies, the improvement of design is up to 82% for Kyber, up to 63% for Dilithium, and up to 79% and 50% for FALCON-512 and FALCON-1024, respectively. The improvement is achieved by reusing resources efficiently and by applying a configurable reordering unit with a feedback register to eliminate the initial time between the processes. We also propose a RISC-V SoC architecture using the accelerator as a peripheral to the system, with software-based NTT/NTT/polyMul evaluation.

We have proposed two data flow scenarios for software implementation. The SoC can support NTT/INTT execution of up to 4 polynomials simultaneously and allow multiplication between 2 vectors of up to 4 polynomials. Software evaluation results show that this design has significantly accelerated compared to the results of SW implementations on ARM and other RISC-V platforms. The software evaluation results show that using the accelerator allows for up to 5.29×, 27.49×, 56.79×, and 58.91× for Kyber, Dilithium, FALCON-512 and FALCON-1024, respectively. Our study also outperforms other HW/SW designs with improvements of up to 10.53×, 9.81×, 9.57×, and 9.99× for the desired algorithms. Furthermore, our design can be applied to other LBC PQC algorithms to solve time-consuming calculations using the SW/HW method. In the future, we aim to deploy LBC-based systems using the designed accelerator and improve the efficiency of a system based on the tightly coupled accelerator with the processors.

REFERENCES

[1] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proc. 35th Annu. Symp. Found. Comput. Sci.*, Nov. 1994, pp. 124–134.

[2] G. Alagic, D. Cooper, Q. Dang, T. Dang, J. Kelsey, J. Lichtinger, Y.-K Liu, C. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, D. S. Tone, and D. Apon, "Status report on the third round of the NIST post-quantum cryptography standardization process," U.S. Dept. Commerce, Gaithersburg, MD, USA, Tech. Rep. NIST IR 8413, Jul. 2022.

[3] Nat. Inst. Standards Technol. (Aug. 2024). *Module-Lattice-Based KeyEncapsulation Mechanism Standard*. [Online]. Available: https://doi.org/10.6028/NIST.FIPS.203

[4] Nat. Inst. Standards Technol. (Aug. 2024). *Module-Lattice-Based Digital Signature Standard*. [Online]. Available: https://doi.org/10.6028/NIST.FIPS.204

[5] P. -A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang. (Oct. 2020). *Falcon: Fast-Fourier Lattice-Based Compact Signatures Over NTRU (v1.2)*. NIST PQC Round. [Online]. Available: https://falcon-sign.info/falcon.pdf

[6] A. A. Karatsuba and Y. P. Ofman, "Multiplication of many-digital numbers by automatic computers," *Proc. Doklady Akademii Nauk*, vol. 145, no. 2, pp. 293–294, Feb. 1962.

[7] A. L. Toom, "The complexity of a scheme of functional elements realizing the multiplication of integers," *Soviet Phys. Doklady*, vol. 3, no. 4, pp. 714–716, 1963.

[8] S. A. Cook and S. O. Aanderaa, "On the minimum computation time of functions," *Trans. Amer. Math. Soc.*, vol. 142, pp. 291–314, Aug. 1969.

[9] D.-T. Dam, T.-H. Tran, V.-P. Hoang, C.-K. Pham, and T.-T. Hoang, "A survey of post-quantum cryptography: Start of a new race," *Cryptography*, vol. 7, no. 3, p. 40, Aug. 2023.

[10] C.-M.-M. Chung, V. Hwang, M. J. Kannwischer, G. Seiler, C.-J. Shih, and B.-Y. Yang, "NTT multiplication for NTT-unfriendly rings: New speed records for saber and NTRU on cortex-M4 and AVX2," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2021, pp. 159–188, Feb. 2021.

[11] G. Seiler, "Faster AVX2 optimized NTT multiplication for ring-LWE lattice cryptography," *Cryptol. ePrint Arch.*, vol. 2018, pp. 1–14, Jun. 2018.

[12] Y. Gao, J. Xu, and H. Wang, "CuNH: Efficient GPU implementations of post-quantum KEM NewHope," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 3, pp. 551–568, Mar. 2022.

[13] N. Gupta, A. Jati, A. K. Chauhan, and A. Chattopadhyay, "PQC acceleration using GPUs: FrodoKEM, NewHope, and Kyber," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 3, pp. 575–586, Mar. 2021.

[14] H. Vincent, K. YoungBeom, and S. S. Chung, "Barrett multiplication for Dilithium on embedded devices," *Cryptol. ePrint Arch.*, vol. 2023, pp. 1–19, Apr. 2023. [Online]. Available: https://eprint.iacr.org/2023/1955

[15] P. Nannipieri, S. Di Matteo, L. Zulberti, F. Albicocchi, S. Saponara, and L. Fanucci, "A RISC-V post quantum cryptography instruction set extension for number theoretic transform to speed-up CRYSTALS algorithms," *IEEE Access*, vol. 9, pp. 150798–150808, 2021.

[16] A. Waterman, Y. Lee, D. Patterson, and K. Asanovic. (May 2017). *The RISC-V Instruction Set Manual—Volume I: User-Level ISA*. [Online]. Available: https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf

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

[18] K. Derya, A. C. Mert, E. Öztürk, and E. Savaş, "CoHA-NTT: A configurable hardware accelerator for NTT-based polynomial multiplication," *Microprocessors Microsyst.*, vol. 89, Mar. 2022, Art. no. 104451.
[19] Z. Ye, R. C. C. Cheung, and K. Huang, "PipeNTT: A pipelined number theoretic transform architecture," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 69, no. 10, pp. 4068–4072, Oct. 2022.
[20] J. Sun, X. Bai, and Y. Kang, "An FPGA-based efficient NTT accelerator for post-quantum cryptography CRYSTALS-Kyber," in *Proc. IEEE Int. Conf. Integr. Circuits, Technol. Appl. (ICTA)*, Oct. 2023, pp. 142–143.
[21] S. D. Matteo, I. Sarno, and S. Saponara, "CRYPHTOR: A memory-unified NTT-based hardware accelerator for post-quantum CRYSTALS algorithms," *IEEE Access*, vol. 12, pp. 25501–25511, 2024.
[22] A. Y. Hummdi, A. Aljaedi, Z. Bassfar, S. S. Jamal, M. M. Hazzazi, and M. U. Rehman, "Unif-NTT: A unified hardware design of forward and inverse NTT for PQC algorithms," *IEEE Access*, vol. 12, pp. 94793–94804, 2024.
[23] B. Li, Y. Yan, Y. Wei, and H. Han, "Scalable and parallel optimization of the number theoretic transform based on FPGA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 32, no. 2, pp. 291–304, Feb. 2024.
[24] S. Mandal and D. B. Roy, "KiD: A hardware design framework targeting unified NTT multiplication for CRYSTALS-Kyber and CRYSTALS-Dilithium on FPGA," in *Proc. 37th Int. Conf. VLSI Design 23rd Int. Conf. Embedded Syst. (VLSID)*, Jan. 2024, pp. 455–460.
[25] J. Sun and X. Bai, "A high-speed hardware architecture of NTT accelerator for CRYSTALS-Kyber," *Integr. Circuits Syst.*, vol. 1, no. 2, pp. 1–11, May 2024.
[26] G. Mao, D. Chen, G. Li, W. Dai, A. I. Sanka, Ç. K. Koç, and R. C. C. Cheung, "High-performance and configurable SW/HW co-design of post-quantum signature CRYSTALS-Dilithium," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 16, no. 3, pp. 1–28, Jun. 2023.
[27] W. Guo and S. Li, "Split-radix based compact hardware architecture for CRYSTALS-Kyber," *IEEE Trans. Comput.*, vol. 73, no. 1, pp. 97–108, Jan. 2024.
[28] J. Mu, Y. Ren, W. Wang, Y. Hu, S. Chen, C.-H. Chang, J. Fan, J. Ye, Y. Cao, H. Li, and X. Li, "Scalable and conflict-free NTT hardware accelerator design: Methodology, proof, and implementation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 5, pp. 1504–1517, May 2023.
[29] X. Chen, B. Yang, S. Yin, S. Wei, and L. Liu, "CFNTT: Scalable radix-2/4 NTT multiplication architecture with an efficient conflict-free memory mapping scheme," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2021, pp. 94–126, Nov. 2021.
[30] E. Karabulut and A. Aysu, "RANTT: A RISC-V architecture extension for the number theoretic transform," in *Proc. 30th Int. Conf. Field-Programmable Log. Appl. (FPL)*, Aug. 2020, pp. 26–32.
[31] E. Alkim, H. Evkan, N. Lahr, R. Niederhagen, and R. Petri, "ISA extensions for finite field arithmetic: Accelerating Kyber and NewHope on RISC-V," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2020, pp. 219–242, Jun. 2020.
[32] M. Wygrzvwalski, P. Skrzypiec, and R. Szczygiel, "Hardware acceleration method using RISC-V core with no ISA extensions," in *Proc. 31st Int. Conf. Mixed Design Integr. Circuits Syst. (MIXDES)*, Jun. 2024, pp. 265–269.
[33] L. Li, G. Qin, Y. Yu, and W. Wang, "Compact instruction set extensions for Kyber," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 43, no. 3, pp. 756–760, Mar. 2024.
[34] Y. Zhao, R. Xie, G. Xin, and J. Han, "A high-performance domain-specific processor with matrix extension of RISC-V for module-LWE applications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 7, pp. 2871–2884, Jul. 2022.
[35] S.-H. Liu, C.-Y. Kuo, Y.-N. Mo, and T. Su, "An area-efficient, conflict-free, and configurable architecture for accelerating NTT/INTT," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 32, no. 3, pp. 519–529, Mar. 2024.
[36] F. Hirner, A. C. Mert, and S. S. Roy, "Proteus: A pipelined NTT architecture generator," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 32, no. 7, pp. 1228–1238, Jul. 2024.
[37] Q. D. Truong, P. N. Duong, and H. Lee, "Efficient low-latency hardware architecture for module-lattice-based digital signature standard," *IEEE Access*, vol. 12, pp. 32395–32407, 2024.
[38] T.-H. Nguyen, B. Kieu-Do-Nguyen, C.-K. Pham, and T.-T. Hoang, "High-speed NTT accelerator for CRYSTAL-Kyber and CRYSTAL-Dilithium," *IEEE Access*, vol. 12, pp. 34918–34930, 2024.
[39] Y. Zhao, X. Liu, Y. Hu, and H. Xiao, "Design of an efficient NTT/INTT architecture with low-complex memory mapping scheme," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 71, no. 1, pp. 400–404, Jan. 2024.
[40] E. Alkim, Y. A. Bilgin, M. Cenk, and F. Gérard, "Cortex-M4 optimizations for {R, M} LWE schemes," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2020, pp. 336–357, Jun. 2020.
[41] J. Huang, H. Zhao, J. Zhang, W. Dai, L. Zhou, R. C. C. Cheung, Ç. K. Koç, and D. Chen, "Yet another improvement of Plantard arithmetic for faster Kyber on low-end 32-bit IoT devices," *IEEE Trans. Inf. Forensics Security*, vol. 19, pp. 3800–3813, 2024.
[42] A. Abdulrahman, V. Hwang, M. J. Kannwischer, and A. Sprenkels, "Faster Kyber and Dilithium on the cortex-M4," in *Proc. Int. Conf. Appl. Cryptography Netw. Secur.* Cham, Switzerland: Springer, Jan. 2022, pp. 853–871.
[43] V. Lyubashevsky, D. Micciancio, C. Peikert, and A. Rosen, "SWIFFT: A modest proposal for FFT hashing," in *Proc. 15th Int. Workshop Fast Softw. Encryption*. Cham, Switzerland: Springer, Jul. 2008, pp. 54–72.
[44] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, no. 90, pp. 297–301, Apr. 1965.
[45] W. M. Gentleman and G. Sande, "Fast Fourier transforms: For fun and profit," in *Proc. Fall Joint Comput. Conf. (AFIPS)*, Nov. 1966, pp. 563–578.
[46] S. S. Roy, F. Vercauteren, N. Mentens, D. D. Chen, and I. Verbauwhede, "Compact ring-LWE cryptoprocessor," in *Proc. 16th Int. Workshop Cryptograph. Hardw. Embedded Syst.–CHES*. Cham, Switzerland: Springer, Jan. 2014, pp. 371–391.
[47] T. Pöppelmann, T. Oder, and T. Güneysu, "High-performance ideal lattice-based cryptography on 8-bit ATxmega microcontrollers," in *Proc. Int. Conf. Cryptol. Info. Secur. Latin Amer.* Cham, Switzerland: Springer, Jan. 2015, pp. 346–365.
[48] P. Barrett, "Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor," in *Proc. Annu. Int. Crypto. Conf. (CRYPTO)*, Apr. 2007, pp. 311–323.
[49] P. L. Montgomery, "Modular multiplication without trial division," *Math. Comput.*, vol. 44, no. 170, pp. 519–521, Apr. 1985.
[50] P. Longa and M. Naehrig, "Speeding up the number theoretic transform for faster ideal lattice-based cryptography," in *Proc. Int. Conf. Cryptol. Netw. Secu. (CANS)*, Jan. 2016, pp. 124–139.
[51] ChipYard. (2024). *Rocketchip—Version: Stable*. [Online]. Available: https://chipyard.readthedocs.io/en/stable/Generators/Rocket-Chip.html
[52] Sifive, Inc. (Jan. 2020). *SiFive TileLink Specification—Version 1.8.1*. [Online]. Available: https://sifive.cdn.prismic.io/sifive/7bef6f5c-ed3a-4712-866a-1a2e0c6b7b13_tilelink_spec_1.8.1.pdf
[53] Y. Xing and S. Li, "A compact hardware implementation of CCA-secure key exchange mechanism CRYSTALS-KYBER on FPGA," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2021, pp. 328–356, Feb. 2021.
[54] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS-Kyber: Algorithm specifications and supporting documentation (version 3.02)," NIST PQC Round, pp. 1–43, Aug. 2021. [Online]. Available: https://www.pq-crystals.org/kyber/data/kyber-specification-round3-20210804.pdf#page=36.69
[55] S. Bai, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS-Dilithium: Algorithm specifications and supporting documentation (version 3.1)," NIST PQC Round, pp. 1–38, Feb. 2021. [Online]. Available: https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf
[56] A. C. Mert, E. Karabulut, E. Öztürk, E. Savas, and A. Aysu, "An extensive study of flexible design methods for the number theoretic transform," *IEEE Trans. Comput.*, vol. 71, no. 11, pp. 2829–2843, Nov. 2022.

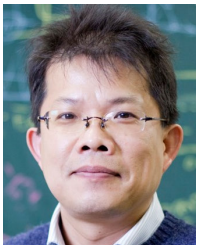
DUC-THUAN DAM (Graduate Student Member, IEEE) received the B.Sc. degree in electrical and electronic engineering and the M.S. degree in electronic engineering from Le Quy Don Technical University, Hanoi, Vietnam, in 2013 and 2019, respectively. He is currently pursuing the Ph.D. degree in information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

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

THAI-HA TRAN (Graduate Student Member, IEEE) received the B.Sc. degree in electrical and electronic engineering and the M.S. degree in electronic engineering from Le Quy Don Technical University, Hanoi, Vietnam, in 2012 and 2018, respectively. He is currently pursuing the Ph.D. degree in information and network engineering with The University of Electro-Communications (UEC), Tokyo, Japan. His research interests include hardware security, digital circuits and systems, and digital signal processing.

DUC-HUNG LE (Member, IEEE) received the B.Sc. degree in physics and the M.Sc. degree in electronic physics from the University of Science, Vietnam National University Ho Chi Minh City, in 2001 and 2005, respectively, and the Ph.D. degree in advanced science and engineering from The University of Electro-Communications (UEC), Tokyo, Japan, in 2013. He is currently the Head of the Electronics Department and the DESLAB, Faculty of Electronics and Telecommunications, University of Science, Vietnam National University Ho Chi Minh City. His research interests include IC design, SoC design, digital signal processing, neuromorphic computing, and biomedical electronics.

TRONG-THUC HOANG (Member, IEEE) received the B.Sc. and M.S. degrees in electronic engineering from Ho Chi Minh City University of Science (HCMUS), Ho Chi Minh City, Vietnam, in 2012 and 2017, respectively, and the Ph.D. degree in engineering from The University of Electro-Communications (UEC), Tokyo, Japan, in 2022. Since April 2022, he has been an Assistant Professor with the Department of Computer and Network Engineering, UEC. His research interests include digital signal processing, computer architecture, cyber-security, and ultra-low power system-on-a-chip.



CONG-KHA PHAM (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electronics engineering from Sophia University, Tokyo, Japan, in 1988, 1990, and 1992, respectively. He is currently a Professor with the Department of Information and Network Engineering, The University of Electro-Communications (UEC), Tokyo, Japan. His research interests include hardware system design and implementation by FPGAs and integrated circuits. The University of Electro-Communications Integrated Circuit Design Laboratory (Pham Lab) educates on the design, implementation, and evaluation of hardware systems and VLSI, aims to design ''system-on-chip'' by integrating various information processing hardware, and develops a high-performance computational circuit realized with a small number of elements.