PQNTRU: Acceleration of NTRU-based Schemes via Customized Post-Quantum Processor

Zewen Ye, Junhao Huang, Tianshun Huang, Yudan Bai, Jinze Li, Hao Zhang, Guangyan Li, Donglong Chen, *Member, IEEE*, Ray C.C. Cheung, *Senior Member, IEEE*, Kejie Huang, *Senior Member, IEEE*

Abstract—Post-quantum cryptography (PQC) has rapidly evolved in response to the emergence of quantum computers, with the US National Institute of Standards and Technology (NIST) selecting four finalist algorithms for PQC standardization in 2022, including the Falcon digital signature scheme. The latest round of digital signature schemes introduced Hawk, both based on the NTRU lattice, offering compact signatures, fast generation, and verification suitable for deployment on resource-constrained Internet-of-Things (IoT) devices. Despite the popularity of Crystal-Dilithium and Crystal-Kyber, research on NTRU-based schemes has been limited due to their complex algorithms and operations. Falcon and Hawk's performance remains constrained by the lack of parallel execution in crucial operations like the Number Theoretic Transform (NTT) and Fast Fourier Transform (FFT), with data dependency being a significant bottleneck. This paper enhances NTRU-based schemes Falcon and Hawk through hardware/software co-design on a customized Single-Instruction-Multiple-Data (SIMD) processor, proposing new SIMD hardware units and instructions to expedite these schemes along with software optimizations to boost performance. Our NTT optimization includes a novel layer merging technique for SIMD architecture to reduce memory accesses, and the use of modular algorithms (Signed Montgomery and Improved Plantard) targets various modulus data widths to enhance performance. We explore applying layer merging to accelerate fixed-point FFT at the SIMD instruction level and devise a dual-issue parser to streamline assembly code organization to maximize dual-issue utilization. A System-on-chip (SoC) architecture is devised to improve the practical application of the processor in real-world scenarios. Evaluation on 28 nm technology and FPGA platform shows that our design and optimizations can increase the performance of Hawk signature generation and verification by over 7×.

Index Terms—Post-quantum Cryptograph, NTRU, Falcon, Hawk, RISC-V, System on Chip

Zewen Ye is with the College of Information Science & Electronic Engineering, Zhejiang University and the Department of Electrical Engineering, City University of Hong Kong. E-mail: lucas.zw.ye@zju.edu.cn.
Junhao Huang and Donglong Chen are with BNU-HKBU United International College, Zhuhai, China. E-mail: {huangjunhao, donglongchen}@uic.edu.cn.
Guangyan Li and Ray C.C. Cheung are with the Department of Electrical Engineering, City University of Hong Kong, Hong Kong, China. E-mail: guangyali5-c@my.cityu.edu.hk, r.cheung@cityu.edu.hk.
Tianshun Huang, Yudan Bai, Jinze Li, Hao Zhang, and Kejie Huang are with the College of Information Science & Electronic Engineering, Zhejiang University, Hangzhou, China. E-mail: z1458152445@163.com, {byd.baiyudan, lijinze2233, floyd.haozhang, huangkejie}@zju.edu.cn.

1 Introduction

Quantum computing has been rapidly developing. One of the most significant implications of quantum computing is its potential to disrupt current cryptographic systems, particularly public key cryptography (PKC). In 1994, Peter W. Shor [1] proposed the powerful polynomial-time algorithms for prime factorization and discrete logarithm problems on quantum computers, which are considered hard on classic computers and widely used in the existing PKC schemes like RSA, ElGamal, and Elliptic Curve Cryptography (ECC). In 2016, the National Institute of Standards and Technology (NIST) of the United States initiated a worldwide PQC standardization competition to call for quantum-safe key encapsulation and digital signature algorithms. As of the latest round of evaluations in 2022, NIST has selected four finalist algorithms to be standardized. For the key encapsulation mechanism (KEM), the Crystal-Kyber [2] is the only KEM finalist that will be standardized. And Crystal-Dilithium [3], Falcon [4], and SPHINCS+ [5] are the three digital signature finalists. Apart from the four announced finalists, NIST also initiated an additional call for digital signature schemes to diversify the post-quantum signature standards [6]. Therefore, the efficient implementation of NIST PQC finalists and additional digital signature candidates in various scenarios will be of great interest in the next few years, which will effectively help NIST select efficient and secure digital signature candidates.

As Internet-of-Thing (IoT) [7] devices have revolutionized various industries like smart homes, smart cities, smart healthcare, etc., the transition from the traditional PKC schemes to PQC schemes poses serious challenges to these IoT devices. This is because the existing PQC schemes normally have a larger public key, secret key or signature than the traditional PKC schemes such as ECC Curve25519 [8] and Ed25519 [9]. For example, the signature size of Ed25519 is merely 64 bytes [9] while Dilithium2's signature size is up to 2,420 bytes [3], which is 37× larger. For IoT devices that have low memory, limited power consumption and small computing capability, the PQC deployment on IoT devices requires significant effort to achieve low power consumption, high efficiency, small chip area and flexibility.

To meet these requirements, in this paper, we focus on two NTRU-based digital signature schemes: the NIST PQC finalist Falcon [4] and the additional digital signature candidate Hawk [10]. The reason we choose to focus on these two signature schemes is that they both have relatively compact signatures (e.g., 617 and 555 bytes for Falcon-512 and Hawk-512, respectively), fast signature generation and verification. These advantages make it more suitable to be deployed on IoT devices compared to other lattice-based cryptography like Kyber and Dilithium. Moreover, unlike Falcon, Hawk managed to eliminate all the floating-point operations, which makes Hawk well-suited for IoT devices and a competitive candidate in NIST's additional digital signature competition.

There are already many software optimizations for Falcon [11]–[14], which target IoT platforms ARM Cortex M4, M7 and etc. However, due to the limitation of floating-point arithmetic, many hardware implementations work only target Falcon signature verification. Some works only target the Number Theoretic Transform (NTT) acceleration [15]–[19]. The works focused on hardware/software co-design are also very limited [20], [21]. [20] proposed to use high-level synthesis only for the acceleration of NTT, which is very limited to support the whole signature verification. [21] presented a customized RISC-V processor to accelerate the Falcon signature verification. Though it has good performance, the area and power consumption are large. On the other hand, to the best of our knowledge, Hawk has no hardware and hardware/software co-design implementations. Existing works [10], [12] only focused on the software acceleration on Intel AVX2 and ARM Cortex-M4.

In this paper, we proposed PQNTRU, an efficient System on Chip (SoC) design with a customized RISC-V Single-Instruction-Multiple-Data (SIMD) processor for two NTRU-based schemes Falcon and Hawk. The proposed SoC is built upon a customized SIMD architecture processor, AXI system bus, and peripheral and data memory, which is practical for real-world applications. To enhance the performance of NTRU-based schemes Falcon and Hawk, we propose several optimizations at both the algorithmic level and hardware architecture level. To the best of our knowledge, our work is the first in the hardware/software co-design in accelerating the Hawk scheme. The contributions of this paper can be summarized as follows:

- First, we propose a novel layer merging technique in our SIMD architecture for the NTT to reduce the memory accesses. Based on the limited SIMD register files, three layers of NTT can be merged for performance enhancement. With this layer merging technique, NTT is implemented as a loop-based assembly code to reduce the instruction memory space of NTT.
- Second, two different modular algorithms are utilized and optimized in our optimization for different modular data widths. We design novel SIMD instructions to support the acceleration of the Improved Plantard algorithm. With this, the modular reduction of the key pair generation of Hawk is performed with the Signed Montgomery algorithm since it uses large data width moduli, and the other small modular operations are accelerated with the Improved Plantard algorithm.
- Finally, the fixed-point 32-bit and 64-bit Fast Fourier Transform (FFT) are accelerated via instruction-level optimization. In addition, the idea of layer merging is also applied to the acceleration of fixed-point FFT.

To further enhance performance, a dual-issue parser is proposed to automatically organize assembly code with full utilization of dual-issue. Our proposed SoC is evaluated under $28\ nm$ technology and Zynq-7000 FPGA platform. Experimental results show that by using the proposed design, the performance of Hawk signature generation and verification increases by more than $7\times$.

1.1 Paper Organization

The following paper is organized as follows. Firstly, we provide a brief introduction of the related PQC schemes and core operations in Section 2. Then, Section 3 details the hardware architecture designed in this paper. The software implementations of Falcon and Hawk are presented in Section 4. We present the evaluation results and compare our work in Section 5. Finally, we conclude this paper in Section 6.

2 Preliminaries

In this section, we briefly review NTRU, Falcon, Hawk, and the core operations involved in this paper.

2.1 NTRU

Since NTRU was first proposed in 1998 by Jeffrey Hoffstein, Jill Pipher and Joseph H. Silverman [22], many NTRU variants have been proposed based on the original NTRU design, e.g. NTRUEncrypt [23], NTRUSign [24], NTRU prime [25], Falcon [4], Hawk [10], and etc.

The original NTRU lattices was introduced in the NTRUEncrypt [22] and NTRUSign [24]. Specifically, the NTRUEncrypt [22] consists of the triple integer parameters (n, p, q), where p, q are co-prime number $gcd(p, q) = 1$ with q being considerably larger than p, and n is the degree parameter. Given a monic polynomial $\phi \in \mathbb{Z}[x]$ of degree n, the polynomial operations are conducted over the polynomial ring $R = \mathbb{Z}[x]/(\phi)$, $R_q = (\mathbb{Z}/q\mathbb{Z})[x]/(\phi)$ or $R_p = (\mathbb{Z}/p\mathbb{Z})[x]/(\phi)$. The key generation of NTRUEncrypt consists of finding two polynomials f and g, in which the polynomial f has inverses modulo both p and q in R_q and R_p, respectively, and the polynomial g also needs to be invertible in R_q. The private key of this scheme is $(f, f^{-1} \bmod p)$, where $f^{-1} \bmod p$ is also kept for later computation. The public key of NTRUEncrypt is computed as follows:

$$h = gf^{-1} \mod q. \tag{1}$$

For more details about the encryption and decryption procedures of NTRUEncrypt, we refer to [22].

The NTRUSign [24], on the other hand, uses a slightly different lattice, which is defined by two integer parameters (n, q). The polynomial ring used in NTRUSign is $R = \mathbb{Z}[x]/(\phi)$. The main difference is that the key generation procedure of NTRUSign requires finding two "small"

polynomials f and g that are invertible modulo q. Then, one needs to solve the following NTRU equation to find matching "small" F and G and generate the appropriate public key and private key (f, g):

$$fG - gF = q. \tag{2}$$

For more details about the signature generation and verification procedures of NTRUSign, we refer to [24].

2.2 NTRU Solver and Babai' reduction

Efficiently solving the NTRU solution has a crucial impact on the key generation performance of NTRU-based digital signature algorithms. Previous solutions in [24], [26] require cubic and quadratic time and space complexities, which makes it impractical in IoT devices. Later in 2019, Thomas Pornin and Thomas Prest [27] proposed a more efficient NTRU solver for the NTRU equation, which has been adopted in Falcon and Hawk. Since their proposed NTRU solver is a recursive algorithm, and the polynomial operations involved are performed over $R = \mathbb{Z}[x]/(\phi)$, the coefficients of the polynomial would become increasingly larger as the recursive depth gets deeper. Therefore, the Residue Number System (RNS) and Chinese Remainder Theorem (CRT) are leveraged to split the polynomials with large coefficients into a series of polynomials with smaller coefficients modulo small prime moduli r_i. They also ensure that the monic polynomial ϕ has n distinct roots modulo each r_i, thus enabling the use of the NTT to speed up the polynomial multiplication. In this case, the polynomial operations are all computed with cheaper pointwise addition or multiplication modulo each modulus r_i. Finally, we need to recover the polynomial from the RNS representation back to the normal representation. One can directly leverage the CRT (see [27, Equation 32]) to recover the original polynomial. Thereafter, Babai's reduction [27] is used to reduce the coefficient size of F and G down to a specific target size. This computation is computed with the FFT in the floating-point representation.

2.3 Falcon

The Falcon signature scheme [4] is a lattice-based signature scheme that has been selected for standardization as part of NIST's PQC standardization project. It is based on the NTRU lattice: for a given degree $n = 2^\ell$, the private key is a pair of polynomials f and g, with small integer coefficients and taken modulo $X^n + 1$, completed with another pair of such polynomials F and G, again with small integer coefficients, such that the NTRU equation (Equation 2) is fulfilled. The q of the NTRU equation in Falcon is a small and fixed integer (in the case of Falcon, $q = 12289$). The key generation of Falcon directly adopts the efficient NTRU solver from [27], and it also requires the floating-point number to perform the FFT computation, which makes it kind of limited on devices without floating-point unit (FPU). The supported degrees of Falcon are $n = 512$ and $n = 1024$ (for $\ell = 9$ and $\ell = 10$, respectively). We refer to [4] for more details about Falcon.

2.4 Hawk

The Hawk signature scheme [10] is based on a different hard problem, called the Lattice Isomorphism Problem (LIP), but uses lattices similar to those used by Falcon. In Hawk, f and g use a different sampling distribution and generally have a smaller norm, and the target integer for the NTRU equation is $q = 1$. Hawk is also equipped with a similar NTRU solver in [27]. However, Babai's reduction in Hawk uses FFT with fixed-point 64-bit numbers instead of floating-point, which makes it more suitable on devices without an FPU. The degree is $n = 256$, 512 or 1024 (the degree 256 variant does not provide enough security).

2.5 Polynomial Multiplication, FFT and NTT

Polynomial multiplication is a time-consuming operation in lattice-based cryptography. The naive polynomial multiplication with the schoolbook method has time complexity $O(n^2)$. When it is operated over the polynomial ring $\phi = x^n + 1$ over $\mathbb{C}$, where each coefficient is a complex number represented with real and imaginary parts, FFT can be used to speed up polynomial multiplication by first transforming the polynomial into Fourier form. This is achieved using a divide-and-conquer methodology to reduce the time complexity to $O(n \cdot \log n)$ [27]. Then, the multiplication can be performed coefficient-wise. In Falcon, these coefficients are represented with 64-bit floating-point numbers, while fixed-point numbers are used in Hawk.

NTT is an FFT variant over the polynomial ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ where all coefficients are integers and operated modulo q. NTT takes a polynomial $a(x) = \sum_{i=0}^{n-1} a_i x^i$ as input and returns the output $\hat{a}(x) = \sum_{i=0}^{n-1} \hat{a}_i x^i$, where

$$\hat{a}_i = \sum_{j=0}^{n-1} a_j \omega^{ij} \pmod q \tag{3}$$

where ω is the primitive n-th root-of-unity modulo q. For INTT, the twiddle factor ω is replaced with the multiplicative inverse ω^{-1}, and additional final scaling n^{-1} is required:

$$a_i = N^{-1} \sum_{j=0}^{n-1} \hat{a}_j \omega^{-ij} \pmod q \tag{4}$$

Similar to FFT, a divide-and-conquer method can be used to reduce the complexity of NTT to $O(n \cdot \log n)$. Polynomial multiplication in R_q can be performed by negacyclic convolution, as shown in Algorithm 1. Note that γ is the primitive $2n$-th root of unity. In this case, the modulus q should satisfy $q \equiv 1 \pmod{2n}$.

Algorithm 1 Polynomial multiplication [28]

Require: $A(x), B(x) \in \mathbb{Z}_q[x]/(x^n + 1)$
Require: Primitive $2n$-th root of unity $\gamma \in Z_q$
Ensure: $C(x) = A(x)B(x), C(x) \in Z_q[x]/(x^n + 1)$
1: // Pre-processing and NTT
2: $\overline{A}(x) \leftarrow \mathbf{NTT}(A(x) \odot (\gamma^0, \gamma^1, ..., \gamma^{n-1}))$
3: $\overline{B}(x) \leftarrow \mathbf{NTT}(B(x) \odot (\gamma^0, \gamma^1, ..., \gamma^{n-1}))$
4: // Point-wise Multiplication
5: $\overline{C}(x) \leftarrow \overline{A}(x) \odot \overline{B}(x)$
6: // INTT and Post-processing
7: $C(x) \leftarrow \mathbf{INTT}(\overline{C}(x)) \odot (\gamma^0, \gamma^{-1}, ..., \gamma^{-(n-1)})$

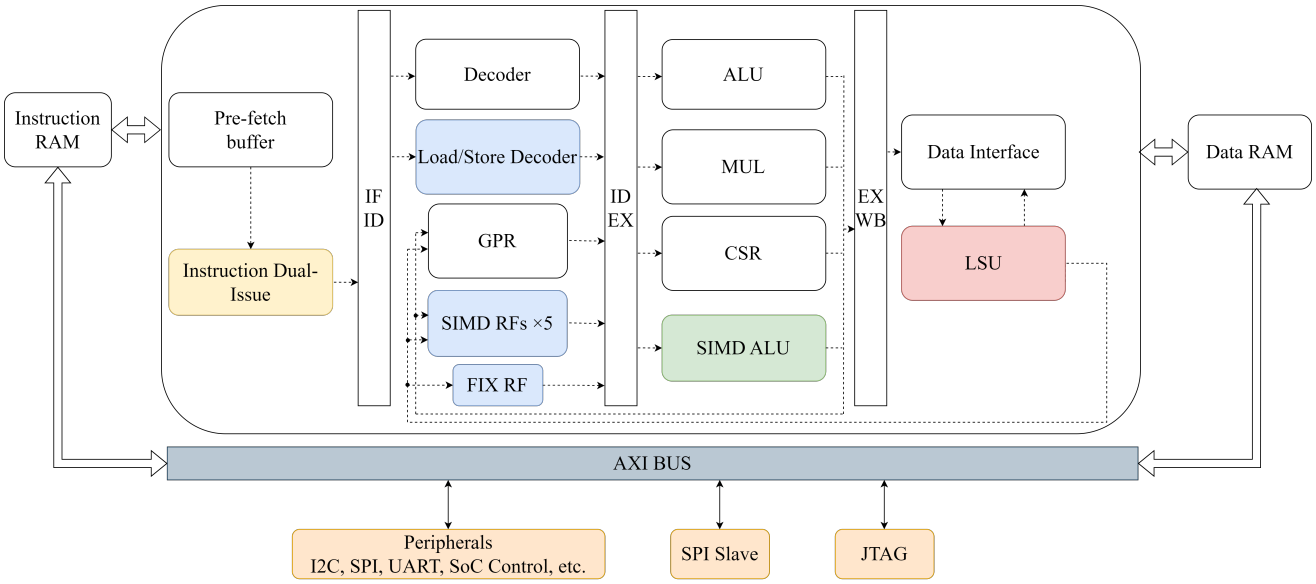

Fig. 1. Proposed SoC Architecture.

3 Hardware Architecture

In this section, we present our proposed System-on-chip (SoC) architecture. The entire system consists of a RISC-V processor core with tailored SIMD instruction sets, an AXI bus, instruction and data Random Access Memory (RAM), and various peripheral interfaces, as depicted in Figure 1. The suggested processor core builds upon previous SIMD processor research [29], which was improved from CV32E40P, a 32-bit, in-order RISC-V core with a 4-stage pipeline and RV32IMC instruction sets. To enhance the efficiency of post-quantum algorithms, specialized SIMD instructions and corresponding hardware designs have been developed. The SIMD supports data widths of eight 32-bit and ten 32-bit (for Keccak) for improved performance.

3.1 SIMD Instruction-Set Architecture

The processor core is composed of four pipeline stages: Instruction Fetch (IF), Instruction Decode (ID), Execution (EX), and Write-Back (WB). The proposed SIMD instruction set architecture enables parallel computation of 256/320 bits and facilitates load/store operations with 64/128 bits.

3.1.1 Instruction Dual-Issue

To improve performance considering the narrower SIMD load/store data width compared to SIMD computing data width, a **dual-issue** path is implemented. This path allows the simultaneous issuance of two 32-bit instructions (one load/store and one non-load/store) for fetching and execution. The two issued instructions can execute in parallel only when there are no data dependencies. It is important to note that this dual-issue design applies to both SIMD instructions and RV32 instructions.

3.1.2 Register File (RF)

In the ID stage, 5 SIMD Register Files (RFs) are introduced, each containing 16 rows of 64 bits. These RFs can collectively store and load a total of 10×32-bit data during computation. Additionally, a small Register File (**FIX RF**) with 2 rows of 32 bits is included to store frequently used constant values and parameters such as q and q^{-1} in Falcon and Hawk.

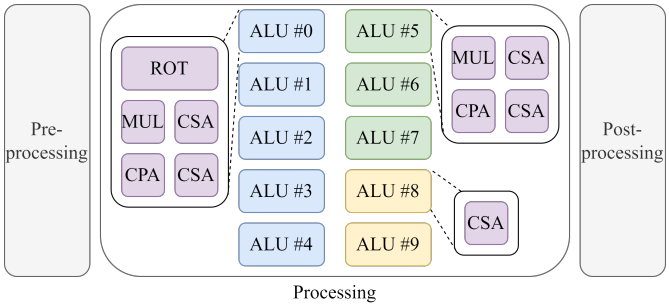

Fig. 2. The Structure of SIMD ALU.

3.1.3 SIMD ALU

An **SIMD ALU** is integrated into the EX stage to handle 8/10 concurrent operations on 32-bit data, as depicted in Figure 2. The ALUs are responsible for computation, with pre-processing and post-processing facilitating internal data shuffling. This includes input shuffling and output shuffling to rearrange data before and after computation and a reverse shuffling feature for reversing the order of eight 32-bit SIMD data to expedite polynomial auto-adjoint operations. The various supported shuffling methods are detailed in Table 1 and are essential for addressing data dependencies in scenarios like NTT and FFT. The SIMD ALU comprises a 10-core design that can accept up to three 320-bit data inputs from PRs and produce one 320-bit output. These cores are categorized into three types based on their hardware resources: Carry Propagate Adders (CPA), Carry Save Adders (CSA), multipliers (MUL), and rotators (ROT). While all 10 cores are active during the execution of customized SIMD Keccak instructions, typically, only 8 cores are utilized for SIMD computations.

TABLE 1
Supported shuffling methods.

	op	input data order	output data order
input shuffling	rs1	$0,1,2,3,4,5,6,7$	$0,8,1,9,2,10,3,11$
	rs2	$8,9,10,11,12,13,14,15$	$4,12,5,13,6,14,7,15$
output shuffling	rd	$0,1,2,3,4,5,6,7$	$0,2,4,6,1,3,5,7$
reverse shuffling	rd	$0,1,2,3,4,5,6,7$	$7,6,5,4,3,2,1,0$

3.2 System on Chip (SoC)

In addition to the processor core, the proposed System on Chip (SoC) incorporates the AXI interface as the primary system bus for data communication. The peripheral component encompasses various protocols such as I2C, SPI, UART, and SoC control, serving as fundamental hardware modules for software library functions like *printf*, interrupt handling, and more. Both a data RAM and an instruction RAM, along with their respective control logic, are linked to both the processor core and the AXI bus. SPI Slave and JTAG functionalities are utilized for memory programming and debugging purposes during runtime. With this configuration, the entire SoC is capable of executing programs compiled for RISC-V architecture and our customized SIMD instructions.

4 Software Optimization

In this section, we will outline our proposed software optimization at both the algorithmic and hardware architecture levels. We incorporate two modular reduction algorithms to enhance the efficiency of various moduli used in Falcon and Hawk schemes. Our optimizations of NTT/INTT aim to minimize memory access. The computation of twiddle factors is determined by our NTT/INTT design. Furthermore, we will introduce optimizations for 32-bit and 64-bit fixed-point FFT/IFFT.

4.1 Modular Reduction Optimization

Algorithm 2 Signed Montgomery Reduction [30]

Require: $0 < q < \frac{\beta}{2}, -\frac{\beta}{2}q \leq a = a_1\beta + a_0 < \frac{\beta}{2}q$ where $\beta \cdot \beta^{-1} \equiv 1 \pmod{q}, 0 \leq a_0 < \beta$
Ensure: $r = \beta^{-1}a \pmod{q}, -q < r < q$
1: $m \leftarrow a_0 q^{-1} \bmod^{\pm} \beta$
2: $t_1 \leftarrow \lfloor mq/\beta \rfloor$
3: $r \leftarrow a_1 - t_1$

We apply two modular reduction algorithms in our optimized Falcon and Hawk schemes. The first algorithm is the Signed Montgomery reduction algorithm [30] used in our design. This algorithm processes a signed number as an input and produces a modular result within the range of $(-q, q)$, as outlined in Algorithm 2. By setting $\beta = 2^{32}$, this algorithm can be utilized for any modulus smaller than 2^{32}. Additionally, Algorithm 3 illustrates another modular reduction algorithm initially introduced in [31] and subsequently enhanced in [32], [33]. This algorithm accommodates $l = 16$ and is suitable for $q < 2^{15-\alpha}$. A notable advantage of this algorithm is that, when multiplied by a constant, it can save a multiplication with q' by pre-multiplying the constant with q', potentially enhancing efficiency.

Algorithm 3 Improved Plantard Modular Reduction [33]

Require: Input signed integer a such that $a \in [q2^l - q2^{l+\alpha}, 2^{2l} - q2^{l+\alpha}), q < 2^{l-\alpha-1}, q' = q^{-1} \bmod^{\pm} 2^{2l}$
Ensure: $r = a(-2^{-2l}) \bmod^{\pm} q$ where $r \in [-\frac{q+1}{2}, \frac{q}{2})$
1: $t_1 \leftarrow aq' \bmod 2^{2l}$
2: $t_2 \leftarrow \lfloor t_1/2^l \rfloor + 2^{\alpha}$
3: $r \leftarrow \lfloor t_2 \cdot q \cdot 2^l/2^{2l} \rfloor$

Table 2 illustrates the specific algorithms integrated into our enhanced designs. The Signed Montgomery algorithm is employed in the **Hawk.sign** function when dealing with moduli exceeding 2^{16}. In the **Falcon.verify** process, we utilize the modular value of $q = 12289$ along with the Improved Plantard algorithm. While there are multiple modular options for both **Hawk.sign** and **Hawk.verify**, the selection of $q = 12289$ in the Hawk scheme is aimed at optimizing performance.

TABLE 2
Modular algorithms implemented in our design.

	Modular	Applied algorithm
Falcon.verify	12289	Improved Plantard algorithm
Hawk.keypair	$> 2^{16}$	Signed Montgomery algorithm
Hawk.sign	12289	Improved Plantard algorithm
Hawk.verify	12289	Improved Plantard algorithm

4.1.1 Implementation of Signed Montgomery Algorithm

The SIMD implementation of the Signed Montgomery algorithm is demonstrated in Algorithm 4. For more details about the SIMD instructions, readers can refer to our previous work [29]. This algorithm computes the modular multiplication result of x and ω. Line 1 calculates the lower 32-bit multiplication product a_0, while line 2 computes the higher 32-bit multiplication product a_1. In line 3, **mulv** is used to determine $a_0 \cdot q \bmod \beta$, where $\beta = 2^{32}$. The higher 32-bit multiplication outcome of m and q is obtained from line 4. Finally, a subtraction operation is performed to derive the modular result.

Algorithm 4 SIMD implementation of Signed Montgomery multiplication

Require: One signed integer x, the 32-bit twiddle factor ω, $q > 2^{16}$, $\beta = 2^{32}$
Ensure: $r = x \cdot \omega \bmod q$
1: **mulv** a_0, x, ω
2: **mulvh** a_1, x, ω
3: **mulv** m, a_0, q^{-1}
4: **mulvh** t, m, q
5: **subv** r, a_1, t
6: **return** r

4.1.2 Implementation of Improved Plantard Algorithm

Algorithm 5 illustrates the instructions utilized in the implementation of the Improved Plantard modular multiplication, requiring only 3 instructions. To enhance efficiency, pre-computation of ω' is done to eliminate one additional

instruction needed for the multiplication of ω and q'. Additionally, a new instruction **asravi** is introduced in line 2 of Algorithm 5 to expedite the Improved Plantard algorithm by enabling arithmetic right-shift and addition simultaneously. This instruction allows for a right-shift of 16 bits and the addition operation with 2^{α} to be performed in a single step. Notably, the Improved Plantard algorithm necessitates two instructions less than the Signed Montgomery algorithm, making it the preferred choice for $q < 2^{16}$.

Algorithm 5 SIMD implementation of Improved Plantard modular multiplication

Require: One signed integer x, the twiddle factor $\omega, \omega' = \omega \cdot q'$, $q = 12289$, $\alpha = 1$, $l = 16$
Ensure: $r = x \cdot \omega \bmod q$
1: **mulv** t_1, x, ω'
2: **asravi** t_2, t_1, α, l
3: **mulvh** $r, t_2, q \cdot 2^l$
4: **return** r

4.2 Optimization for NTT/INTT

As depicted in Table 2, two kinds of moduli are utilized in Falcon and Hawk schemes. For modulus 12289, the polynomial lengths include 512 and 1024 in Falcon and Hawk schemes, while for moduli greater than 2^{16}, the polynomial lengths range from 2 to 1024.

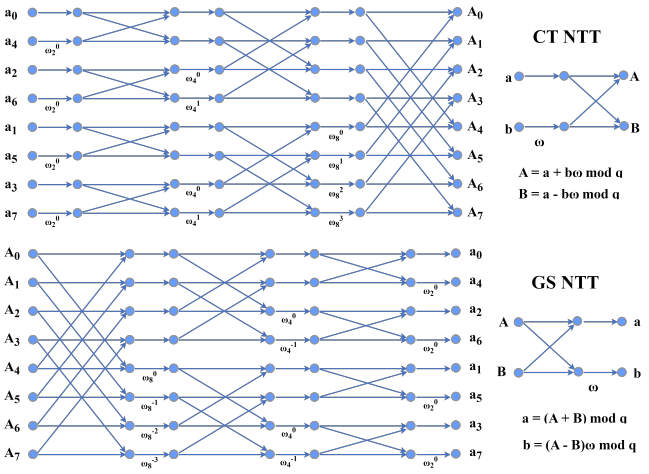

Fig. 3. 8-point polynomial Cooley-Tukey and Gentleman-Sande NTT computing flow.

4.2.1 Butterfly Operation

The butterfly operation is a fundamental component of NTT/INTT, involving one multiplication, one addition, and one subtraction with two inputs and two outputs. In the context of NTT and butterfly operation, there are two main approaches: Cooley-Tukey (CT) and Gentleman-Sande (GS) methods [34], as depicted in Figure 3. The key distinction lies in the sequence of addition/subtraction and multiplication operations. In our configuration, we employ the GS method for NTT and the CT method for INTT. For efficient execution of the butterfly operation, our hardware supports modular addition/subtraction in a single instruction. Modular multiplications are performed using 3 to 5 instructions, as detailed in Algorithm 4 and Algorithm 5.

4.2.2 Layer Merging

The prior research [29] focused exclusively on NTT with a 256 polynomial length, necessitating the loading and storing of coefficients from memory in each layer. This approach led to increased power consumption and code size, where the same data are loaded and stored multiple times. Scaling this method to larger polynomial lengths resulted in significant rises in code size and power usage. To address various polynomial lengths and enhance execution efficiency, we introduce a layer merging technique that minimizes memory accesses and code sizes using our proposed SIMD processor. Our SIMD processor allows parallel processing of eight 32-bit data, facilitating NTT/INTT acceleration for polynomial lengths exceeding 8. Notably, each butterfly operation entails approximately 6 SIMD registers, and our processor boasts 16 SIMD registers. To optimize memory access, we employ 8 SIMD registers to store polynomial coefficients, effectively utilizing most of the SIMD register resources. The remaining registers are allocated for caching computational data and loading processed data for subsequent computations. We meticulously examine the data access patterns of NTT/INTT. For clarity, we delve into layer merging in both CT NTT and GS NTT.

4.2.2.1 GS NTT: In the GS NTT Layer 1, as depicted in Figure 3, A_0 undergoes a butterfly operation with A_4 with a gap of 4. This gap decreases with each subsequent layer. To prevent memory access issues within each layer, the loaded coefficients must be capable of spanning multiple layers.

With eight coefficients, the highest number of layers that can be combined is $\log_2 8 = 3$. For a polynomial of length N, the data chosen for the initial round are $0, N/8, 2N/8, 3N/8, 4N/8, 5N/8, 6N/8, 7N/8$, which represent the starting addresses of eight 32-bit data points. These data points allow for the merging and execution of three layers, as demonstrated in Table 3. Once these layers are processed, the current round of data is stored in memory, and the subsequent round of data, with addresses 8 units apart, is loaded and computed. As the three merged layers are completed, the following layers are also merged and handled using a similar data organization scheme. In general, the addresses of the subsequent merged layers can be treated akin to the first merged layer with a polynomial length of $N/(2^3)$, meaning the data points are $0/(2^3), N/8/(2^3), 2N/8/(2^3), ...$

TABLE 3
Layer merging in GS NTT.

Layer 1				
coefficient a	0	$N/8$	$2N/8$	$3N/8$
coefficient b	$4N/8$	$5N/8$	$6N/8$	$7N/8$
Layer 2				
coefficient a	0	$N/8$	$4N/8$	$5N/8$
coefficient b	$2N/8$	$3N/8$	$6N/8$	$7N/8$
Layer 3				
coefficient a	0	$2N/8$	$4N/8$	$6N/8$
coefficient b	$N/8$	$3N/8$	$5N/8$	$7N/8$

When the difference between the operands of the butterfly is less than 64, only 6 NTT layers are left. Due to the operands' difference being less than 16 in the final three layers, rearranging data within the SIMD data becomes

necessary. With our hardware facilitating internal data shuffling, these 6 layers can be combined to enhance memory access efficiency, as illustrated in Table 4.

TABLE 4
Layer merging of the last 6 layers in GS NTT.

Last Layer 6				
coefficient a	0	8	16	24
coefficient b	32	40	48	56
Last Layer 5				
coefficient a	0	8	32	40
coefficient b	16	24	48	56
Last Layer 4				
coefficient a	0	16	32	48
coefficient b	8	24	40	56
Last Layer 3 w/ input shuffling				
coefficient a	0	16	32	48
coefficient b	8	24	40	56
Last Layer 2 w/ input shuffling				
coefficient a	0	16	32	48
coefficient b	8	24	40	56
Last Layer 1 w/ input shuffling				
coefficient a	0	16	32	48
coefficient b	8	24	40	56

Table 5 shows the merged layers, and memory accesses polynomial lengths ranging from 64 to 1024 in detail. It can be seen that our proposed layer merging significantly reduces the number of memory accesses when compared with the unmerged design.

TABLE 5
Merged layers of polynomial length from 64 to 1024 in GS NTT.

polynomial length	merged layer	# of layers w/ memory accesses	#of reduced memory accesses
1024	Layer 1-3, 4-6, 7-10	3	7 (**70.0%**)
512	Layer 1-3, 4-5, 6-9	3	6 (**66.7%**)
256	Layer 1-3, 4-8	2	6 (**75.0%**)
128	Layer 1-3, 4-7	2	5 (**71.4%**)
64	Layer 1, Layer 2-6	2	4 (**66.7%**)

4.2.2.2 CT NTT: The layer merging process in CT NTT can be seen as the opposite of GS NTT, as the dataflow in CT NTT is essentially the reverse of GS NTT (Figure 3). To restructure the data, output shuffling is implemented in the initial combined layers. Subsequent layers are merged with a maximum of three layers. The specific layer merging approach for various polynomial lengths in CT NTT is outlined in Table 6. Through our layer merging method, memory accesses in CT NTT can be decreased by approximately 70%.

TABLE 6
Merged layers of polynomial length from 64 to 1024 in CT NTT.

polynomial length	merged layer	# of layers w/ memory accesses	# of reduced memory accesses
1024	Layer 1-6, 7-9, 10	3	7 (**70.0%**)
512	Layer 1-6, 7-9	2	7 (**77.8%**)
256	Layer 1-6, 7-8	2	6 (**75.0%**)
128	Layer 1-6, 7	2	5 (**71.4%**)
64	Layer 1-6	1	5 (**83.3%**)

4.2.3 Twiddle Factor Generation

In Hawk's key pair generation, the use of the Residue Number System (RNS) helps in breaking down polynomials into various lengths with different moduli to tackle the NTRU equation efficiently. Due to the significant volume of data involved, it is impractical to pre-compute and store the NTT/INTT twiddle factors in memory. Therefore, in the original Hawk implementation, these twiddle factors are dynamically generated during runtime based on the specific moduli and polynomial length being utilized.

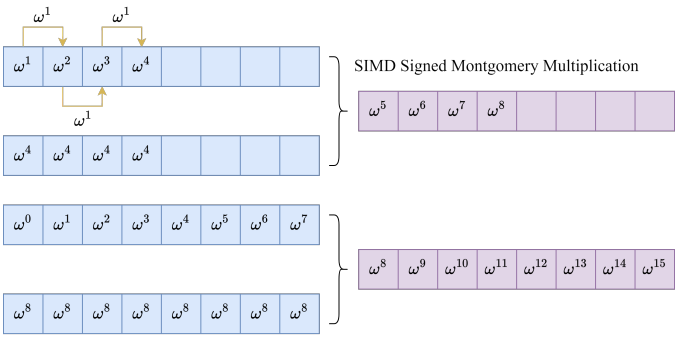

Fig. 4. Computing flow of twiddle factors generation.

To enhance efficiency, SIMD acceleration is employed to decrease the latency associated with twiddle factor computation. The twiddle factors of the final CT NTT layer can be shared with preceding layers, while twiddle factors with higher exponent powers can be derived from those with lower exponent powers using modular multiplications. As a result, the primary twiddle factor ω^1 is pre-computed and stored in memory, and twiddle factors with larger exponent powers are calculated from this primary twiddle factor ω_N^1. As illustrated in the figure depicting twiddle factor generation, ω^2 to ω^4 are derived from ω^1, and ω^5 to ω^8 are computed through SIMD Signed Montgomery multiplication. Subsequent twiddle factors are then obtained from these fundamental twiddle factors by executing additional Signed Montgomery multiplications. By leveraging Signed Montgomery modular reduction in conjunction with our SIMD implementation, an acceleration rate exceeding $10\times$ can be achieved.

4.3 Optimization for Fixed-point FFT/IFFT

FFT and IFFT play crucial roles in the Falcon and Hawk schemes. In the Falcon scheme, computations involve float-point numbers, necessitating hardware support for float-point arithmetic. Conversely, our focus lies on utilizing fixed-point FFT and IFFT in the Hawk scheme. The Hawk scheme employs 32-bit FFT/IFFT for signature verification and 64-bit FFT/IFFT for key pair generation. Additionally, complex numbers are integral to the FFT/IFFT computations. To enhance performance, we introduce the layer merging scheme and hardware-level optimizations tailored for the FFT/IFFT implementations.

For 64-bit number operations, an additional 32-bit carry is needed for higher-order operations. Details of 64-bit number addition and subtraction can be found in Algorithm 6 and Algorithm 7. The SIMD instruction **cmpuv** is used to compare unsigned numbers in the first operand (**rs1**) with those in the second operand (**rs2**), where each 32-bit

Algorithm 6 SIMD implementation of 64-bit addition

Require: 64-bit integers $a = a_1, a_0, b = b_1, b_0$
Ensure: $t = t_1, t_0 = a + b$
1: **addv** t_0, a_0, b_0 // $t_0 = a_0 + b_0$
2: **addv** t_1, a_1, b_1 // $t_1 = a_1 + b_1$
3: **cmpuv** t_c, t_0, a_0 // calculate the lower 32-bit carry, $t_c = 1$ if $a_0 + b_0 > 2^{32} - 1$ (i.e., $t_0 < a_0$)
4: **addv** t_1, t_1, t_c // plus the carry
5: **return** t_1, t_0

Algorithm 7 SIMD implementation of 64-bit subtraction

Require: 64-bit integers $a = a_1, a_0, b = b_1, b_0$
Ensure: $t = t_1, t_0 = a - b$
1: **subv** t_0, a_0, b_0 // $t_0 = a_0 - b_0$
2: **subv** t_1, a_1, b_1 // $t_1 = a_1 - b_1$
3: **cmpuv** t_c, a_0, t_0 // calculate the lower 32-bit carry, $t_c = 1$ if $a_0 - b_0 < 0$ (i.e., $a_0 < t_0$)
4: **subv** t_1, t_1, t_c // minus the carry
5: **return** t_1, t_0

unsigned number in **rs1** is compared with its corresponding 32-bit number in **rs2**.

Algorithm 8 SIMD implementation of 32-bit complex multiplication

Require: 32-bit complex number $a = a_r + a_i j, b = b_r + b_i j$
Ensure: $c = c_r + c_i j$
1: **mulv** $t_r l, a_r, b_r$
2: **mulvh** $t_r h, a_r, b_r$
3: **mulv** $s_r l, a_i, b_i$
4: **mulvh** $s_r h, a_i, b_i$
5: **64-bit subtraction** for $t_r h, t_r l, s_r h, s_r l$ and right-shifted by 32, output c_r
6: c_i is calculated similarly, where the **subtraction** is replace with **addition**
7: **return** c_r, c_i

Given the use of complex numbers, each coefficient in a polynomial is represented by two numbers. Complex number addition and subtraction are performed independently. In the context of 32-bit complex multiplication, the SIMD instructions **mulv** and **mulvh** can yield the lower and higher 32-bit multiplication results, respectively. When executing 32-bit complex multiplication, 64-bit additions and subtractions become essential. The Karatsuba method [35] reduces the number of multiplications from four to three. Nonetheless, our analysis indicates that it necessitates three additional additions, which exhibit a comparable computational latency to multiplications in our proposed processor. The implementation of 32-bit complex multiplication employs conventional complex multiplication, as outlined in Algorithm 8. It is noted that over ten SIMD instructions are utilized per single 32-bit complex multiplication operation.

The 64-bit ordinary multiplication process involves both signed and unsigned multiplications in its computation. When dealing with $a = a_h + a_l \cdot 2^{32}$ and $b = b_h + b_l \cdot 2^{32}$, the 64-bit multiplication process is depicted in Figure 5. This procedure resembles the 32-bit complex multiplication, albeit with minor differences in the addition operations. By employing this 64-bit ordinary multiplication technique, the 64-bit complex multiplication can be efficiently executed using SIMD instructions. This approach involves breaking down the complex multiplication into 64-bit standard multiplication, along with additions, subtractions, and shifts.

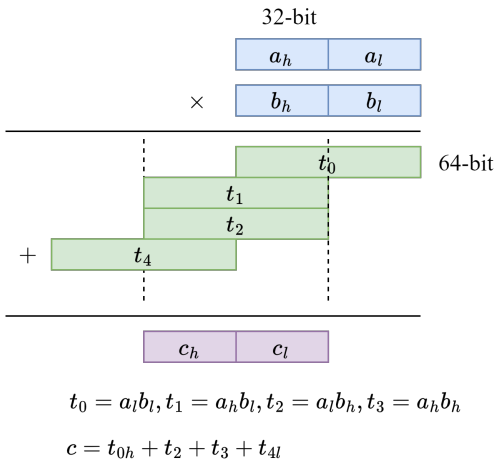

Fig. 5. The SIMD implementation of 64-bit ordinary multiplication.

4.3.1 32-bit FFT/IFFT

The computational process of FFT/IFFT closely resembles that of NTT/INTT, with the key distinction being the absence of modular operations. Building upon the foundational methods for fixed-point calculations, the butterfly operation within a 32-bit FFT/IFFT can be significantly expedited through the utilization of SIMD instructions for 32-bit complex arithmetic, including multiplication, addition, and subtraction. Furthermore, to enhance efficiency, our SIMD implementation also incorporates a technique known as layer merging. Given that two SIMD registers can hold eight 32-bit complex numbers, up to four sets of complex numbers can be accommodated within eight SIMD registers. As a result, the maximum number of layers that can be merged is determined by $\log_2 4 = 2$, leading to an approximate 50% reduction in the total memory accesses.

4.3.2 64-bit FFT/IFFT

The 64-bit FFT/IFFT needs additional SIMD registers to hold the butterfly operands. Since each operand consists of two 64-bit complex numbers, where eight SIMD registers are necessary for one SIMD butterfly operation. Furthermore, four extra SIMD registers are employed to store the relevant twiddle factors. As a result, due to the restricted number of SIMD registers, the layer merging technique cannot be implemented in this scenario.

4.4 Polynomial Auto-adjoint

An auto-adjoint polynomial is defined as a polynomial f where f is equal to its adjoint, denoted as $f = adj(f)$. This condition leads to the property:

$$f[i] = -f[N - i], \text{ for } i > 0 \tag{5}$$

This property essentially reverses the order of the polynomial coefficients and changes their signs. To expedite this reversal process, we introduce a novel SIMD instruction called **subrv** that involves subtracting two SIMD data elements and reversing their order using hardware-supported **reverse shuffling**.

4.5 Rejection Sampling Optimization

Rejection sampling is necessary for $q = 12289$ in Falcon and Hawk schemes. The initial rejection rate stands at $1 - 12289/2^{14} \approx 25\%$. To enhance efficiency, the value $k \cdot q = 174747 \times 12289$ is sampled, resulting in a rejection rate of $1-174747\times12289/2^{31} \approx 99.999\%$. Subsequently, a modular reduction is employed to bring the data back within the q range. This enhancement notably decreases the latency of parallel data rejection sampling.

4.6 RNS Decomposition and CRT Reconstruction

RNS decomposition and CRT reconstruction play crucial roles in Hawk's key pair generation process. These techniques are necessary to handle the large-number calculations involved in solving the NTRU equation effectively. By employing RNS decomposition, the complex computations of large numbers are transformed into numerous parallel calculations involving small numbers. Subsequently, the outcomes of these small-number calculations are reassembled into large numbers through CRT reconstruction.

The large numbers are managed as a 32-bit array in the original approach, which leads to multiple loads and stores during computations. Nevertheless, the existing implementation of RNS decomposition and CRT reconstruction proves to be inefficient and challenging to parallelize. To enhance their performance, we exploit the presence of multiple groups of large numbers that can be restructured to leverage hardware-level parallelism effectively, which has an acceleration rate of over $8\times$. On the other hand, when only a single large number is involved and stored in a 32-bit array, we propose to isolate common operations such as modular reduction and multiplication from RNS decomposition or CRT reconstruction. These operations can then be applied to adjacent data using SIMD instructions, while the remaining operations are executed through non-SIMD instructions, which only demonstrates an acceleration rate of approximately 4 to $8\times$.

4.7 Dual-issue Parser

Our proposed processor enables the simultaneous execution of load/store and arithmetic instructions in the absence of data dependencies. Consider a program P comprising i_{ls} load/store instructions and i_o non-load/store instructions, assuming a CPI of 1 for all instructions and a cycle count of C for program P. Let Δi represent the number of load/store instructions that are unable to take advantage of dual-issue processing. The aim is to maximize the utilization of the dual-issue capability to minimize the cycle count C of program P, while considering the Δi load/store instructions that cannot benefit from dual-issue processing:

$$\begin{aligned} \text{minimize} \quad & C = i_o + \Delta i \\ \text{subject to} \quad & i_o \geq 0, \\ & i_{ls} \geq 0, \\ & i_{ls} \geq \Delta i \geq 0. \end{aligned} \tag{6}$$

To reduce C, it is essential for load/store instructions to exhibit a high level of data independence within their execution context, allowing them to effectively utilize dual-issue processing. However, prior research [29] achieved this through manual programming, which is not an efficient approach. To tackle this challenge, we develop a dual-issue parser that rearranges the instruction sequence without altering functionality. This parser optimizes the instruction order, exploiting dual-issue processing whenever feasible (in the absence of data dependencies), thereby enhancing overall performance. It is important to note that this optimization problem may encounter local minima. To determine a solution with the lowest cycle count, it is necessary to calculate the adjusted cycle count and iterate the tool multiple times.

5 Implementation and Evaluation

In this section, we report and compare the computing cycles of the Falcon and Hawk schemes with those of previous studies. Additionally, we present the evaluation and comparison of the performance, power, and area of the proposed SoC architecture on 28 nm technology, and resource and performance on Zynq-7000 FPGA.

5.1 Performance of Falcon and Hawk

We optimized the code of Falcon and Hawk from the NIST submissions at both the algorithm level and hardware architecture level.

TABLE 7
Cycle counts of Falcon.

	Falcon512.verify	Falcon1024.verify
CV32E40P Baseline	1,029,677 (9.35×)	1,961,000 (10.95×)
[21] (RISC-V)	314,639 (2.86×)	613,911 (3.43×)
[13] (Cortex M7)	559,000 (5.08×)	1,136,000 (6.34×)
[14] (Cortex M4)	504,051 (4.58×)	977,058 (5.46×)
Ours	110,117 (1.0×)	179,080 (1.0×)

5.1.1 Falcon

Since float-point support is lacking, we focused solely on enhancing Falcon's signature verification process, which involves float-point operations for key pair and signature generation. The computational cycles of Falcon are detailed in Table 7. Optimization techniques such as layer merging and the Improved Plantard modular algorithm have been applied to the NTT and INTT operations, while enhanced rejection sampling methods have been implemented to increase the sampling rate for a wider data range. Our optimizations have resulted in a 9-fold enhancement in cycle count for both Falcon512 and Falcon1024 when compared to the baseline performance.

TABLE 8
Cycle counts of Hawk.

	CV32E40P Baseline	[10] (Cortex M4)	**Ours**
Hawk512			
keypair	74,529,506 (2.89×)	52,316,870 (2.03×)	25,801,254 (1.0×)
sign	3,185,113 (7.87×)	2,801,495 (6.93×)	404,481 (1.0×)
verify	2,531,985 (8.13×)	1,418,539 (4.56×)	311,522 (1.0×)
Hawk1024			
keypair	279,095,730 (2.88×)	225,658,496 (2.34×)	96,606,377 (1.0×)
sign	6,625,515 (7.88×)	6,179,673 (7.35×)	840,335 (1.0×)
verify	5,159,475 (8.26×)	3,006,983 (4.81×)	624,621 (1.0×)

TABLE 9
Hardware Synthesis Results and Comparisons.

	Tech. (nm) or FPGA	Volt. (V)	Freq. (MHz)	Area (mm^2) or (DSP/BRAM/FF/LUT)	Falcon512.verif			Hawk512.sign		
					Power (mW)	Perf. (ms)	PPAP	Power (mW)	Perf (ms)	PPAP
[36]	28 (Cortex-M4)	0.9	84	0.053	0.94	6.00	0.30	0.94	33.35	1.66
[21]	22	0.8	800	0.167	4.17	0.39	0.27	-	-	-
Ours	**28**	**0.9**	**200**	**0.102**	**3.05**	**0.55**	**0.17**	**3.27**	**2.02**	**0.67**
[37]	Artix-7	-	-	18/26/17.7k/57.6k	-	0.996	-	-	-	-
[38]	UltraScale+	-	214	15/13/8.0k/11.5k	-	0.618	-	-	-	-
[39]	Artix-7	-	142	1/2/3.5k/6.6k	-	0.049	-	-	-	-
Ours	**Zynq-7000**	**-**	**125**	**37/24/10k/30k**	**-**	**0.88**	**-**	**-**	**3.24**	**-**

5.1.2 Hawk

The cycles of Hawk512 and Hawk1024 are displayed in Table 8. A comparison with the baseline and Cortex M4 implementation is also provided. Due to Hawk's utilization of fixed-point arithmetic, the entire algorithm can be expedited on our proposed platform. We have enhanced the NTT/INTT, fixed-point FFT/IFFT, RNS conversions, and other aspects to boost Hawk's speed. Our optimizations have led to approximately a $7\times$ acceleration in cycle count for both signature generation and verification, surpassing the baseline and M4 implementation. Notably, a speedup of $2.8\times$ for key pair generation has been attained, with the performance bottleneck being the binary GCD algorithm (e.g., around 7 million cycles for binary GCD in Hawk512). Moreover, numerous polynomial operations with lengths less than 8 cannot be effectively optimized in SIMD architecture. Despite key pair generation being a one-time computation in certain scenarios, we have made significant strides in enhancing signature generation and verification.

5.2 Evaluation and Comparison of the Proposed SoC

Table 9 presents the results of ASIC and FPGA synthesis along with comparisons of prior works. Within the table, the Performance, Power, and Area Products (PPAP) are calculated. Our evaluation of performance, power, and area was conducted using Synopsys tools on a 28 nm processing node. The total area occupied by the proposed SoC is 0.102 mm^2, achieving a maximum frequency of 200 MHz. The average power consumption ranges from 3.05 to 3.27 mW at 200 MHz. A comparison was made between our work and recent hardware/software co-design efforts [21] as well as the widely used embedded Cortex-M4. We standardized the power consumption of [21] to the 28 nm technology node as per [40]. Our design demonstrates notable performance improvements over the Cortex-M4, in both Falcon and Hawk. While [21] achieves high performance due to its significantly higher operating frequency, it also incurs larger area and power consumption. Consequently, our design exhibits superior PPAP metrics compared to [21].

The FPGA synthesis results are performed by Vivado 2017.4 on the Zynq-7000 platform, with a maximum frequency of 125 MHz. There is no hardware works for the Hawk scheme. [37]–[39] proposed pure hardware implementations for the signature verification of Falcon. Our performance is not as good as that of pure hardware, as they are more specific for Falcon512 signature verification. However, the main drawback is that these hardware implementations cannot be repurposed for other PQC schemes, and they cannot even be used for the same scheme with different parameter sets. As can be seen, our SoC design is flexible for different PQC schemes and parameter sets with the advantage of hardware/software co-design. Nevertheless, our design still achieve a relatively good performance and achieves a balance of performance, area and power, with the flexibility to support different PQC schemes.

6 Conclusion

In this study, we have proposed PQNTRU, a highly efficient post-quantum processor and SoC tailored for the PQC schemes Falcon and Hawk. We propose a series of optimizations at both the hardware architecture and algorithm levels, focusing on enhancements for NTT, fixed-point FFT, rejection sampling, and other related aspects. Additionally, we have developed an automatic dual-issue parser to maximize the utilization of the hardware's dual-issue capability. Our efforts have led to a significant improvement in cycle counts compared to previous software implementations. When evaluated on a 28 nm ASIC, our design surpasses state-of-the-art solutions in terms of performance, power efficiency, and area utilization. It is worth noting that our work represents a pioneering effort in the realm of hardware/software co-design for the Hawk PQC signature scheme.

References

[1] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th annual symposium on foundations of computer science*. Ieee, 1994, pp. 124–134.

[2] 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," *NIST PQC Round*, vol. 2, no. 4, pp. 1–43, 2019.

[3] V. Lyubashevsky, L. Ducas, E. Kiltz, T. Lepoint, P. Schwabe, G. Seiler, D. Stehlé, and S. Bai, "Crystals-dilithium," *Algorithm Specifications and Supporting Documentation*, 2020.

[4] P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, Z. Zhang *et al.*, "Falcon: Fast-Fourier lattice-based compact signatures over NTRU," *Submission to the NIST's post-quantum cryptography standardization process*, vol. 36, no. 5, pp. 1–75, 2018.

[5] D. J. Bernstein, A. Hülsing, S. Kölbl, R. Niederhagen, J. Rijneveld, and P. Schwabe, "The sphincs+ signature framework," in *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, 2019, pp. 2129–2146.

[6] NIST, "Call for additional digital signature schemes for the post-quantum cryptography standardization process," https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/call-for-proposals-dig-sig-sept-2022.pdf, July 2022, (Accessed on 06/27/2024).

[7] F. Xia, L. T. Yang, L. Wang, A. Vinel *et al.*, "Internet of things," *International journal of communication systems*, vol. 25, no. 9, p. 1101, 2012.

[8] D. J. Bernstein, "Curve25519: new diffie-hellman speed records," in *Public Key Cryptography-PKC 2006: 9th International Conference on Theory and Practice in Public-Key Cryptography, New York, NY, USA, April 24-26, 2006. Proceedings 9*. Springer, 2006, pp. 207–228.

[9] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-speed high-security signatures," *Journal of cryptographic engineering*, vol. 2, no. 2, pp. 77–89, 2012.

[10] Joppe W. Bos, Olivier Bronchain, Léo Ducas, Serge Fehr, Yu-Hsuan Huang, Thomas Pornin, Eamonn W. Postlethwaite, Thomas Prest, Ludo N. Pulles, Wessel van Woerden, , " Hawk: a signature scheme inspired by the Lattice Isomorphism Problem." 2023. [Online]. Available: https://hawk-sign.info/

[11] Y. Kim, J. Song, and S. C. Seo, "Accelerating falcon on armv8," *IEEE Access*, vol. 10, pp. 44 446–44 460, 2022.

[12] M. J. Kannwischer, M. Krausz, R. Petri, and S.-Y. Yang, "pqm4: Benchmarking nist additional post-quantum signature schemes on microcontrollers," *Cryptology ePrint Archive*, 2024.

[13] J. Howe and B. Westerbaan, "Benchmarking and analysing the nist pqc finalist lattice-based signature schemes on the arm cortex m7." *IACR Cryptol. ePrint Arch.*, vol. 2022, p. 405, 2022.

[14] T. Pornin, "New efficient, constant-time implementations of falcon," *Cryptology ePrint Archive*, 2019.

[15] A. C. Mert, E. Öztürk, and E. Savaş, "Design and implementation of a fast and scalable ntt-based polynomial multiplier architecture," in *2019 22nd Euromicro Conference on Digital System Design (DSD)*. IEEE, 2019, pp. 253–260.

[16] J. Mu, Y. Ren, W. Wang, Y. Hu, S. Chen, C.-H. Chang, J. Fan, J. Ye, Y. Cao, H. Li *et al.*, "Scalable and conflict-free ntt hardware accelerator design: Methodology, proof and implementation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[17] Z. Ye, R. C. Cheung, and K. Huang, "PipeNTT: A pipelined number theoretic transform architecture," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 10, pp. 4068–4072, 2022.

[18] N. Zhang, Q. Qin, H. Yuan, C. Zhou, S. Yin, S. Wei, and L. Liu, "Nttu: An area-efficient low-power ntt-uncoupled architecture for ntt-based multiplication," *IEEE Transactions on Computers*, vol. 69, no. 4, pp. 520–533, 2019.

[19] A. C. Mert, E. Karabulut, E. Öztürk, E. Savaş, and A. Aysu, "An extensive study of flexible design methods for the number theoretic transform," *IEEE Transactions on Computers*, vol. 71, no. 11, pp. 2829–2843, 2020.

[20] D. T. Nguyen, V. B. Dang, and K. Gaj, "A high-level synthesis approach to the software/hardware codesign of ntt-based post-quantum cryptography algorithms," in *2019 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2019, pp. 371–374.

[21] P. Karl, J. Schupp, T. Fritzmann, and G. Sigl, "Post-quantum signatures on risc-v with hardware acceleration," *ACM Transactions on Embedded Computing Systems*, vol. 23, no. 2, pp. 1–23, 2024.

[22] J. Hoffstein, J. Pipher, and J. H. Silverman, "NTRU: A ring-based public key cryptosystem," in *International algorithmic number theory symposium*. Springer, 1998, pp. 267–288.

[23] J. Hoffstein, J. Pipher, J. M. Schanck, J. H. Silverman, W. Whyte, and Z. Zhang, "Choosing parameters for NTRUEncrypt," in *Cryptographers' Track at the RSA Conference*. Springer, 2017, pp. 3–18.

[24] J. Hoffstein, N. Howgrave-Graham, J. Pipher, J. H. Silverman, and W. Whyte, "NTRUSIGN: Digital signatures using the NTRU lattice," in *Cryptographers' track at the RSA conference*. Springer, 2003, pp. 122–140.

[25] D. J. Bernstein, C. Chuengsatiansup, T. Lange, and C. van Vredendaal, "NTRU prime: reducing attack surface at low cost," in *Selected Areas in Cryptography–SAC 2017: 24th International Conference, Ottawa, ON, Canada, August 16-18, 2017, Revised Selected Papers 24*. Springer, 2018, pp. 235–260.

[26] D. Stehle and R. Steinfeld, "Making NTRUEnrypt and NTRUSign as secure as standard worst-case problems over ideal lattices," *Cryptology ePrint Archive, Report 2013/004*, 2013.

[27] T. Pornin and T. Prest, "More efficient algorithms for the NTRU key generation using the field norm," in *IACR International Workshop on Public Key Cryptography*. Springer, 2019, pp. 504–533.

[28] D. D. Chen, N. Mentens, F. Vercauteren, S. S. Roy, R. C. C. Cheung, D. Pao, and I. Verbauwhede, "High-speed polynomial multiplication architecture for ring-LWE and SHE cryptosystems," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 62, no. 1, pp. 157–166, 2014.

[29] Z. Ye, R. Song, H. Zhang, D. Chen, R. C.-C. Cheung, and K. Huang, "A highly-efficient lattice-based post-quantum cryptography processor for iot applications," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2024, no. 2, pp. 130–153, 2024.

[30] G. Seiler, "Faster avx2 optimized ntt multiplication for ring-lwe lattice cryptography," *Cryptology ePrint Archive*, 2018.

[31] T. Plantard, "Efficient Word Size Modular Arithmetic," *IEEE Trans. Emerg. Top. Comput.*, vol. 9, no. 3, pp. 1506–1518, 2021. [Online]. Available: https://doi.org/10.1109/TETC.2021.3073475

[32] J. Huang, J. Zhang, H. Zhao, Z. Liu, R. C. Cheung, Ç. K. Koç, and D. Chen, "Improved plantard arithmetic for lattice-based cryptography," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2022, no. 4, pp. 614–636, 2022.

[33] J. Huang, H. Zhao, J. Zhang, W. Dai, L. Zhou, R. 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 Transactions on Information Forensics and Security*, 2024.

[34] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of computation*, vol. 19, no. 90, pp. 297–301, 1965.

[35] A. A. Karatsuba, "The complexity of computations," *Proceedings of the Steklov Institute of Mathematics-Interperiodica Translation*, vol. 211, pp. 169–183, 1995.

[36] Arm Ltd., "Arm Cortex-M4 Datasheet." 2020. [Online]. Available: https://www.arm.com/-/media/Arm%20Developer%20Community/PDF/Processor%20Datasheets/Arm%20Cortex-M4%20Processor%20Datasheet.pdf

[37] D. Soni, K. Basu, M. Nabeel, N. Aaraj, M. Manzano, and R. Karri, *FALCON*. Cham: Springer International Publishing, 2021, pp. 31–41. [Online]. Available: https://doi.org/10.1007/978-3-030-57682-0_3

[38] M. Schmid, D. Amiet, J. Wendler, P. Zbinden, and T. Wei, "Falcon takes off-a hardware implementation of the falcon signature scheme," *Cryptology ePrint Archive*, 2023.

[39] L. Beckwith, D. T. Nguyen, and K. Gaj, "Hardware accelerators for digital signature algorithms dilithium and falcon," *IEEE Design & Test*, 2023.

[40] A. Stillmaker and B. Baas, "Scaling equations for the accurate prediction of cmos device performance from 180 nm to 7 nm," *Integration*, vol. 58, pp. 74–81, 2017.

7 Biography Section

Zewen Ye received his bachelor's degree in microelectronics science and engineering from Zhejiang University in 2020. He is currently pursuing a joint PhD degree at Zhejiang University and the City University of Hong Kong, advised by Prof. Kejie Huang and Prof. Ray C. C. Cheung. His research interests include post-quantum cryptography, hardware design and RISC-V.

Junhao Huang received his Bachelor and Master degrees from Nanjing University of Aeronautics and Astronautics in 2018, and 2021, respectively. He is currently a PhD student at BNU-HKBU United International College. His research interests are public-key cryptography, post-quantum cryptography and cryptographic engineering.

Tianshun Huang received his Master degree from Zhejiang University in 2024. His research interests include post-quantum cryptography and hardware design.

Yudan Bai received his Bachelor degree from Zhejiang University in 2023. He is currently a Master student at Zhejiang University. His research interests are hardware design and RISC-V.

Jinze Li received his Bachelor degree from Zhejiang University in 2023. He is currently a Master student at Zhejiang University. His research interests are hardware design and RISC-V.

Hao Zhang received his Bachelor degree from Zhejiang University in 2023. He is currently pursuing the PhD degree at Zhejiang University, advised by Prof. Kejie Huang. His research interests are post-quantum cryptography, hardware security and design.

Guangyan Li received the B.Eng degree in 2020 from the Department of Electrical Engineering, City University of Hong Kong. He joined the overseas internship scheme to the LIRMM at Montpellier, France from June to August of 2019. He is now pursuing the PhD degree in the Department of Electrical Engineering from City University of Hong Kong. His research interests include reconfigurable computing with FPGA, and post-quantum cryptography algorithm design.

Donglong Chen received the PhD degree from the Department of Electronic Engineering, City University of Hong Kong, in 2015. He was a visiting research scholar of COSIC, KU Leuven, Belgium, in 2013. After completing his PhD degree study, he spent four years with the industry including Huawei Technology Co., Ltd. and Tencent Technology Co., Ltd. He is currently an associate professor with the Faculty of Science and Technology, BNU-HKBU United International College (UIC), China. His research interests include cryptographic engineering, software/hardware co-design for AI algorithms, and privacy computing.

Ray C. C. Cheung received the B.Eng. (Hons) and M.Phil. degrees in computer engineering and computer science & engineering from the Chinese University of Hong Kong (CUHK) in 1999 and 2001 respectively, and the DIC and Ph.D. degree in computing from Imperial College London (IC) in 2007. After completing his Ph.D. study, he received the Hong Kong Croucher Foundation Fellowship and moved to Los Angeles, in the Electrical Engineering department at UCLA, where he spent two years with Image Communication Lab for continuing his research work. He is currently a Professor in the Department of Electrical Engineering at the City University of Hong Kong and with Digital Systems Lab. His current research interests include cryptographic hardware and embedded system designs.

Kejie Huang received the Ph.D. degree from National University of Singapore, Singapore, in 2014. He has been a principal investigator at College of Information Science & Electronic Engineering, Zhejiang University (ZJU) since 2016. Prior to joining ZJU, he spent five years in Samsung and Xilinx, two years in A*STAR, and another three years at Singapore University of Technology and Design (SUTD), Singapore. He has authored or co-authored 50 scientific papers in international peer-reviewed journals and conference proceedings. He holds more than 10 granted patents, and another 20 pending ones. His research interests include hardware acceleration for deep learning and privacy computing, architecture development for post Moore's law era, and computer visions. He is an IEEE Senior Member, the Associate Editor of IEEE TCASII, and the reviewer of many international journals such as Nature Communications, IEEE TCAS, TVLSI, EDL.