Area and Power Efficient FFT/IFFT Processor for FALCON Post-Quantum Cryptography

Ghada Alsuhli, Hani Saleh, *Senior Member, IEEE*, Mahmoud Al-Qutayri, *Senior Member, IEEE*, Baker Mohammad, *Senior Member, IEEE*, and Thanos Stouraitis, *Life Fellow, IEEE*

***Abstract*—Quantum computing is an emerging technology on the verge of reshaping industries, while simultaneously challenging existing cryptographic algorithms. FALCON, a recent standard quantum-resistant digital signature, presents a challenging hardware implementation due to its extensive non-integer polynomial operations, necessitating FFT over the ring $\mathbb{Q}[x]/(x^n+1)$. This paper introduces an ultra-low-power and compact processor tailored for FFT/IFFT operations over the ring for efficient FALCON implementation. The proposed processor incorporates various optimization techniques, including twiddle factor compression and conflict-free scheduling. In an ASIC implementation using a 22 nm GF process, the proposed processor demonstrates an area occupancy of 0.15 mm^2 and a power consumption of 12.6 mW/28.1 mW at an operating frequency of 167 MHz/500 MHz for the non-pipelined/pipelined version of the processor. Since a hardware implementation of FFT/IFFT over the ring is currently non-existent, the execution time achieved by this processor is compared to the reference software implementation of FFT/IFFT of FALCON on a Raspberry Pi 4 with Cortex-A72, where the proposed pipelined processor achieves a speedup up to 3.8×. Furthermore, in comparison to dedicated state-of-the-art hardware accelerators for classic FFT, the pipelined architecture occupies 42% less area and consumes 64% less power, on average. The quantified speedup in the context of FALCON suggests that the proposed hardware design offers a promising solution for the efficient implementation of FALCON.**

***Index Terms*—ASIC, FALCON, FFT/IFFT processor, polynomial operations, post-quantum cryptography.**

I. Introduction

CRYPTOGRAPHY is crucial for ensuring the security of information processing and communication, especially for applications that involve sensitive information, such as online banking, medical devices, and autonomous cars. However, the development of quantum algorithms, like Shor's and Grover's, has shown that commonly used asymmetric key cryptographic schemes, such as the Rivest-Shamir-Adleman (RSA) algorithm, can be easily broken by powerful quantum computers [1]. Although symmetric cryptography can be strengthened against quantum threats by increasing key lengths, the potential impact of quantum algorithms on symmetric encryption still presents a notable concern, especially in future-proofing against advancements in quantum computing [2]. This means that the integrity and confidentiality of information communications could be seriously threatened once such computers become widely available. Several leading companies, including IBM, Intel, and Google, are currently working on developing superconducting quantum processors [3]. Although these quantum computers are not yet powerful enough to pose a threat, they represent a significant step toward the development of more powerful quantum technology in the future.

To prepare for the post-quantum era, a new round of cryptosystem innovation has recently been initiated and become an active research topic [4]. In addition, the National Institute of Standards and Technology (NIST) has launched a post-quantum standardization process for standardizing new Post-Quantum Cryptography (PQC) algorithms that remain secure even in worst-case scenarios when an attacker has a quantum computer. As a result of this standardization process, several Digital Signature (DS) and Key Encapsulation Mechanism (KEM) cryptosystems that are believed to be quantum-resistant have been identified and selected for standardization, such as SPHINCS+, CRYSTALS-KYBER, CRYSTALS-Dilithium, and FALCON. As the NIST PQC standardization process approaches its end, the deployment of PQC algorithms and the evaluation of their hardware efficiency have become major concerns in cryptographic engineering [5]. Many research studies have focused on building dedicated hardware for PQC algorithms [6], [7], [8], [9], [10], aiming either to implement the original PQC algorithms [6], [7], [8] or to implement a lightweight variant of these algorithms that target resource-constrained applications, such as the implementations of binary Ring-Learning with Errors (LWE) that uses binary errors [9], [10]. However, the literature is more focused on a few PQC algorithms, such as CRYSTALS-KYBER and CRYSTALS-Dilithium. This narrow focus is attributed to the simple approach of CRYSTALS algorithms (both based on LWE) which has been studied widely even before the standardization process started. However, other algorithms, such as the FALCON DS scheme [11], have some advantages over CRYSTALS algorithms.

Manuscript received 28 September 2023; revised 6 May 2024; accepted 18 May 2024. Date of publication 7 June 2024; date of current version 27 June 2025. This work was supported by the Khalifa University of Science and Technology under SOCL Grant 2018-018-20 and Grant CIRA-2020-053. *(Corresponding author: Ghada Alsuhli.)*

The authors are with the System-on-Chip Center, Department of Computer and Information Engineering, Khalifa University, Abu Dhabi, UAE (e-mail: ghada.alsuhli@ku.ac.ae; hani.saleh@ku.ac.ae; mahmoud.alqutayri@ku.ac.ae; baker.mohammad@ku.ac.ae; thanos.stouraitis@ku.ac.ae).

Digital Object Identifier 10.1109/TETC.2024.3407124

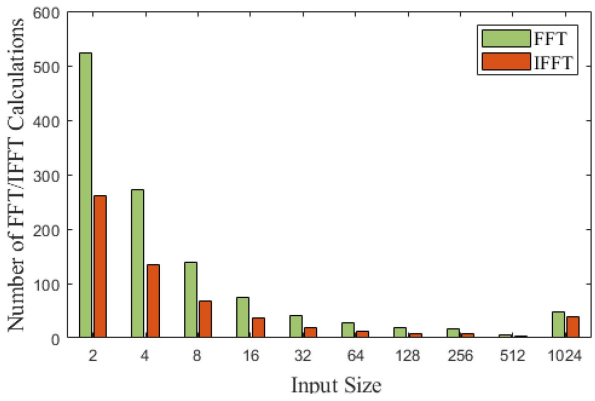


Figure 1. Required FFT/IFFT calculations and the corresponding sizes for FALCON, for security parameter $N = 1024$.

FALCON is a digital signature algorithm known for its quantum security and efficiency in terms of communication bandwidth and verification simplicity. Although FALCON has the smallest signature and public key size and faster verification process compared to other PQC digital signature schemes [12], FALCON has two major challenges to be suitable for resource-constrained devices. First, the key and signature generation parts of FALCON are not naturally hardware-friendly, and this poses a challenge for implementation on devices that have low power budgets, limited computation capabilities, and small memory. Second, because of its use of FP operations, FALCON can be more vulnerable to side-channel attacks. To date, the issue of securing PQC implementations against side-channel attacks remains an open research topic that requires further investigation. This challenge extends beyond FALCON to other PQC algorithms, such as CRYSTALS-Dilithium, as indicated by the NIST report [12]. Our work represents a significant step in addressing the former challenge by presenting an efficient hardware solution for one of the most challenging parts of FALCON.

FALCON consists of three main stages: key generation, signing, and verification. The key generation and signing heavily rely on Fast Fourier Transform (FFT) calculations. In fact, FFT accounts for 26% and 48% of the total clock cycles at key generation and signing processes, respectively [13]. Additionally, to claim meaningful security bounds for FALCON, FFT with double-precision Floating-Point (FP) arithmetic is required [11]. However, this poses a significant limitation for implementing FALCON on devices that lack a Floating-Point Unit (FPU). Moreover, the FFT accelerator should be designed to support all possible FALCON standard parameters, with the main FFT-related parameter being $N \in \{512, 1024\}$, which defines the dimension of the lattice. Even with specific parameters, FALCON uses FFTs of different sizes, which are determined by the parameters of FALCON and are powers of two that divide N.

Figure 1 illustrates the number of required FFT/IFFT per FFT size for FALCON parameter $N = 1024$. It can be observed from this figure that (i) hundreds of FFT/IFFT calculations are required, and (ii) FALCON needs more FFT/IFFT calculations of small sizes. Furthermore, it should be noted that, unlike the typical FFT/IFFT, FALCON uses FFT/IFFT over the ring $\mathbb{Q}[x]/(\phi)$, where ϕ is the irreducible polynomial used in the ring and $\mathbb{Q}$ is the set of all rational numbers. Consequently, custom FFT algorithms are used to implement FFT and Inverse FFT (IFFT) [11]. Therefore, a customized, high-precision, and on-the-fly configurable FFT accelerator that considers different requirements of FALCON while maintaining hardware efficiency is required.

The contributions of this paper are summarized as follows:

- We have examined the specific requirements and unique characteristics of FFT/IFFT over a ring for FALCON PQC. To implement an efficient hardware solution, we utilize the special properties of FFT/IFFT over the ring and adopt an approach inspired by the Negative Wrapped Convolution (NWC) method [14]. This implementation is, to our knowledge, the first of its kind in the literature.
- We propose a Hardware architecture of the processor that relies on a processing elements (PE) array, distributed single-port SRAMs, and distributed ROMs. Each component of the architecture is optimized for power and area efficiency. The proposed architecture offers three levels of reconfigurability. It considers the security parameters, accommodates various input sizes based on the polynomial degree and supports both FFT and IFFT operations.
- We present a conflict-free scheduling algorithm that enables multiple PEs to access the coefficient memory simultaneously without conflicts. This approach maximizes the utilization of the PEs and ensures stalling-free memory access.
- We have compressed the size of the twiddle factor table used in the FALCON reference implementation from 16 KB to just 4 KB for more area and power reduction.
- Employing the Global Foundries (GF) 22 nm process, our Application-Specific Integrated Circuit (ASIC) implementation for the processor exhibits a footprint of 0.15 mm^2 and power consumption of 12.6 mW/28.1 mW at 167 MHz/500 MHz for the non-pipelined/pipelined variants. Given the absence of a current hardware FFT/IFFT implementation over the ring, we benchmarked the processor against the reference software implementation of FFT/IFFT for FALCON on a Raspberry Pi 4 with Cortex-A72. The proposed pipelined processor achieves a noteworthy speedup of up to 3.8×. Additionally, when compared to dedicated state-of-the-art hardware accelerators for classic FFT, the pipelined architecture occupies 42% less area and consumes 64% less power, on average. The quantified speedup within the context of FALCON implies that our hardware design presents a promising solution for the efficient realization of FALCON.

II. Literature Review

In this section, we discuss the available work related to optimized implementations of FALCON and the state-of-the-art FFT/IFFT accelerators.

A. FALCON Implementation Optimization

Several implementations of the FALCON PQC algorithm that aim to optimize its performance for different platforms

and applications have been proposed. In addition to the official reference implementation of FALCON available for multiple platforms, including ARM Cortex-M4 and x86-64 processors [12], there are several optimized software implementations [13], [15], [16].

No full hardware implementation of FALCON exists to date. Available hardware solutions consider implementing the verification part only, which requires only integer, rather than rational, polynomial operations [17], [18], [19], or implementing discrete Gaussian sampling of FALCON [20]. Because the key and signature generations of FALCON are delicate to implement, the authors in [17] chose to implement the hardware of the verification part only. The main focus of this work was to optimize the polynomial multiplication in the integer domain using the Number Theoretic Transform (NTT)[1] and optimize its hardware architecture. Similarly, Field-Programmable Gate Arrays (FPGA) and ASIC implementations and Optimizations for the verification process using High-Level Synthesis (HLS) have been proposed [18]. Since FALCON key generation and signing parts require FP operations and recursive functions, the authors could not use HLS to generate the hardware of these parts. In addition, a hardware-software co-design approach was employed to implement FALCON verification. This resulted in an architecture that incorporates a RISC-V processor integrated with various hardware accelerators, mainly performing the efficient execution of the hash function and polynomial multiplications [19]. In a similar approach, as demonstrated in [20], a hardware-software co-design strategy was employed to implement discrete Gaussian sampling for FALCON. The software module dedicated to discrete Gaussian sampling runs on an ARM Cortex-A9, integrated into an FPGA architecture. This FPGA not only houses the hardware component but also provides the necessary communication bus for seamless interaction between the hardware and software modules.

The lack of full hardware implementation of FALCON is attributed to the facts that: i) FALCON is non-trivial to understand and needs a rigid understanding of the underlying mathematical and security concepts [16], ii) FALCON is a relatively new PQC algorithm, iii) FALCON relies heavily on FFT over the ring calculations in the complex domain which require double-precision FP arithmetic to be used. In this work, we target designing a hardware accelerator for FFT over the ring which is the main expensive operation of FALCON key generation and signing processes. The integration of the proposed accelerator with a FALCON System on a Chip (SoC) or within the pipeline of a FALCON crypto-processor is anticipated to facilitate the development of a complete hardware or hardware-software co-design implementation for FALCON.

B. FFT Accelerators

Hardware acceleration for FFT has been the subject of extensive research over the past few decades [21]. Numerous FFT hardware accelerators have been proposed in the literature [22], [23], [24]. With the growing proliferation of FFT applications, there has been a trend towards customized hardware designs tailored for specific use cases [25], [26], while others aim for a more generic solution to accommodate the requirements of different applications [27], [28]. For instance, several recent accelerator designs involve reconfigurability to enable FFT computations on variable size input [27], [29].In the case of FALCON, it becomes imperative to include such reconfigurability. As depicted in Figure 1, FALCON scheme necessitates extensive FFT operations on polynomials of varying degrees, indicating the need for FFTs with different sizes.

FFT accelerators commonly employ fixed-point, i.e. integer, arithmetic for calculations [30], [31]. However, this approach introduces approximation and low precision to the FFT computations. In contrast, some researchers have opted for single-precision FP (32-bit) representations to enhance FFT accuracy [27], [32], [33], [34]. Although the use of FP arithmetic reduces FFT errors, it comes at the cost of reduced speed, increased power consumption, and greater chip silicon requirements [27]. When it comes to FALCON, the double-precision FP format (64-bit) is essential for proper functionality [11]. This requirement adds further complexity to the design of an FFT specifically tailored for FALCON.

Utilizing FFT-based for polynomial multiplication has become prevalent in security applications, including homomorphic encryption [35], [36] and post-quantum cryptography [37], [38]. These works target the multiplication of polynomials over the ring of integers, in which the FFT becomes an NTT. Unlike other PQC algorithms, FALCON requires polynomial operations to happen frequently in both complex and integer fields, i.e. FFT-based and NTT-based polynomial operations. In this work, we focus on implementing FFT-based polynomial operations in the complex domain. To the best of our knowledge, such an implementation has not yet been presented in the existing literature.

III. Preliminaries

This section focuses on key concepts and algorithms relevant to the classic FFT/IFFT as well as the FFT/IFFT over a ring, which is required for FALCON. Additionally, it discusses the utilization of FFT/IFFT over a ring for polynomial operations in FALCON and explores how to implement it efficiently.

A. Basics of Classic FFT/IFFT

FFT is an efficient algorithm for computing the Discrete Fourier Transform (DFT) of a sequence of n numbers $\boldsymbol{a} \in \mathbb{R}^n$ ($\mathbb{R}$ is the set of real numbers). The classic DFT is defined as $\hat{a}_k = FFT(\boldsymbol{a}) = DFT(\boldsymbol{a}) = \sum_{j=0}^{n-1} a_j \omega_n^{kj}$, where $k = 0, 1, \ldots, n-1$ and $\omega_n = e^{2\pi i/n}$ is a primitive nth root of 1 in the complex domain $\mathbb{C}$, also known as the twiddle factors. The classic inverse DFT is obtained by replacing ω_n with ω_n^{-1} and normalizing the result by n, i.e. $a_j = IFFT(\hat{\boldsymbol{a}}) = IDFT(\hat{\boldsymbol{a}}) = \frac{1}{n} \sum_{k=0}^{n-1} \hat{a}_k \omega_n^{-kj}$,where $j = 0, 1, \ldots, n-1$.

FFT can be calculated using several algorithms. The most commonly used algorithm is the Cooley-Tukey (CT) algorithm which relies on the divide-and-conquer strategy to recursively divide every DFT into two smaller DFTs until we reach 2-point DFTs, which are called butterflies. This is usually called

[1]NTT can be seen as an FFT in a finite field, in which polynomial arithmetic takes place in the integer domain.

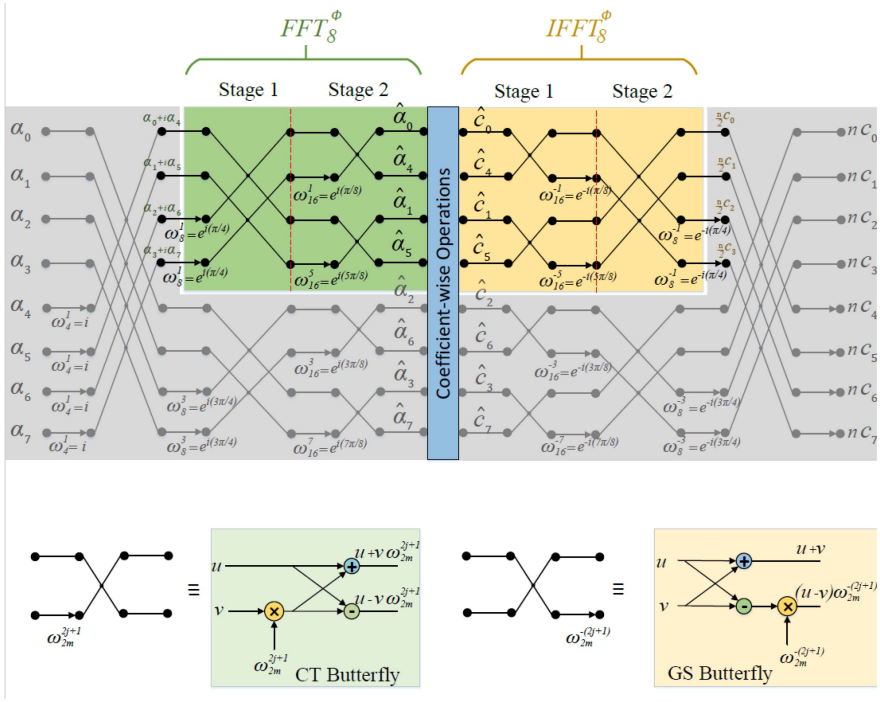


Figure 2. Dataflow of FFT_8^{ϕ} and $IFFT_8^{\phi}$ used for FALCON. The coefficient-wise operations can be any operation to be executed efficiently in FFT representation such as polynomial multiplication and division. The lower part of this figure shows the CT and GS butterflies used for FFT^{ϕ} and $IFFT^{\phi}$, respectively.

Decimation-in-Time (DIT) FFT. Another common algorithm is the Decimation-in-Frequency (DIF) or Gentleman-Sande (GS) algorithm which uses different butterflies. The two butterflies used in these algorithms are illustrated in the lower part of Figure 2.

B. FFT-Based Polynomial Operations

In FALCON, polynomials are defined over the ring $\boldsymbol{R}[x] = \mathbb{Q}[x]/(\phi)$, where $\phi = x^n + 1$ is a cyclotomic polynomial and n is a power of 2. FALCON also presents polynomials over the quotient ring $\mathbb{Z}/q\mathbb{Z}$ with prime q. Polynomial over the ring $\boldsymbol{R}[x]$ are defined by $\boldsymbol{a} = \sum_{i=0}^{n-1} a_i x^i$, where $a_i \in \mathbb{Q}$ are the ith coefficients of the polynomial $\boldsymbol{a}$. Performing additions, subtractions, multiplications, and divisions of polynomials modulo ϕ can be efficiently computed in FFT representations by applying the corresponding operations to each coefficient of the polynomials, i.e. coefficient-wise operations. Next, we investigate using FFT for polynomial multiplication in more detail.

Let $\boldsymbol{a}$ and $\boldsymbol{b}$ be two polynomials of the ring $\boldsymbol{R}[x]$. Using the convolution in the frequency domain, the product $\boldsymbol{c} \in \boldsymbol{R}[x]$ of these two polynomials can be calculated using the classic FFT/IFFT as

$$\boldsymbol{c} = \boldsymbol{a} \times \boldsymbol{b} = [IFFT_{2n}(FFT_{2n}(\boldsymbol{a}) \otimes FFT_{2n}(\boldsymbol{b}))] \bmod \phi. \tag{1}$$

where $\otimes$ denotes coefficient-wise multiplication and the notation $2n$ in $FFT_{2n}(IFFT_{2n})$ indicates the FFT(IFFT) size. Namely, the polynomial multiplication over $\boldsymbol{R}[x]$ can be calculated by applying $2n$-point FFT on the polynomials after padding them, performing coefficient-wise multiplication, applying $2n$-point IFFT on the product, and finally reducing the resultant polynomial to the degree of $n-1$ with the modular polynomial ϕ.

For more efficient exploitation of FFT for polynomial multiplication over $\boldsymbol{R}[x]$, we adopt an approach inspired by the NWC method [14], recently employed in NTT for implementing polynomial operations over the ring $\mathbb{Z}/q\mathbb{Z}$ [37], [39]. However, our application focuses on FFT-based polynomial operations defined over the ring $\boldsymbol{R}[x] = \mathbb{Q}[x]/(\phi)$. To the best of our knowledge, this work is among the first to propose the use of NWC for this specific purpose. This approach allows the application of $n-$point instead of $2n-$point FFT/FFT and alleviates the need to apply the modular reduction. With NWC, the polynomial multiplication in $\boldsymbol{R}[x]$ becomes

$$\dot{\boldsymbol{c}} = IFFT_n(FFT_n(\dot{\boldsymbol{a}}) \otimes FFT_n(\dot{\boldsymbol{b}})), \tag{2}$$

where $\dot{\boldsymbol{a}} = \sum_{i=0}^{n-1} \psi_{2n}^i a_i x^i$ and $\dot{\boldsymbol{b}} = \sum_{i=0}^{n-1} \psi_{2n}^i b_i x^i$ are the two polynomials $\boldsymbol{a}$ and $\boldsymbol{b}$ after applying pre-processing, i.e. scaling, to them. The used scaling factor ψ_{2n} is the $2n$th root of unity with $\psi_{2n}^2 = \omega_n$. Similarly, the actual product is calculated by applying post-processing such that $\boldsymbol{c} = \sum_{i=0}^{n-1} \psi_{2n}^{-i} \dot{c}_i x^i$.

The original NWC requires $3 \times n$ extra complex multiplications for FFT/IFFT-based polynomial multiplication, which is expensive for large n. Fortunately, it has been shown that this scaling can be merged with the twiddle factor when NTT is implemented using the Cooley-Tukey algorithm [40]. Similarly, merging the scaling for INTT is possible and requires switching to the Gentleman-Sande algorithm [41]. Following the same approach, our FFT and IFFT are implemented using two separate algorithms and two different butterflies, to let the twiddle factors absorb the pre-processing and post-processing overheads.

Using the notations FFT^{ϕ} and $IFFT^{\phi}$ to distinguish FFT/IFFT over $\mathbb{Q}[x]/(\phi)$ from the classic FFT/IFFT, the FFT of a polynomial $\boldsymbol{a}$ after applying the NWC becomes

$$\hat{a}_k = FFT_n^{\phi}(\boldsymbol{a}) = \sum_{j=0}^{n-1} a_j \psi_{2n}^{j} \omega_n^{kj} = \sum_{j=0}^{n-1} a_j \omega_{2n}^{j(2k+1)}, \tag{3}$$

while the IFFT is

$$a_j = IFFT_n^{\phi}(\hat{\boldsymbol{a}}) = \frac{1}{n}\sum_{k=0}^{n-1} \hat{a}_k \psi_{2n}^{-k} \omega_n^{-kj} = \frac{1}{n}\sum_{k=0}^{n-1} \hat{a}_k \omega_{2n}^{-k(2j+1)}. \tag{4}$$

It is worth noting that $\omega(k) = \omega_{2n}^{2k+1} = e^{i(\pi/n)(2k+1)}, k = 0, .., n-1$, are the complex roots of ϕ over $\mathbb{C}$ (n distinct roots). Thus, calculating the FFT of a polynomial $\boldsymbol{a} \in \mathbb{Q}[x]/(\phi)$ using NWC is equivalent to evaluating it for the complex roots of ϕ over $\mathbb{C}$, i.e. calculating $\boldsymbol{a}(\omega(k))$.

By applying the divide-and-conquer approach followed by the Cooley-Tukey algorithm, which divides the coefficients based on their index parity, for FFT^{ϕ} in (3), we get

$$\begin{aligned}\hat{a}_k &= \sum_{j=0}^{n/2-1} a_{2j}\, \omega_n^{j(2k+1)} + \omega_{2n}^{(2k+1)} \sum_{j=0}^{n/2-1} a_{2j+1}\, \omega_n^{j(2k+1)},\\ &= \hat{a}_k^1 + \omega_{2n}^{(2k+1)}\, \hat{a}_k^2,\\ \hat{a}_{k+n/2} &= \hat{a}_k^1 - \omega_{2n}^{(2k+1)}\, \hat{a}_k^2,\end{aligned} \tag{5}$$

where $k = 0, 1, \ldots, n$, $\hat{a}_k^1 = \sum_{j=0}^{n/2-1} a_{2j}\ \omega_n^{j(2k+1)}$ and $\hat{a}_k^2 = \sum_{j=0}^{n/2-1} a_{2j+1}\ \omega_n^{j(2k+1)}$ are the $FFT_{n/2}^{\phi}$ of a_{2j} and a_{2j+1}, respectively. This decimation is applied recursively until we reach a FFT_2^{ϕ}. Similarly, the Gentleman-Sande algorithm can be applied to (4) for the IFFT case. Figure 2 shows the data flows of FFT_8^{ϕ} and $IFFT_8^{\phi}$ as an example.

$\omega(k)$ has several properties that further simplify the calculation of FFT^{ϕ} and $IFFT^{\phi}$. First, $\omega(n-1-k) = \text{conjugate}(\omega(k)) = 1/\omega(k)$, where conjugate$(x)$ is the complex conjugate of x. Given that the polynomial $\boldsymbol{a}$ is in $\mathbb{Q}[x]/(\phi)$, it follows that $\boldsymbol{a}(\omega(n-1-k)) = \text{conjugate}(\boldsymbol{a}(\omega(k)))$. As a result, it is sufficient to evaluate $\boldsymbol{a}(\omega(k))$ for $k = 0, \ldots, n/2-1$, which reduces the storage space and calculation requirements for $FFT^{\phi}/IFFT^{\phi}$ by a factor of 2 [11]. The remaining $\boldsymbol{a}(\omega(k))$ for $k = n/2, \ldots, n-1$ can be easily calculated, if necessary.

The second useful property of $\omega(k)$ is that when the final recursion stage is reached, the twiddle factor to be evaluated is $\omega_4^1 = i$. Hence, we can think about the first stage of FFT flow as transforming the FFT input $\in \mathbb{Q}$ into the complex domain such that the first stage can be eliminated and the input of the second stage is $a_k^{(0)} = a_k + i\ a_{k+n/2}$ for $k = 0, \ldots, n/2-1$. This step is easily done in hardware by separating the storage of the imaginary and real parts and storing the second $n/2$ of the input as an imaginary part. The discussion above is also applicable to the case of IFFT. We should mention here that to consider the impact of the ignored redundant half of the FFT in the final stage of IFFT, the real and imaginary values in the output should be duplicated. This is compensated by normalizing the IFFT output over $n/2$ instead of n.

Consequently, we can think about $FFT_n^{\phi}/IFFT_n^{\phi}$ as an FFT/IFFT of $n/2$ complex numbers with special twiddle factors. This reduces the size of FFT by a factor of 2, as illustrated in Figure 2, where the grey part of the dataflow shows the reduced unnecessary calculation because of exploiting the two properties of $\omega(k)$ mentioned above. The $FFT_n^{\phi}/IFFT_n^{\phi}$ algorithm consists of $\log_2(n) - 1$ stages with each stage involving $n/4$ butterfly operations.

Figure 2 reveals that the order of stage 2 in FFT (and stage 1 in IFFT) differs from the natural order derived in (5). This variation in the ordering is due to FALCON's original software implementation [42], where this specific order was chosen to simplify the software computations. While this ordering produces different results, it functions correctly as long as the same order is maintained for both FFT and IFFT [11]. Although our hardware implementation does not require this order, we adopt the same ordering for easier verification of our hardware design by comparing it to the FALCON software implementation.

One problem with using CT FFT and GS IFFT is that the CT usually requires the FFT input to be in bit-reversed[2] order and produce the FFT output in the natural order. On the other hand, the opposite happens with GS IFFT. Thus, the bit-reversing step is needed before FFT and after IFFT when calculating the polynomial operations. To overcome this, we follow an approach similar to the one observed for NTT in [43], in which the CT and GS algorithms are manipulated and the butterflies are used independently, as illustrated in Figure 2. This alleviates the need for any bit-reversal calculation at the beginning or end of the polynomial operation calculation. Further manipulation of these algorithms is needed in our processor to obtain conflict-free scheduling, as we will see in Section D.

IV. Design Methodology

In this section, we describe the design methodology of our processor hardware implementation. First, we explain the run-time reconfigurability of our processor to accommodate the requirements of FALCON. Second, we provide an in-depth discussion of the architecture adopted for our processor. Furthermore, we discuss the various optimization techniques employed in our design, including twiddle factor preprocessing and conflict-free scheduling.

A. Reconfigurability of $FFT^{\phi}/IFFT^{\phi}$ Processor

The proposed design of the $FFT^{\phi}/IFFT^{\phi}$ processor for FALCON incorporates three levels of reconfigurability.

First, the design takes into account all security parameters that impact the processor design. The primary parameter of interest is

[2]The bit-reversed order refers to the rearrangement of the input sequence of data points, swapping their positions in a binary reflection pattern.

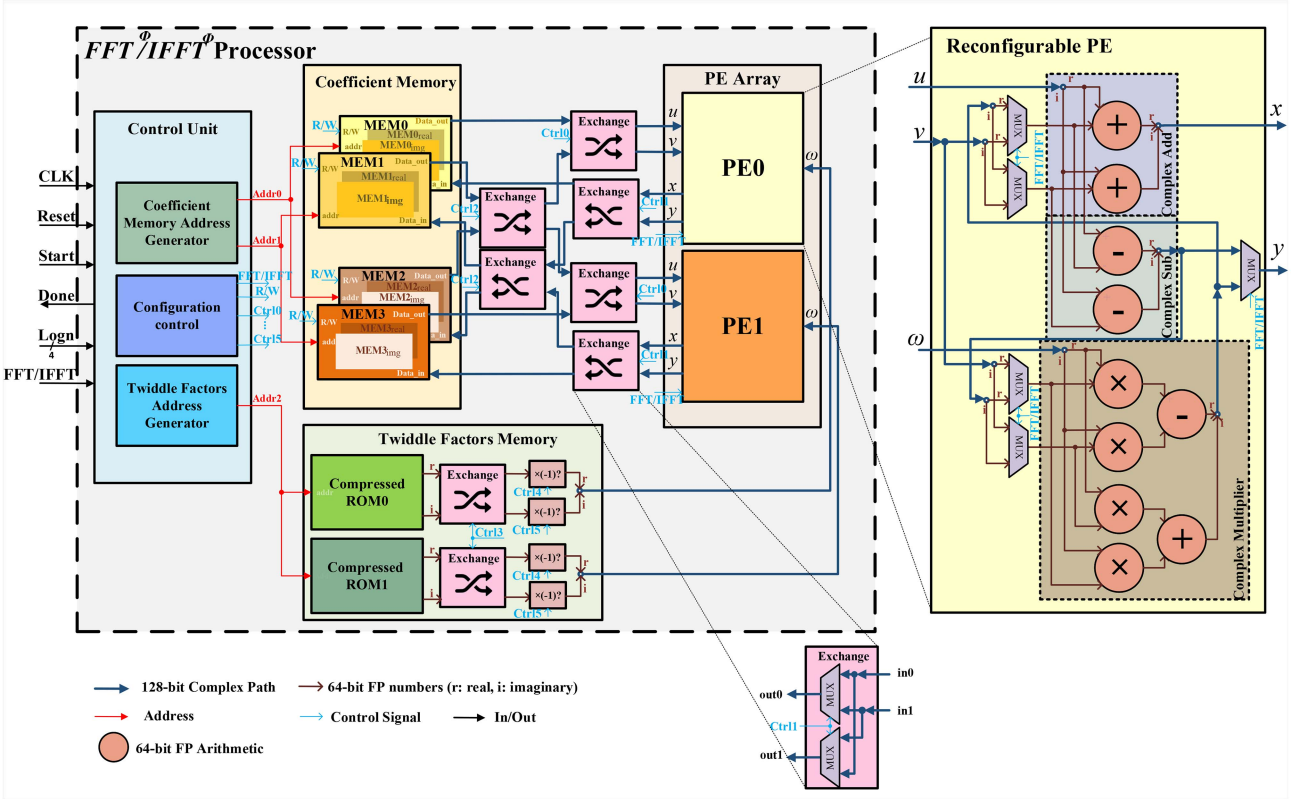


Figure 3. The architecture of the proposed $FFT^{\phi}/IFFT^{\phi}$ processor that involves an array of PEs, distributed coefficient memory banks, distributed twiddle factor ROMs, and a control unit. The internal design of the reconfigurable PE is illustrated on the left.

the lattice dimension N, which determines the quantum hardness of FALCON. Our processor is designed to support both security levels of FALCON, namely NIST Level I and V, which require N to be either 512 or 1024, respectively. As a result, the maximum size of $FFT^{\phi}/IFFT^{\phi}$ supported by our processor is 1024.

The second level of reconfigurability arises from the key generation process in FALCON, which involves calculations such as NTRU equations and fast Fourier sampling. These calculations begin with a polynomial of degree 512 or 1024, depending on the security level, and the polynomial's degree is halved iteratively until a single rational coefficient is obtained [13]. Consequently, our processor is optimized to dynamically adapt to various FFT sizes $FFT_n^{\phi}/IFFT_n^{\phi}$ with $\log(n) \in \{1, 2, \ldots, \log(N)\}, N \in \{512, 1024\}$.

Third, the proposed processor is designed to handle both FFT^{ϕ} and $IFFT^{\phi}$, which utilize the CT and GS algorithms. Therefore, our processor can be configured as an FFT^{ϕ} processor or an $IFFT^{\phi}$ processor, requiring appropriate adjustments to the control flow, memory access, and datapath.

B. Proposed Overall Architecture

We adopt an architecture that involves an array of processing elements, distributed coefficient memory banks, distributed twiddle factor read-only memory (ROM), and a control unit. An overview of the proposed architecture is illustrated in Figure 3.

1 Reconfigurable PE Array: Utilizing a PE array architecture in the design of our FFT accelerator proves advantageous due to its inherent features, including parallel processing, scalability for different FFT sizes, and the ability to optimize performance. The extensive adoption of PE arrays in recent hardware accelerators further motivated our decision to employ this technique for our hardware implementation of the $FFT^{\phi}/IFFT^{\phi}$ processor. When the number of PEs in the array is n_{PE}, the architecture is expected to perform n_{PE} PE operations on every step.

Each PE in our design is adaptable depending on whether the target task of the processor is to perform FFT^{ϕ} or $IFFT^{\phi}$. This is determined by the $FFT/IFFT$ signal. When the signal indicates FFT, the PE executes radix-2 CT butterfly operations. On the other hand, when the signal indicates $IFFT$, the PE performs GS butterfly operations. Thus, the PE has three inputs (u, v, ω) and two outputs (x, y) and performs radix-2 CT or GS butterfly operations.

Figure 3 shows the implementation of resource-efficient reconfigurable PE. Since CT and GS butterflies involve similar operations such as complex addition, subtraction, and multiplication, but in different orders, we utilize multiple multiplexers (MUXs) to facilitate the required reordering and reduce the overhead of duplicating hardware-unfriendly operations. The reconfigurable butterfly within the PE consists of six 64-bit FP adders/subtractors and four 64-bit FP multipliers, following the IEEE-754 double-precision FP standard.

The choice of the number of PEs in the processor significantly influences its performance. Increasing the number of PEs can lead to faster execution for larger FFT sizes, but it comes at the expense of increased area and power consumption that are

not fully utilized at all times. Considering the characteristics of FFT operations required for FALCON and based on the analysis shown in Figure 1, we observed that a substantial majority (68%) of the required FFT operations for FALCON have relatively small sizes ($n \leq 8$). These smaller FFT sizes can be efficiently handled by a smaller number of PEs (n_{PE}). This allows to efficiently handle the majority of FFT operations for FALCON while minimizing the area and power consumption overhead associated with additional PEs.

2 Distributed Coefficient Memory: The proposed architecture utilizes multi-bank single-port SRAMs to store the initial polynomial coefficients, intermediate calculations, and the final results. Although with single-port SRAMs conflicts can arise when two PEs attempt to read from or write to the same memory bank simultaneously, it is more area and power-efficient than the dual-port one. To avoid memory read and write conflicts, it is essential to ensure that the two coefficients processed by each PE in every round are read from different memory banks and that the two output values are written to different memory banks. The minimum number of required memory banks denoted as M, is calculated as $M = 2 \times n_{PE}$. The size of each memory bank is given by $S_{max}/(2 \times M)$, where $S_{max} = 1024$ is the maximum $FFT^{\phi}/IFFT^{\phi}$ size. However, conflict is still possible unless proper conflict-free scheduling of the multi-PE is adopted. Details of this conflict-free scheduling approach will be discussed in Section D of the paper.

3 Distributed Twiddle Factors Memory: In the design of our processor, we encountered two options for implementing the twiddle factors: precomputing and storing them in memory (typically ROM), or calculating them on the fly during runtime. The first approach, involving precomputed twiddle factors, is more advantageous for the following reasons:

First, the twiddle factors required for lower FFT^{ϕ} sizes are subsets of those needed for larger sizes. Therefore, we only need to store the twiddle factors required for the maximum FFT^{ϕ} size (1024). Furthermore, when computing the $IFFT^{\phi}$, we can utilize the conjugates of the twiddle factors used in the FFT^{ϕ}, eliminating the need to store separate sets of twiddle factors.

Second, computing the twiddle factors on the fly during runtime would impose significant double-precision FP computational overhead, introducing substantial processing delays and negatively impacting the overall performance of the FALCON $FFT^{\phi}/IFFT^{\phi}$ processor.

Lastly, we have introduced an optimization technique that allows the reduction of the size of twiddle factor memory compared to the original implementation of FALCON. This optimization enables the minimization of the memory footprint and further enhances the efficiency of the FFT processor design. In addition, each PE uses a different set of twiddle factors which allows the distribution of the memory and the use of separate ROM for each PE. Hence, we need a total of n_{PE} ROMs with a typical size of $S_{max}/(2 \times n_{PE}) + 1$ for each ROM. These optimizations are discussed in Section C.

4 $FFT^{\phi}/IFFT^{\phi}$ Control Unit: The controller plays a crucial role as it oversees the efficient execution of butterfly operations within the PE array. It is implemented using a Finite State Machine (FSM) and is responsible for coordinating the scheduling of operations, ensuring the optimal utilization of the PEs. One of the primary tasks of the controller is to generate the appropriate addresses for accessing the coefficient memory banks and twiddle factor ROMs. By dynamically generating these addresses, the controller facilitates the seamless retrieval of the required data for each butterfly operation. Furthermore, the controller is designed to be flexible and adaptable, accommodating different configurations of the processor. It can handle various operations, such as performing either FFT^{ϕ} or $IFFT^{\phi}$, depending on the desired functionality. Additionally, it can adapt to different input sizes, allowing the processor to handle FFT operations of varying dimensions.

Algorithm 1: Twiddle Factors Permutation Algorithm.

Require: ω, n
Ensure: $\hat{\omega}$
1: **for** $i = \log_2(n) - 1$ down to 3 **do**
2: **for** $j = 2^{i-2} + 2^i + 1$ to n **do**
3: $\omega \leftarrow \boldsymbol{PermutateBlocks}(\omega, j, 2^{i-3})$
4: $j \leftarrow j + 2^{i-1}$
5: **end for**
6: **end for**
7: $\hat{\omega} \leftarrow \omega$

C. Twiddle Factors Preprocessing

In the reference software implementation of FALCON, the twiddle factors were stored in a table using a bit-reversed order scheme, requiring a 16 KB ROM to store the twiddle factors. However, in our approach, we utilize a different indexing scheme that only requires half of the twiddle factors to be stored. This reduces the ROM storage requirement to 8 KB. Additionally, to facilitate the indexing in our conflict-free scheduling algorithm explained in the next section, we permute the order of the remaining twiddle factors to match the permutation caused by applying the conflict-free scheduling algorithm. Indeed, some stages of the FFT and IFFT necessitate a specific ordering of the twiddle factors.

Algorithm 1 presents the permutation scheme used to achieve the required ordering of twiddle factors in case $n_{PE} = 2$. The algorithm utilizes the function $\boldsymbol{PermutateBlocks}(x, st, sz)$, where x is the vector to be processed, st is the starting index of the block to be permuted, and sz is the size of the block. This function swaps the specified block with the subsequent block of the same size. Specifically, for our twiddle factors, it swaps $\omega(j : j + 2^{i-3} - 1)$ with $\omega(j + 2^{i-3} : j + 2^{i-2} - 1)$. It is important to note that the preprocessing of the twiddle factors occurs offline before storing them in the ROMs, resulting in no additional overhead.

Subsequently, we divide the ROM into n_{PE} ROMs to distribute them in a way that avoids conflicts when multiple PEs attempt to access the same ROM. Noting that each pair of consecutive twiddle factors in the ROM consists of $\omega(k)$ and $\omega(k \pm \frac{\pi}{2})$, We can further compress each ROM by a factor of 2. These twiddle factors can be derived from each other by multiplying $\omega(k)$ by i or $-i$. We only store the twiddle

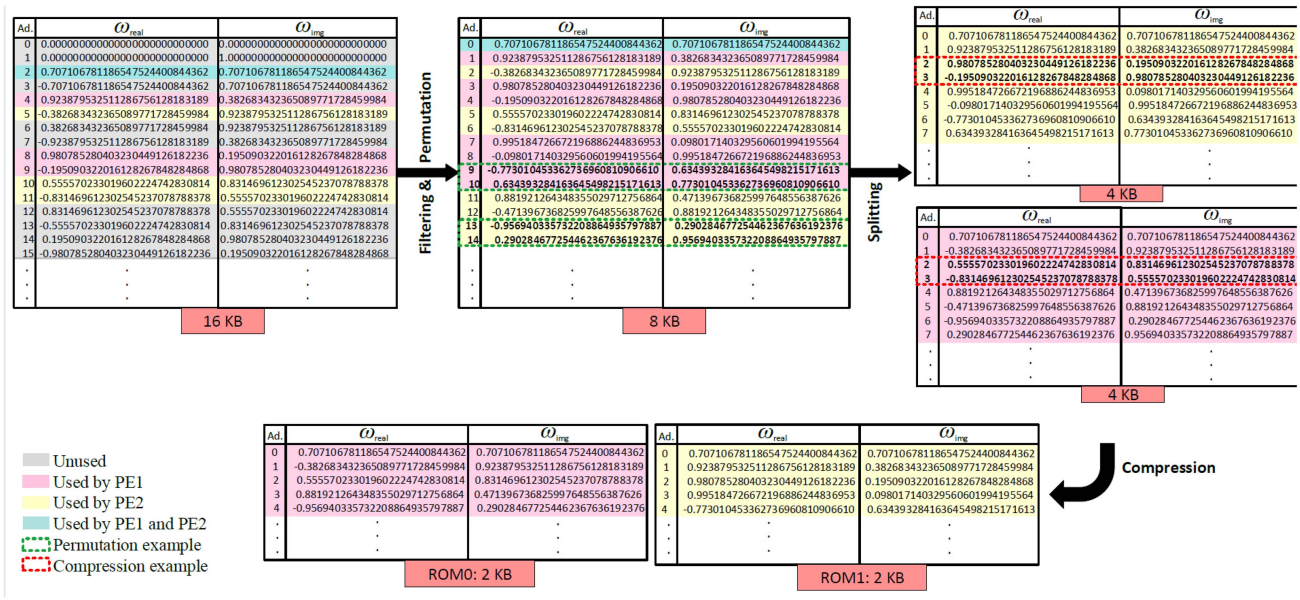


Figure 4. Twiddle factor preprocessing through filtering and permutation, along with ROM splitting and compression for the case when $n_{PE} = 2$. Illustrations demonstrate instances where permutation and compression techniques are applied.

factors of even indices in the ROM. The twiddle factors of odd indices can be easily calculated in hardware by swapping the real and imaginary parts (using MUXs) and negating the sign bit of one of them. Since the coefficient RAM and the ROMs are read simultaneously in the same clock cycle, the introduced MUXs and negation do not result in any additional delay on the critical path. Hence, this calculation does not impact the overall performance. As a result, the overall twiddle factor ROM size is reduced by a factor of four compared to the original FALCON implementation, with almost no overhead. Figure 4 illustrates an example of the proposed preprocessing and compression of the twiddle factor ROM when $n_{PE} = 2$.

D. Conflict-Free Scheduling

Memory conflicts may arise in the architecture when two PEs simultaneously request access to the same memory bank. We address this type of conflict by allocating separate memory banks for each PE to avoid contention. However, a second source of conflict is when both PE inputs are in the same memory bank. This happens because, as illustrated in Figure 2, the distance between the indices of the two inputs of each butterfly becomes smaller for the latter stages of the FFT^{ϕ}, and the earlier stages of $IFFT^{\phi}$. To resolve this conflict, our architecture incorporates a conflict-free scheduling algorithm (Algorithm 2) that intelligently coordinates the assignment of butterfly operations to the PEs and manages the associated memory accesses, ensuring smooth data flow and avoiding conflicts during the execution of FFT^{ϕ} or $IFFT^{\phi}$ operations. Our conflict-free scheduling technique, while sharing the overarching concept with several works such as the one presented in our prior work [31] and other recently introduced algorithms such as in [37], distinguishes itself in terms of implementation details. Specifically tailored for FALCON FFT/IFFT, our approach features unique indexing of the coefficients and a distinct data flow. Before applying the approach from [31] designed for classic FFT to our design for FALCON FFT/IFFT, we needed to redesign the indexing and data flow to align with the requirements of FALCON. Moreover, our implementation takes a highly hardware-friendly approach in generating indexes and addresses, in addition to utilizing single-port instead of dual-port RAM compared to [37], aimed at minimizing both area and power consumption. This emphasis on hardware efficiency is evident in the minimal overhead of our control unit, constituting less than 1% of the overall area and power consumption in our design as we will see later.

The main idea of our conflict-free scheduling algorithm is to predict whether a conflict will appear in the next stage and take action to avoid it. Let the initial inputs of FFT_n^{ϕ}, denoted as $a_k^{(0)} = a_k + i\, a_{k+n/2}$, where $a_k \in \mathbb{Q}$ and $a_k^{(0)} \in \mathbb{C}$ be arranged in the memories MEM_0 to MEM_{M-1}[3] in the natural order, such that $k = 0, \ldots, n/2 - 1$. In each stage, $n/4$ butterfly operations are performed, where $bt \in 0, \ldots, n/4 - 1$ represents the butterfly operation index in the current stage. In a specific stage sg, a PE_j, $j = 0, \ldots, n_{PE} - 1$ performs BT_{PE} butterfly operations from $bt = j \times n/(4 \times n_{PE})$ to $bt = (j+1) \times n/(4 \times n_{PE}) - 1$. Let $\delta^{sg} = |q - d|$ be the absolute difference between the indices d and q of the coefficients $a_d^{(sg)}$ and $a_q^{(sg)}$ whose butterfly operation is allocated to a specific PE in the sg-th stage. It can be expressed as:

$$\delta^{sg} = \frac{n}{2^{sg+2}}. \tag{6}$$

For FFT_n^{ϕ}, this distance starts at $n/4$ for the first stage and is divided by two for each subsequent stage. The conflict occurs only when $a_d^{(sg)}$ and $a_q^{(sg)}$ fall in a single memory space, i.e., δ^{sg}

[3]Actually, MEM_i for $i = 0, \ldots, M-1$ consists of two separate memories MEM_i^{real} to store the real part and MEM_i^{imag} to store the imaginary part.

Algorithm 2: Conflict-Free Scheduling Algorithm.

Require: n, n_{PE}, ω in ROMs, $\boldsymbol{a}$ in MEMs
Ensure: $FFT_n^{\phi}(\boldsymbol{a})$ in MEMs
$Logn \leftarrow \log_2(n) - 1$ ▷ # stages
$BT_{PE} \leftarrow \frac{n}{4n_{PE}}$ ▷ # operations per PE
$S_M \leftarrow \frac{n}{4n_{PE}}$ ▷ Memory size
$S_{sg} \leftarrow \log_2(n_{PE})$ ▷ Largest safe stage
for $sg = 0$ to $(Logn - 1)$ **do**
  for $bt_{PE} = 0$ to $(BT_{PE} - 1)$ **do**
    for $PE_i = 0$ to $(n_{PE} - 1)$ **do**
      $bt \leftarrow BT_{PE} \cdot PE_i + bt_{PE}$
      $Addr_0, Addr_1 \leftarrow \mathbf{MEM\_addr}(bt_{PE}, sg)$
      $i0, i1 \leftarrow \mathbf{MEM\_Select}(sg, bt, S_M)$
      $sg_r \leftarrow Logn - sg - 1$ ▷ sg reverse
      $gp \leftarrow \lfloor \frac{bt}{2^{sg_r}} \rfloor$ ▷ Operation group in sg
      if gp is even **and** $sg \leq S_{sg}$ **then**
        $u \leftarrow \text{MEM}_{i0}[Addr_0]$
        $v \leftarrow \text{MEM}_{i1}[Addr_1]$
      else ▷ Input exchange
        $v \leftarrow \text{MEM}_{i0}[Addr_0]$
        $u \leftarrow \text{MEM}_{i1}[Addr_1]$
      end if
      $Addr2 \leftarrow bt_{PE}$
      $\omega \leftarrow \text{ROM}_{PE_i}[Addr2]$
      $[y, x] \leftarrow \text{PE}_{PE_i}(u, v, \omega)$
      $EX\_Bit \leftarrow n - sg - 2$
      $EX\_Flag \leftarrow bt[EX\_Bit]$
      if $sg \geq S_{sg}$ **and** $EX\_Flag = 1$ **then**
        $\text{MEM}_{i0}[Addr_0] \leftarrow y$ ▷ Output exchange
        $\text{MEM}_{i1}[Addr_1] \leftarrow x$
      else
        $\text{MEM}_{i0}[Addr_0] \leftarrow x$
        $\text{MEM}_{i1}[Addr_1] \leftarrow y$
      end if
    end for
  end for
end for

is less than the size of the memory bank $n/(2 \times M)$. We refer to the pair $(a_d^{(sg)}, a_q^{(sg)})$ as a Conflict-Prone (CP) pair. Hence, this conflict occurs from stage $\log_2(n_{PE}) + 1$ to the last stage, which we call CP stages. We define $S_{sg} = \log_2(n_{PE})$ as the largest conflict-free (safe) stage.

The procedure followed in this algorithm involves identifying the CP stages and exchanging the inputs and/or outputs of some PE operations to avoid conflicts. If the present data pair creates a hazard in the next stage, the PE outputs are exchanged. If PE inputs have been reversed previously in memory, the PE input is exchanged. This is implemented by dividing the butterfly operations into groups, with the group number varying across different stages. The current group index, gp, and the test for whether this stage is safe or not are used to determine if the inputs of the PEs should be exchanged. On the other hand, a specific bit, EX_Bit, of the current butterfly operation index

Algorithm 3: Coefficient Memory Address Calculation.

Require: bt_{PE}, sg, S_{sg}
Ensure: PE memory addresses $Addr_0, Addr_1$
$Addr_0 \leftarrow bt_{PE}$
if $sg \leq S_{sg}$ **then**
  $Addr_1 \leftarrow Addr_0$
else
  $k \leftarrow S_{sg} + 1$
  $l \leftarrow sg$
  $Addr_1 \leftarrow \mathbf{Rev}(Addr_0, k, l)$
end if

bt is observed as an indication of whether the output of the PE needs to be exchanged in the CP stages.

The **MEM_Select**(sg, bt, S_M) function in Algorithm 2 aims to select which two memory banks are allocated to the PE, and they are different when the stages are CP or not. Because data is rearranged in memory, the algorithm needs to keep track of where the data is. In our approach, the data is moved in memory in a specific pattern to easily track the movement of data from one stage to another. To simplify the hardware implementation of this approach, we observe the pattern by which the coefficients should be exchanged in memory. As a result, the coefficient memory addresses to be accessed by the PEs are calculated in a hardware-friendly way, as illustrated in Algorithm 3. For the safe stages, the PEs access the two associated memory banks with addresses $Addr_0$ and $Addr_1$, where $Addr_0$ is the index bt_{PE} of the PE butterfly operation. However, for CP stages, the second address $Addr_1$ is calculated by reversing some bits of the first address $Addr_0$. The **REV**$((Addr_0, k, l))$ function reverses the bits of $Addr_0$ that start with bit k and end with bit l. On the other hand, calculating the ROM address is very simple since a fixed single ROM is associated with each PE. In addition, the twiddle factors in ROMs are preprocessed to be accessed in the same butterfly operation order.

Figure 5 illustrates the conflict-free 32-point FFT/IFFT process using two PEs as an example. It showcases the initial, intermediate, and final memory contents, along with the scheduling of PEs and the triggered exchange operations. Notably, for the IFFT case, the scheduling and control mirror those of the FFT. This simplifies the integration of IFFT with FFT by primarily reversing the stage counter. It can be observed from Figure 5 that after performing the FFT, the coefficients become disordered in memory. However, since all calculations in FALCON are coefficient-wise operations on the transformed polynomial, this disordering has no impact on the accuracy of these computations. Likewise, during the IFFT, which follows the same scheduling and control as FFT, the coefficients are reordered again to their original order. By using this conflict-free scheduling, a very simple and hardware-friendly design of the control unit is obtained.

V. Processor Performance Evaluation

To evaluate the proposed $FFT^{\phi}/IFFT^{\phi}$ processor we implement it based on the parameters listed in Table I. Given

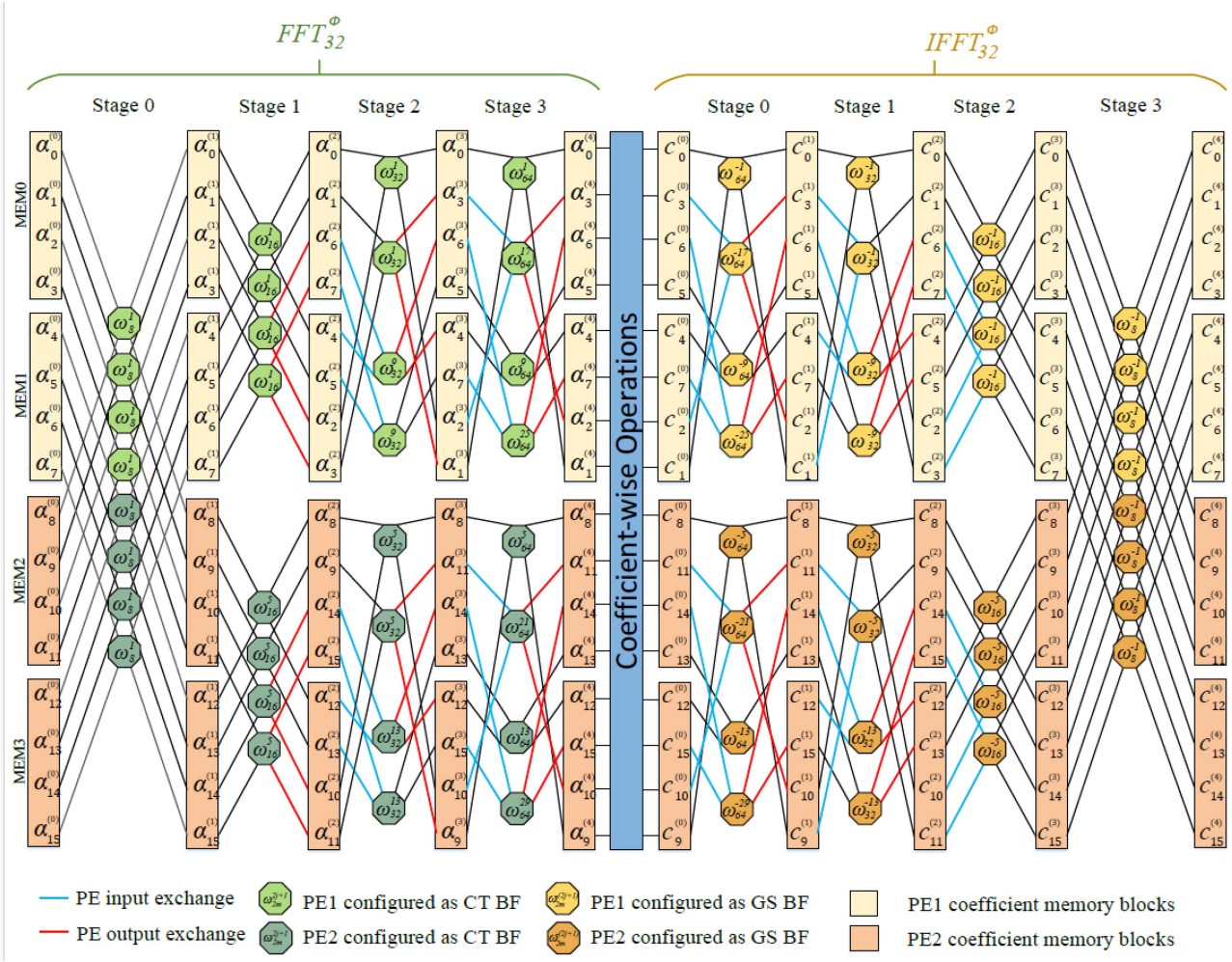


Figure 5. Conflict-free Scheduling of PE Operations for FFT_{32}^{ϕ} and $IFFT_{32}^{\phi}$ with $n_{PE} = 2$. The figure illustrates the initial, intermediate, and final memory contents, along with the scheduling of PEs and the triggered exchange operations. In the IFFT case, the scheduling and control mirror those of the FFT.

TABLE I
FFT IMPLEMENTATION PARAMETERS

Processor parameters	
n_{PE}	2
Largest FFT size S_{max}	1024
FALCON Security level	I and V
VLSI Implementation parameters	
Process	GF FD-SOI ®22 nm
Supply voltage	0.72 V
Temperature	-40C∼125C
CLK frequency	167 MHz/500 MHz
SRAM number×size	8 × 1KB
ROM number×size	2 × 2KB

the focus on optimizing power and area efficiency, we select n_{PE} to be 2 such that two butterflies are performed in parallel. Each butterfly consists of four double-precision FP multipliers and six double-precision FP adders/subtractors. To design these FP modules, we utilized the DesignWare library from Synopsys, which includes essential infrastructure IP for design and verification. Specifically, we employed the DW_fp_mult FP multiplier and DW_fp_addsub FP Adder/Subtractor from this library. These IPs comply with the IEEE 754 Floating-point standard, and we configured them to exclude subnormals as they are not used in FALCON.

This processor supports all security levels of FALCON since it accepts the FFT/IFFT input to be between $n = 4$ and $n = S_{max} = 1024$ (When $n = 2$, the output of the processor is already calculated, i.e., $a_0^{(0)} = a_0 + i\ a_1$). When $n = 4$, only one of the two butterflies is used while the other is idle. Otherwise, always the two butterflies participate in the calculation simultaneously.

To highlight the trade-off between our emphasis on power efficiency and execution time, we implemented another version of our processor. This version is optimized not only for power and area efficiency but also for a higher frequency. We achieved this optimization by simply adding a two-stage pipeline to our processing elements to break the critical path (thus, we refer to it as the pipelined version). Additionally, we specified a higher

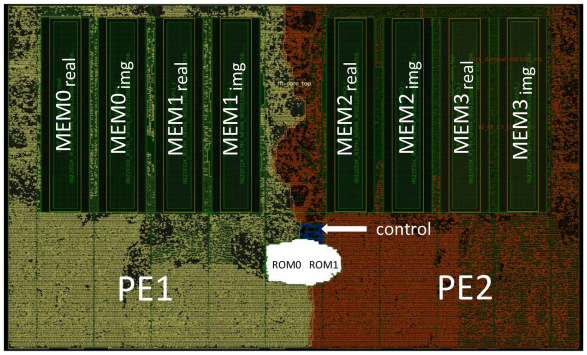


Figure 6. Floorplanning of $FFT^{\phi}/IFFT^{\phi}$ processor. The chip area is 0.3 mm $\times$ 0.5 mm, the clock frequency is 167 MHz, and the power consumption is 12.6 mW.

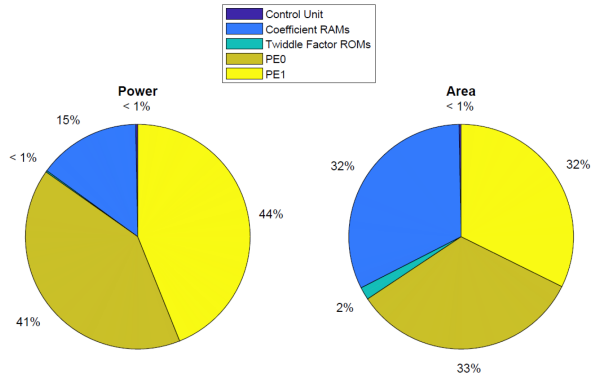


Figure 7. Area and power breakdown of the $FFT^{\phi}/IFFT^{\phi}$ processor (Total Area: 0.15 mm^2, Total Power: 12.6 mW) .

target frequency during the physical design of the chip (synthesis and place and route). This results in a higher chip frequency at the expense of increased power consumption, as we will see later. The subsequent results pertain to the non-pipelined version of our processor unless stated otherwise.

A. ASIC Implementation

In the logic design stage of our processor chip, the processor is implemented using Verilog. The desired functionality of the processor is simulated using Modelsim and comprehensively tested with test vectors generated by the FALCON reference software. To keep our area and power as low as possible, the twiddle factor ROMs are implemented using look-up tables (LUTs). Our design uses hard IP blocks for the coefficient memory banks. Eight single-port 128 $\times$ 64bit SRAM banks with one read/write port were compiled by Synopsys Memory Compilers. These compilers generate the IP blocks and all the physical information needed for the physical design stage. We synthesized the $FFT^{\phi}/IFFT^{\phi}$ processor using Synopsys Design Compiler with GF FD-SOI 22 nm standard cell library. Based on the synthesis report, our design cell area is equivalent to 417 k 2-input NAND with the lowest drive strength.

In the physical design stage, Synopsys IC Compiler II is used to perform floorplan, clock tree synthesis optimization, placement and routing optimization, and chip finishing. We targeted 6 ns for the clock period for the non-pipelined design. The final results show that 1) the area is 0.3 mm $\times$ 0.5 mm, 2) the clock frequency achieves 167 MHz under a wide temperature range from -40 °C to 125 °C, and the critical path is located in the butterfly unit in PE0, and 3) under the corner (40 °C and 0.72 V) the power consumption is calculated to be 12.6 mW. On the other hand, the pipelined design maintains the same area while achieving a clock frequency of 500 MHz. However, it consumes 28.1 mW of power under similar conditions. Figure 6 shows the layout of the FFT processor with sub-block annotations.

1 Area and Power Evaluation: Figure 7 shows the area and power breakdown of the $FFT^{\phi}/IFFT^{\phi}$ processor. The area and power needed for the control unit and ROMs are very small compared to the other components, because of the optimizations we applied; mainly the twiddle factor compression and the conflict-free scheduling. In addition, the two PEs consume the majority of the power and area of the chip, 85% and 65%, respectively. This is attributed to the fact that these PEs are designed to support the double-precision FP.

B. Comparisons With Related Work

In this section, we evaluate our processor by comparing its performance to related work in the literature. First, we compare the performance of the proposed processor (both pipelined and non-pipelined versions) with a software implementation of the reference FFT^{ϕ}, which was executed on a Raspberry Pi 4 with a Cortex-A72 processor. We selected this processor for comparison as it was utilized in [15] to run the reference and optimized (referred to as neon-based) FALCON signature generation, with detailed reported results. Our comparison aims to demonstrate that the proposed co-processor, operating with power consumption as low as tens milliwatts, achieves comparable performance to the ARM Cortex-A72, a powerful processor with power consumption in the range of several watts.

Table II presents comparisons of the number of cycles and execution times for FFT_n^{ϕ} where n is a power of 2 ranging from 8 to 1024, executed on Cortex-A72 and the proposed processor. Despite the lower clock frequency of our proposed processor compared to the Cortex-A72 (1.8 GHz), the computation of our non-pipelined FFT_n^{ϕ} for all n is faster than the reference implementation on Cortex-A72. Additionally, our pipelined processor exhibits shorter execution times for all n compared to both the reference and neon-based implementations. The speed-up of our pipelined processor ranges from 2.78$\times$ to 3.77$\times$, and 1.42$\times$ to 1.74$\times$ for FFT_n^{ϕ} compared to the reference and neon-based software implementations on Cortex-A72, respectively.

Since this is the only work that custom implements $FFT^{\phi}/IFFT^{\phi}$ over a ring in hardware, we compare the performance of our processor with several classic FP-based FFT accelerators available in the literature. Table III presents the performance comparison of the proposed $FFT^{\phi}/IFFT^{\phi}$ processor with several state-of-the-art accelerators. In this table, different accelerators use different technologies, supply voltages, FFT sizes, and data widths. Hence, the direct comparison between these processors is not fair. For a fair comparison, We use normalized metrics, normalized area, and normalized power, similar to the ones proposed in [27], [34] that consider these

TABLE II
COMPARISON OF THE CYCLE COUNTS AND THE EXECUTION TIME FOR THE IMPLEMENTATION OF FFT_n^{ϕ} WITH DIFFERENT NUMBERS OF COEFFICIENTS (n) ON CORTEX-A72 AND THE PROPOSED PROCESSOR

	Proposed Hardware Implementation				**FALCON Software Implementation [15]**			
Platform	ASIC				ARM® Cortex®-A72			
Implementation	Pipelined		Non-Pipelined		Neon		Reference	
Clock frequency	500 MHz		167 MHz		1.8GHz			
FFT size	Cycles	Excution Time [ns]	Cycles	Excution Time [ns]	Cycles	Excution Time [ns]	Cycles	Excution Time [ns]
8	6	**12**	4	24	54	30	100	55.6
16	14	**28**	12	72	92	51.1	232	128.9
32	34	**68**	32	192	221	122.8	516	286.7
64	82	**164**	80	480	540	300	1,132	628.9
128	194	**388**	192	1152	1155	641.7	2,529	1405
256	450	**900**	448	2688	2770	1538.9	5,474	3041.1
512	1026	**2052**	1,024	6144	5951	3306.1	11,807	6559.4
1024	2306	**4612**	2,304	13824	14060	7811.1	27,366	15203.3

TABLE III
COMPARING THE PERFORMANCE OF THE PROPOSED $FFT^{\phi}/IFFT^{\phi}$ PROCESSOR WITH STATE-OF-THE-ART CLASSIC FP-BASED FFT ACCELERATORS

	Proposed	Proposed (Pipelined)	[32]	[27]	[33]	[34]
Technology/Supply voltage	22 nm/ 0.72V	22 nm/ 0.72V	65nm/1.0V	45nm/0.9V	65nm/1.0V	45nm/1.08 V
FFT size	4-1024	4-1024	16-1024	4-1024	16-1024	32–2048
# Butterfly units	2 radix2	2 radix2	5 radix4	4 radix2	3 radix4, 2 half radix4	16 radix2
Data type	64-bit FP	64-bit FP	32-bit FP	32-bit FP	32-bit FP	32-bit FP
Area [mm^2]	**0.15**	**0.15**	1.003	2.4	0.736	0.973
Clock frequency	167 MHz	500MHz	500MHz	1 GHz	400MHz	100MHz
Power [mW]	**12.6**	28.1	43.5–372.3	91.3	35.9–129.5	68
Execution time (1024) [μs]	13.8	4.612	2.03	**1.38**	2.56	196
Run-time configurability	**FFT/IFFT, input size**	**FFT/IFFT, input size**	Input size	Input size	Input size	Input size
Normalized area [mm^2]	**0.146**	**0.146**	0.224	1.12	0.164	0.227
Normalized power [mW]	**12.3**	27.4	377	114.1	131.1	29.5
Normalized Energy [μJ]	170	**126**	765	157	335	5,782

variations. The formulae that calculate normalized area, normalized power, and normalized energy metrics with respect to our implementation are presented in (7), (8), and (9), respectively.

$$\hat{A} = \frac{A * 1000}{n * (\frac{L_m}{22})^2 * (\frac{W_L}{64})}, \tag{7}$$

$$\hat{P} = \frac{P * 1000}{n * (\frac{V}{0.72})^2 * (\frac{W_L}{64})}, \tag{8}$$

$$\hat{E} = \hat{P} * t_E, \tag{9}$$

where A and P are non-normalized values of the area and power, t_E is the execution time, n is the FFT size and we select it to be 1024, L_m is the minimum channel length of the technology in nm normalized to ours (22 nm), V is the supply voltage for each processor normalized to ours (0.72), and W_L is the word size which is 64 b (double-precision FP) in our case. The normalized area, power, and energy are listed in the last three rows of Table III. This table shows that our non-pipelined co-processor exhibits the lowest normalized area and power values, showcasing superior power and area efficiency compared to its counterparts. Conversely, the pipelined version of our co-processor displays the lowest normalized energy and the second-lowest normalized power. This indicates that our design offers optimal power and area efficiency and strikes the best balance between power efficiency and execution time for the non-pipelined and pipelined versions, respectively.

The slightly larger execution time of our processor reported in Table III compared to some of the other works is attributed to the fact that our processor has more capabilities represented by its ability to perform both FFT and IFFT over the ring. In addition, several optimizations have been adopted to reduce the power and the area such as; (i) using only 2 butterfly units (PEs) to reduce the area and power overhead, (ii) using single-port instead of dual-port SRAMs which reduces the power and the area of the coefficient memory almost by the half, (however, it requires 2 cycles to read from and write to the memories which duplicate the execution time), and (iii) using double-precision FP arithmetic which is a requirement for FALCON implementation. Note that both the frequency and the number of cycles of our processor can be further enhanced, albeit at the cost of increased power consumption. This improvement can be achieved by increasing the number of pipeline stages, expanding the number of PEs, and utilizing dual-port SRAMs. As far as we

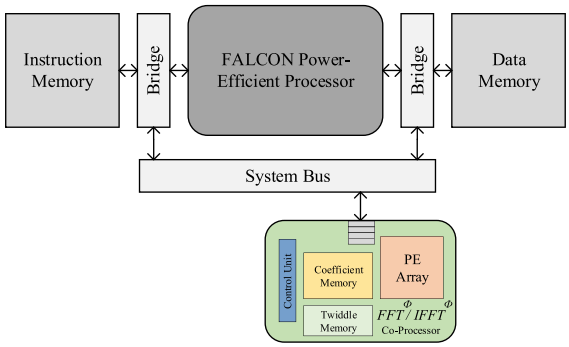


Figure 8. Loosely-coupled approach to connect FFT/IFFT accelerator to the main processor in the context of hardware-software co-design of FALCON.

know, the proposed processor is the only processor that performs $FFT^{\phi}/IFFT^{\phi}$. In addition, we provide a design that has the lowest reported area and power consumption in the literature, positioning it as the optimal choice for facilitating an efficient hardware implementation of FALCON.

C. Speedup Estimation in Context of FALCON

To give insights on the expected speedup that can be gained in the context of FALCON, we need to propose a specific scenario involving the integration of our co-processor with the full FALCON SoC. In this discussion, we consider using our co-processor in implementing a hardware-software co-design of FALCON. For this purpose, two primary options can be considered. The first involves adding custom ISA instructions to the system, and integrating the corresponding accelerators directly into the pipeline of the main processor that runs FALCON. This approach is commonly referred to as tightly-coupled accelerators, which is effective for lightweight operations. The second option is to implement a standalone accelerator connected to the system bus, known as loosely-coupled. This is suitable for large accelerators processing substantial data chunks [19], [20], [44], as in the case of our co-processor. In this discussion, we assume using the loosely-coupled as a potential approach to connect our FFT/IFFT accelerator to the main processor, as illustrated in Figure 8. As our co-processor offers a low-power and area-efficient solution for FTT/FFT of FALCON, it is expected to integrate seamlessly with power-efficient processors, such as ARM Cortex M4 and RISC-V, common choices for building cryptography systems [12], [19], [44], [45]. These processors are optimized for power efficiency and operate at frequencies comparable to our accelerator.

To account for communication overhead, we assume the use of a wide system bus (256 bits) to enable simultaneous reception of two coefficients (complex numbers). With our accelerator capable of starting processing with just four coefficients (assuming $n_{PE} = 2$), a four-cycle latency is adequate for communicating and buffering the first four coefficients, supplemented by two cycles to fill the pipeline (since we consider the case of a pipelined processor in this discussion). Subsequently, our co-processor, requiring two cycles for butterfly operations, ensures the timely availability of coefficients for the next round. The output of the FFT is then sent back to the main processor once every two output coefficients are calculated, adding four more cycles for the final four coefficients. Therefore, we have the flexibility to structure the Full FALCON SoC in a way that limits the communication and buffering overhead to 8 cycles for all values of n.

TABLE IV
COMPARISON OF FALCON SIGNATURE GENERATION (WITH $N = 1024$) EXECUTION TIME ON ARM CORTEX A72 WITH (ACCELERATED) AND WITHOUT (NON-ACCELERATED) OUR FFT/IFFT CO-PROCESSOR INTEGRATION

	Non-accelerated $[\mu s]$	Accelerated $[\mu s]$	Speedup $\times$
Reference Implementation	1713.78	924.8	1.85
Neon-based Implementation	1138.28	931.7	1.22

The presented speedup results, for both the reference and neon-based implementations [15], highlight the efficiency gained by connecting our accelerator

Despite our co-processor being intended for use with power-efficient processors, here we demonstrate that we still have a speedup even when connected to more powerful processors such as ARM Cortex A72. Again, We use this processor as a showcase to use the detailed reported results in [15][4] in this simplified analysis. We used the software implementation of FALCON signature generation to count the required FFT/IFFT operations of all sizes. Subsequently, we calculated the execution time needed for these operations using ARM Cortex A72 and one of the pipelined versions of our co-processor, including communication and pipelining overhead, to determine the reduced time due to using our accelerator. The expected execution time of FALCON signature generation on ARM Cortex A72 with (Accelerated) and without (Non-accelerated) our FFT/IFFT co-processor integration are presented in Table IV. Assuming the worst-case scenario where the main processor is stalled while the co-processor is calculating, we achieved speedups of 1.85 and 1.22 for the reference and neon-based implementations of FALCON signature generation, respectively. Since the neon-based implementation mainly optimizes the FFT implementation, a lower speedup in case of the neon implementation (only 1.22) is reported in Table IV.

We emphasize that our processor is designed for use with more power-efficient processors, which typically have much lower performance than ARM Cortex A72. Consequently, a more significant speedup for FALCON is expected due to the substantial reduction in clock cycles offered by our processor.

VI. CONCLUSION

Considering that FFT/IFFT is the most time and power-consuming operation for FALCON PQC implementation, an optimized $FFT^{\phi}/IFFT^{\phi}$ processor customized for efficient FALCON implementation is presented in this paper. The proposed processor is reconfigurable at run-time to adapt to different FFT sizes (4 to 1024 points) and can perform either FFT^{ϕ} or $IFFT^{\phi}$ operations. Several optimization techniques are proposed in this paper, such as twiddle factor compression and

[4]The reported results are for FALCON signature generation (which requires FFT) and verification(Witch doesn't require FFT).

conflict-free scheduling to save the power and area of the proposed processor. We have introduced two versions of our processor: one optimized primarily for power and area, while the other (pipelined) strikes a balance between power and execution time. The ASIC implementation of the proposed $FFT^{\phi}/IFFT^{\phi}$ processor shows excellent performance in terms of power, area, and execution time. It achieved a speed-up compared to a software implementation of FALCON on a Cortex-A72 processor and outperformed other FP FFT accelerators in terms of power and area efficiency. This design, being the most area and energy-efficient for $FFT^{\phi}/IFFT^{\phi}$, is highly suitable for enabling efficient FALCON implementation. The presented insights on the expected speedup when this co-processor is integrated into a full implementation of FALCON PQC on Cortex A72 shows a 1.85x speedup compared to the reference implementation. In our future work, we will conduct further investigation into the tradeoff between speedup and power and area efficiency when varying the number of processing elements. Additionally, we intend to prototype this co-processor on FPGA and utilize it as a building block for a complete FALCON implementation. Furthermore, we plan to evaluate and fortify our design against diverse side-channel attacks.

References

[1] V. Chamola, A. Jolfaei, V. Chanana, P. Parashari, and V. Hassija, "Information security in the post-quantum era for 5G and beyond networks: Threats to existing cryptography, and post-quantum cryptography," *Comput. Commun.*, vol. 176, pp. 99–118, 2021.

[2] A. K. Malviya, N. Tiwari, and M. Chawla, "Quantum cryptanalytic attacks of symmetric ciphers: A review," *Comput. Elect. Eng.*, vol. 101, 2022, Art. no. 108122.

[3] A. Deshpande, "Assessing the quantum-computing landscape," *Commun. ACM*, vol. 65, no. 10, pp. 57–65, 2022.

[4] D. J. Bernstein and T. Lange, "Post-quantum cryptography," *Nature*, vol. 549, no. 7671, pp. 188–194, 2017.

[5] D. Joseph et al., "Transitioning organizations to post-quantum cryptography," *Nature*, vol. 605, no. 7909, pp. 237–243, 2022.

[6] P. He, Y. Tu, Ç. K. Koç, and J. Xie, "Hardware-implemented lightweight accelerator for large integer polynomial multiplication," *IEEE Comput. Archit. Lett.*, vol. 22, no. 1, pp. 57–60, Jan./Jun. 2023.

[7] M. Imran, A. Aikata, S. S. Roy, and S. Pagliarini, "High-speed design of post quantum cryptography with optimized hashing and multiplication," *IEEE Trans. Circuits Syst. II: Exp. Briefs*, vol. 71, no. 2, pp. 847–851, Feb. 2024.

[8] A. C. Canto, A. Sarker, J. Kaur, M. M. Kermani, and R. Azarderakhsh, "Error detection schemes assessed on FPGA for multipliers in lattice-based key encapsulation mechanisms in post-quantum cryptography," *IEEE Trans. Emerg. Topics Comput.*, vol. 11, no. 3, pp. 791–797, Third Quarter 2023.

[9] B. J. Lucas et al., "Lightweight hardware implementation of binary ring-LWE PQC accelerator," *IEEE Comput. Archit. Lett.*, vol. 21, no. 1, pp. 17–20, Jun. 2022.

[10] K. Shahbazi and S.-B. Ko, "An optimized hardware implementation of modular multiplication of binary ring LWE," *IEEE Trans. Emerg. Topics Comput.*, vol. 11, no. 3, pp. 817–821, Third Quarter 2023.

[11] P.-A. Fouque et al., "FALCON: Fast-fourier lattice-based compact signatures over NTRU," *Submission NIST's Post-Quantum Cryptogr. Standardization Process*, vol. 36, no. 5, pp. 1–75, 2018.

[12] G. Alagic et al., "Status report on the third round of the NIST post-quantum cryptography standardization process," NIST Interagency/Internal Report (NISTIR), National Institute of Standards and Technology, Gaithersburg, MD, USA, 2022. Accessed: Jun. 4, 2024. [Online]. Available: https://doi.org/10.6028/NIST.IR.8413, https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=934458

[13] Y. Kim, J. Song, and S. C. Seo, "Accelerating FALCON on ARMv8," *IEEE Access*, vol. 10, pp. 44446–44460, 2022.

[14] V. Lyubashevsky, D. Micciancio, C. Peikert, and A. Rosen, "SWIFFT: A modest proposal for FFT hashing," in *Proc. Fast Softw. Encryption: Int. Workshop*, Springer, 2008, pp. 54–72.

[15] D. T. Nguyen and K. Gaj, "Fast FALCON signature generation and verification using ARMv8 NEON instructions," in *Proc. Int. Conf. Cryptol. Afr.*, Springer, 2023, pp. 417–441.

[16] T. Oder, J. Speith, K. Höltgen, and T. Güneysu, "Towards practical microcontroller implementation of the signature scheme FALCON," in *Proc. Post-Quantum Cryptogr.: Int. Conf.*, Springer, 2019, pp. 65–80.

[17] L. Beckwith, D. T. Nguyen, and K. Gaj, "Hardware accelerators for digital signature algorithms dilithium and FALCON," *IEEE Des. Test*, early access, Aug. 14, 2023, doi: 10.1109/MDAT.2023.3305156.

[18] D. Soni, K. Basu, M. Nabeel, N. Aaraj, M. Manzano, and R. Karri, *Hardware Architectures for Post-Quantum Digital Signature Schemes*. Berlin, Germany: Springer, 2021.

[19] P. Karl, J. Schupp, T. Fritzmann, and G. Sigl, "Post-quantum signatures on RISC-V with hardware acceleration," *ACM Trans. Embedded Comput. Syst.*, vol. 23, no. 2, pp. 1–23, 2022.

[20] E. Karabulut and A. Aysu, "A hardware-software co-design for the discrete gaussian sampling of falcon digital signature," *Cryptol. ePrint Arch.*, vol. 2023, 2023, Art. no. 908.

[21] M. Garrido, "A survey on pipelined FFT hardware architectures," *J. Signal Process. Syst.*, vol. 94, no. 11, pp. 1345–1364, 2022.

[22] L. Bertaccini, L. Benini, and F. Conti, "To buffer, or not to buffer? a case study on FFT accelerators for ultra-low-power multicore clusters," in *Proc. Int. Conf. Appl.-Specific Syst., Architectures Processors*, 2021.

[23] H. Cilasun et al., "CRAFFT: High-resolution FFT accelerator in spintronic computational RAM," in *Proc. ACM/IEEE Des. Automat. Conf.*, 2020.

[24] W. Di et al., "Multi-dimensional decomposition algorithm and hardware implementation for ultra-long point FFT," in *Proc. Int. Conf. Electron., Circuits Inf. Eng.*, 2021, 2021, pp. 300–305.

[25] Y.-C. Lee, T.-S. Chi, and C.-H. Yang, "A 2.17-mw acoustic DSP processor with CNN-FFT accelerators for intelligent hearing assistive devices," *IEEE J. Solid-State Circuits*, vol. 55, no. 8, pp. 2247–2258, Aug. 2020.

[26] S. Liu and D. Liu, "A high-flexible low-latency memory-based FFT processor for 4G, WLAN, and future 5G," *IEEE Trans. Very Large Scale Integration Syst.*, vol. 27, no. 3, pp. 511–523, Mar. 2019.

[27] X. Chen, Y. Lei, Z. Lu, and S. Chen, "A variable-size FFT hardware accelerator based on matrix transposition," *IEEE Trans. Very Large Scale Integration Syst.*, vol. 26, no. 10, pp. 1953–1966, Oct. 2018.

[28] A. Wang et al., "A 0.37 mm 2 LTE/Wi-Fi compatible, memory-based, runtime-reconfigurable 2 n 3 m 5 k FFT accelerator integrated with a RISC-V core in 16 nm FinFET," in *Proc. Asian Solid-State Circuits Conf.*, 2017, pp. 305–308.

[29] K.-F. Xia, B. Wu, T. Xiong, and T.-C. Ye, "A memory-based FFT processor design with generalized efficient conflict-free address schemes," *IEEE Trans. Very Large Scale Integration Syst.*, vol. 25, no. 6, pp. 1919–1929, Jun. 2017.

[30] C. J. Li, X. Li, B. Lou, C. T. Jin, D. Boland, and P. H. Leong, "Fixed-point FPGA implementation of the FFT accumulation method for real-time cyclostationary analysis," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 16, no. 3, pp. 1–28, 2022.

[31] H. H. Saleh, B. S. Mohammad, and M. Maalouf, "A high-throughput, contention-free low-power, radix-2 1 k, 2 k, 4 k and 8k-point fast fourier transform engine using 28 nm standard-cell process," in *Proc. Int. Conf. Des. Technol. Integr. Syst. Nanoscale Era*, 2013, pp. 184–185.

[32] M. Wang, F. Wang, S. Wei, and Z. Li, "A pipelined area-efficient and high-speed reconfigurable processor for floating-point FFT/IFFT and DCT/IDCT computations," *Microelectronics J.*, vol. 47, pp. 19–30, 2016.

[33] M. Wang and Z. Li, "An area-and energy-efficient hybrid architecture for floating-point FFT computations," *Microprocessors Microsystems*, vol. 65, pp. 14–22, 2019.

[34] A. S. Beulet Paul, S. Raju, and R. Janakiraman, "Low power reconfigurable FP-FFT core with an array of folded DA butterflies," *EURASIP J. Adv. Signal Process.*, vol. 2014, no. 1, pp. 1–17, 2014.

[35] P. Duong-Ngoc, S. Kwon, D. Yoo, and H. Lee, "Area-efficient number theoretic transform architecture for homomorphic encryption," *IEEE Trans. Circuits Syst. I: Reg. Papers*, vol. 70, no. 3, pp. 1270–1283, Mar. 2023.

[36] X. Feng and S. Li, "Accelerating an FHE integer multiplier using negative wrapped convolution and ping-pong FFT," *IEEE Trans. Circuits Syst. II: Exp. Briefs*, vol. 66, no. 1, pp. 121–125, Jan. 2019.

[37] J. Mu et al., "Scalable and conflict-free NTT hardware accelerator design: Methodology, proof and implementation," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 42, no. 5, pp. 1504–1517, May 2023.

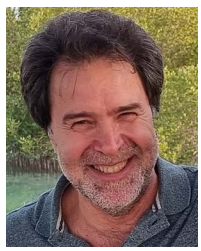
[38] D.-e.-S. Kundi, Y. Zhang, C. Wang, A. Khalid, M. O'Neill, and W. Liu, "Ultra high-speed polynomial multiplications for lattice-based cryptography on FPGAs," *IEEE Trans. Emerg. Topics Comput.*, vol. 10, no. 4, pp. 1993–2005, Fourth Quarter 2022.
[39] N. Zhang et al., "NTTU: An area-efficient low-power NTT-uncoupled architecture for NTT-based multiplication," *IEEE Trans. Comput.*, vol. 69, no. 4, pp. 520–533, Apr. 2019.
[40] P. Longa and M. Naehrig, "Speeding up the number theoretic transform for faster ideal lattice-based cryptography," in *Proc. Cryptol. Netw. Secur.: Int. Conf. Proc.*, Springer, 2016, pp. 124–139.
[41] N. Zhang, B. Yang, C. Chen, S. Yin, S. Wei, and L. Liu, "Highly efficient architecture of NewHope-NIST on FPGA using low-complexity NTT/INTT," *IACR Trans. Cryptogr. Hardware Embedded Syst.*, vol. 2020, pp. 49–72, 2020.
[42] FALCON: Fast-fourier lattice-based compact signatures over NTRU, 2020. [Online]. Available: https://falcon-sign.info/
[43] Z. Liu et al., "High-performance ideal lattice-based cryptography on 8-bit AVR microcontrollers," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 4, pp. 1–24, 2017.
[44] T. Fritzmann et al., "Masked accelerators and instruction set extensions for post-quantum cryptography," *IACR Trans. Cryptographic Hardware Embedded Syst.*, vol. 2022, no. 1, pp. 414–460, 2021.
[45] T. Fritzmann, G. Sigl, and J. Sepúlveda, "RISQ-V: Tightly coupled RISC-V accelerators for post-quantum cryptography," *IACR Trans. Cryptographic Hardware Embedded Syst.*, vol. 2020, pp. 239–280, 2020.

Ghada Alsuhli received the BS and MS degrees in electronics and communication engineering from Damascus University, Syria, in 2009 and 2015, respectively, and the PhD degree in electronics and communication engineering at Cairo University, Egypt, in 2019. Her academic journey was enriched by roles at esteemed research centers including the National Research Center, The American University in Cairo, Egypt, and the Khalifa University SoC Center, UAE. Currently, she serves as a postdoctoral researcher with Khalifa University, leading several projects focused on efficient hardware implementation for AI and post-quantum cryptography. Her research encompasses embedded systems, energy-efficient IoT solutions, edge computing, efficient hardware implementation, and AI applications for wireless communications and biomedical engineering. She is the primary author of numerous papers in reputable journals and conferences, she has also authored a book on efficient DNN hardware implementation. Her contributions extend to reviewing for esteemed journals and committee memberships for international conferences.

Hani Saleh (Senior Member, IEEE) received the PhD degree in computer engineering from the University of Texas at Austin. He is an associate professor of ECE with Khalifa University since 2017, he joined Khalifa on 2012. He is a cofounder of Khalifa University Research Center 2012–2018, and the System on Chip Research Center (SoC 2019-present). He has a total of 19 years of industrial experience in ASIC chip design, Microprocessor/Microcontroller design, DSP core design, Graphics core design and embedded systems design. Prior to joining Khalifa University he worked for many leading semiconductor design companies including Apple, Intel, AMD, Qualcomm, Synopsys, Fujitsu and Motorola Australia. Hani has published more than 48 journal papers, more than 119 conference papers, more than 6 books and 7 book chapters. His research areas includes but not limited to: AI accelerator design, digital ASIC design, digital eesign, computer architecture and computer arithmetic.

Mahmoud Al-Qutayri (Senior Member, IEEE) is a professor of electrical and computer engineering with Khalifa University (KU), UAE. He is also affiliated with KU System-on-Chip Center. Prior to joining Khalifa University, he worked with De Montfort University, U.K., and the University of Bath, U.K. He also worked with Philips Semiconductors, Southampton, U.K. He has authored or coauthored numerous technical papers in peer-reviewed journals and international conferences. His current research interests include embedded systems, in-memory computing and emerging memory technologies, energy-efficient IoT systems, efficient edge computing and artificial intelligence hardware implementation, application of AI to wireless communication systems, wireless sensor networks, and cognitive wireless networks. He received a number of awards during his education and professional career. His professional services include serving on the editorial board of some journals as well as membership of the steering, organizing, and technical program committees of a number of international conferences.

Baker Mohammad (Senior Member, IEEE) received the BS degree in electrical and computer cngineering from the University of New Mexico, Albuquerque, the MS degree in electrical and computer cngineering from Arizona State University, Tempe, and the PhD degree in electrical and computer engineering (ECE) from the University of Texas at Austin. He is a distinguished lecturer of IEEE CAS. He is currently a professor of electrical engineering and computer science (EECS) with Khalifa University and is the director of the SOCL. Before joining Khalifa University, he worked for 16 years in the US industry (Qualcomm and Intel), designing low-power and high-performance processors. His research interests include VLSI, power-efficient computing, high-yield embedded memory, and emerging technologies such as Memristor, STTRAM, In-memory-computing, hardware accelerators and power management. He has authored or coauthored more than 200 referred journals and conference proceedings, more than three books, and more than 20 U.S. patents.



Thanos Stouraitis (Life Fellow, IEEE) received the PhD degree from the University of Florida. He is a professor of EECS with Khalifa University, UAE, and a professor emeritus with the University Patras. He has also served on the faculties of Ohio State University, University of Florida, New York University, and University of British Columbia. His current research interests include AI hardware systems, signal and image processing systems, computer arithmetic, and design and architecture of optimal digital systems with emphasis on cryptographic systems. He has authored about 200 technical papers, several books and book chapters, and holds patents on DSP processor design. He received the IEEE Circuits and Systems Society Guillemin-Cauer Award. He has served as editor or guest editor for numerous technical journals, as well as general chair and/or technical program committee chair for many international conferences. He was president (2012–2013) of IEEE Circuits and Systems Society.