IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 19 March 2024, accepted 1 April 2024, date of publication 11 April 2024, date of current version 29 April 2024.

*Digital Object Identifier 10.1109/ACCESS.2024.3387489*

RESEARCH ARTICLE

# An Efficient Hardware/Software Co-Design for FALCON on Low-End Embedded Systems

**YONGSEOK LEE[1,2], JONGHEE YOUN[3], KEVIN NAM[1,2], HEON HUI JUNG[1,2], MYUNGHYUN CHO[4], JIMYUNG NA[4], JONG-YEON PARK[4], SEUNGSU JEON[4], BO GYEONG KANG[4], HYUNYOUNG OH[5], AND YUNHEUNG PAEK[1,2], (Member, IEEE)**

[1]Department of Electrical and Computer Engineering, Seoul National University, Seoul 08826, South Korea
[2]Inter-University Semiconductor Research Center, Seoul National University, Seoul 08826, South Korea
[3]Department of Computer Engineering, Yeungnam University, Gyeongsan-si 38541, South Korea
[4]Samsung Electronics System LSI, Hwaseong-si, Gyeonggi-do 16677, South Korea
[5]Department of AI·Software, Gachon University, Seongnam-si, Gyeonggi-do 13120, South Korea

Corresponding authors: Hyunyoung Oh (hyoh@gachon.ac.kr) and Yunheung Paek (ypaek@snu.ac.kr)

This work was supported in part by Samsung Electronics Company Ltd., under Grant IO201208-07839-01; in part by the BK21 FOUR Program of the Education and Research Program for Future ICT Pioneers, Seoul National University, in 2023; in part by the Inter-University Semiconductor Research Center (ISRC); in part by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by Korean Government (MSIT) (Development of open edge AI SoC hardware and software platform) under Grant RS-2023-00277060; in part by IITP under the Artificial Intelligence Semiconductor Support Program to Nurture the Best Talents grant funded by Korean Government (MSIT) under Grant IITP-2023-RS-2023-00256081; in part by the National Research Foundation of Korea (NRF) grant funded by Korean Government (MSIT) under Grant RS-2022-00166529 and Grant RS-2023-00277326.

**ABSTRACT** We propose in this paper an efficient FALCON accelerator called EFX based on a HW/SW co-design where FALCON is a post-quantum cryptographic (PQC) scheme tailored as a digital signature algorithm (DSA). Our findings reveal that FALCON exhibits unique characteristics and structures which distinguish it from other PQC-DSAs. A key finding is that, unlike its counterparts, FALCON doesn't prioritize a single, time-consuming task; instead, it processes a variety of tasks with comparable execution times. Consequently, the conventional methods focusing on accelerating dominant few tasks, which are generally effective for other algorithms, prove less efficient for FALCON, especially concerning the minimization of the silicon area used. To overcome this, we strategically focus on the granular optimization of lower-level operations rather than on broader functional segments, aiming to boost performance while conserving hardware space. Moreover, to mitigate the potential degradation due to limitation of hardware resources, we have implemented a pipelined execution strategy for the FALCON functions and refined the *sampling function*–a critical task that is challenging to accelerate due to inherent sequential algorithm–enabling it to run concurrently on both software and hardware, thus reducing latency. Our hardware design, synthesized at 300$MHz$ using Samsung's 28$nm$ and 45$nm$ process technologies, demonstrates superior performance in generating FALCON signatures, with a 3.58× improvement in clock cycles over an existing hardware accelerator. EFX occupies 38K $um^2$ and 74K $um^2$ for 28$nm$ and 45$nm$ processes, respectively, comparatively small compared to other PQC accelerators.

**INDEX TERMS** Post quantum cryptography, digital signature algorithm, cryptography, SW/HW co-design, FALCON, accelerator.

## I. INTRODUCTION

Low-end embedded systems play an important role in emerging ubiquitous systems such as the Internet of Things (IoT) and automatized industrial systems, where numerous low-cost devices need to be connected in a real-time manner while being scattered all across systems, communicating data to and from each other constantly. To keep such systems stable in terms of security, the devices should be able to provide authentication and integrity while being connected to each other. The

The associate editor coordinating the review of this manuscript and approving it for publication was Sedat Akleylek.

challenge for this goal is to find security measurements that are scalable for large systems, with the capability of being implemented efficiently to resource-constrained devices. These devices, however, are usually already overwhelmed with the computation itself. Digital Signature Algorithms (DSA) have been a great fit for the above requirements. The public-key cryptosystems (e.g., RSA and ECC) were both secure and lightweight enough to provide security for the low-end devices in a real-time manner [1]. Until the current classical computing era, they were widely adopted for various applications.

However, it has been shown that classical DSA algorithms can be easily broken using quantum computing [2], a prominent emerging field that has been driving significant development, where lots of research has been conducted to design new algorithms that are resistant to quantum computing, called Post-Quantum Cryptography (PQC). Since 2016, a standardization process was conducted by NIST, and three candidates (SPHINCS+ [3], FALCON [4], and CRYSTALS-Dilithium [5]) were selected as potential PQC-DSA standards in 2022 [6]. All three offer a high degree of security against attacks by both classical and quantum computers, making them a viable option for secure communication in the future.

PQC-DSA algorithms generally require more storage space and computation than classical-DSA, leading to difficulties in being implemented efficiently in real-time systems with low-end devices. To reduce the overhead caused by the increased computation, several dedicated hardware (HW) implementation studies have been conducted [7], [8]. For instance, previous studies on CRYSTALS-Dilithium and SPHINCS+ have mainly focused on improving performance through parallelism of operations using HW design methods [9], [10], [11], [12], [13], [14], [15], [16]. In CRYSTALS-Dilithium, 63% of the time during signature generation is consumed by NTT/INTT operations, and efforts are focused on accelerating these operations through highly parallel methods. Similarly, in SPHINCS+, 94% of the time during signature generation is consumed by the hash function execution, and efforts are focused on designing accelerators that utilize the internal slice parallelism of the hash function. As a result, much of the research for PQC-DSA algorithms has aimed to maximize overall performance by focusing on accelerating a few dominantly large functions that consume most of the time during the signature generation process.

Unlike other algorithms, FALCON is made up of multiple middling functions that have similar execution times. In our preliminary analysis, FFT/IFFT, SamplerZ and Split/Merge/LDL functions account, respectively, for 26%, 24% and 28% of the total execution time. Thus, it is clear that accelerating solely one function while neglecting the others would not achieve a significant performance gain through HW acceleration in the signature generation. Conclusively, to fully utilize HW acceleration and enhance performance, it appears essential to implement nearly all FALCON functions in HW. However, this straightforward acceleration method may lead to a significant increase in the total HW area because virtually all FALCON functions must be implemented as separate HW modules. To make matters worse, unlike others, FALCON performs double-precision floating-point (FP) operations. This results in an increase in the area of each individual HW module, which together further adds to the overall size of the HW, unsuitable for low-end devices.

The objective of this paper is to design and develop an efficient FALCON accelerator (which we call EFX) that can attain superior performance while occupying a small area to fit the requirements of low-end devices. We adhere to the conventional approach of HW/SW co-design where both HW and SW components carry out the desired functions and collaborate in a joint-optimized manner. In this approach, functions are normally partitioned into two groups: those that can be efficiently accelerated by HW and those that cannot. For efficiency in terms of both performance and area cost, designers consider the amount of necessary HW resources as well as the potential speedup that can be achieved through HW acceleration. By carefully balancing these factors, they aim to create a HW accelerator that provides significant performance gains while keeping minimum costs. To this aim, designers usually only choose a minimal set of functions that consume a large portion of the execution time and exhibit a high degree of potential parallelism. However, FALCON does not have a dominantly time-consuming function. Hence, implementing nearly all FALCON functions in HW may be the only viable option to achieve dramatic improvement in the overall performance through HW acceleration despite the potentially high costs in terms of area and resource usage. We, therefore, conclude that the conventional co-design approach, which performs *function-level partitioning* to choose functions for HW implementations selectively, is not an efficient way to accelerate FALCON while achieving both high performance and low HW cost.

Based on our observations, we seek to find an alternative approach that can enable us to implement FALCON in HW in a way to achieve both high performance and low cost simultaneously. We first investigate operations within a function, where an operation refers to the smallest unit of computation, such as arithmetic or bit-wise operations, that are executed sequentially during the function's processing. Then, we reveal a common set of operation blocks that are repeatedly executed during the processing of each function. We named these operation blocks as *common operation blocks* (COBs). Finally, we perform *operation-level partitioning*, where each function was partitioned into COBs and non-COBs, with COBs being implemented in HW and non-COBs in SW. Our approach was applied to FALCON and we found that COBs accounted for a significant proportion of the overall execution time, estimated at 90%, indicating its dominance in FALCON. Through our approach, we achieved our goal of efficient acceleration by minimizing the HW area and maximizing the performance increase.

Additionally, we made the following three efforts to maximize the performance of our accelerator.

- **Common Computing Unit (CCU) Development**: We introduced a hardware module named CCU, equipped with multiple hardware units tasked with executing COB operations. This singular CCU integration within our accelerator was aimed at reducing hardware space while enabling concurrent various COB executions to boost performance.
- **Pipelined Architecture for COBs**: CCU was engineered with a highly pipelined architecture, intended to maximize the efficiency of executing COBs. To this aim, we meticulously analyzed data dependencies and resolved internal memory bottlenecks on a cycle-by-cycle basis.
- **Optimization of *SamplerZ* Function**: Our focus extended to refining the SamplerZ function, a critical component noted for its high volume of non-COB sequential operations relative to other functions. Recognizing the importance of hardware and software synergy, we facilitated the parallel execution of COB and non-COB pairs. This was achieved by ensuring these pairs were free of data dependencies, allowing for their simultaneous operation. This method leverages the strengths of our hardware-software co-design strategy, emphasizing efficient and collaborative function execution.

To demonstrate the efficiency of EFX, we implemented EFX using the Samsung 28*nm* and 45*nm* process technology, respectively, with 300*MHz* of clock frequency. Compared to the NIST reference implementation (*i.e.*, SW running over the low-end Cortex M4 ARM processor), EFX demonstrates superior performance in terms of clock cycles by a factor of 9.70×. EFX also outperforms the recent work on SW/HW co-design of FALCON [17] in terms of clock cycles by a factor of 3.58×. Regarding area efficiency, EFX occupied the utilization of HW area less than other PQC works.

## II. BACKGROUND

### A. LATTICE-BASED CRYPTOGRAPHY AND PQC

Lattice-based cryptography (LBC) is one of the most promising PQC categories owing to its efficiency and versatility. Aiming the four algorithms that NIST selected as potential candidates for the PQC standard, three (CRYSTALS-Kyber, CRYSTALS-Dilithium, and FALCON) belong to the LBC category. Most LBCs are based on the well-known *learning with errors* (LWE) [18] problem and the *short integer solution* (SIS) [19]. LWE involves the identification of a vector $s$ when a vector $b = As + e$ is given, where $e$ is sampled from an error distribution. SIS involves the identification of a nonzero vector $z \in R^n$ such that $Az = 0$. NTRU (N-th degree Truncated polynomial Ring Units) involves the identification of a polynomial short pair $(f, g) \in R$ when a polynomial $h = f/g \in R$ is given. FALCON uses SIS over NTRU lattices as its underlying algorithm.

### B. FALCON

FALCON [4] is a DSA (digital signature algorithm) that utilizes the Gentry-Peikert-Vaikuntanathan (GPV) framework to construct hash-and-sign LBC. FALCON relies on the class of NTRU lattices and uses a trapdoor sampler [20] that combines the quality of Klein's algorithm with the efficiency of Peikert's algorithm. FALCON exhibits the smallest sum of the public key and the signature size among the NIST PQC algorithms. Sikeridis [21] argued that FALCON is suitable for the web if FP (floating point) HW module is available at the server, and Bindel [22] demonstrated that it is suitable for secure vehicle-to-vehicle (V2V) communication due to its small basic safety message packet size. FALCON consists of three procedures–key generation, signature generation, and signature verification. When considering reusing a key multiple times, the frequency of performing signature generation and verification is relatively higher among these three procedures. Therefore, focusing on accelerating signature generation and verification can be the most effective. However, compared to signature verification, which has a relatively simple algorithm structure and has already been the subject of acceleration research by many researchers [23], [24], research on accelerating signature generation is relatively lacking. Thus, in this study, we focused on signature generation. Algorithm 1 presents the procedure for generating FALCON signatures. The message $m$ and a random salt value $r$ are hashed to obtain $c$ using the HashToPoint function, and then FFT functions are performed. Next, the ffSampling function, which will be explained further in the following subsection, is repeatedly executed, followed by performing IFFT functions. Finally, the result is encoded into the output string $s$ using the compress function. Table 1 depicts the two standard parameter sets of FALCON, which we target to accelerate.

**Algorithm 1** FALCON Signature Generation Algorithm

**Input:** message $m$, secret key $sk$, bound $\lfloor \beta^2 \rfloor$
**Output:** signature $sig$ of $m$

$B \leftarrow \begin{bmatrix} g & -f \\ G & -F \end{bmatrix}$
$\hat{B} \leftarrow \mathrm{FFT}(B)$
$G \leftarrow \hat{B} \times \hat{B}^*$
$r \leftarrow \{0,1\}^{320}$ *uniformly*
$c \leftarrow$ **HashToPoint**$(r||m, q, n)$
$t \leftarrow \left( -\frac{1}{q}\mathrm{FFT}(c) \odot \mathrm{FFT}(F),\ \frac{1}{q}\mathrm{FFT}(c) \odot \mathrm{FFT}(f) \right)$
**do**
  **do**
    $z \leftarrow$ **ffSampling_dyntree**$_n(t, G)$
    $s = (t - z)\hat{B}$
  **while** $||s||^2 > \lfloor \beta^2 \rfloor$
  $(s_1, s_2) \leftarrow$ **invFFT** $(s)$
  $s \leftarrow$ **Compress** $(s_2, 8 \cdot sbytelen - 328)$
**while** $(s = \perp)$
**return** $sig = (r, s)$

**TABLE 1. FALCON parameter sets.**

| Parameters | Target Security Level | Ring Degree $n$ | Modulus $q$ | Max Bound $\lfloor \beta^2 \rfloor$ | Signature Bytelength $sbytelen$ |
|---|---|---|---|---|---|
| FALCON-512 | 1 | 512 | 12,289 | 34,034,726 | 666 |
| FALCON-1024 | 5 | 1,024 | 12,289 | 70,265,242 | 1,280 |

**Algorithm 2** Fast Fourier Sampling (ffSampling)

**Input:** $\mathbf{t} = (t_0, t_1) \in \text{FFT}\ (\mathbb{Q}[x]/(x^n+1))^2$, Gram Matrix $G$
**Output:** $\mathbf{z} = (z_0, z_1) \in \text{FFT}\ (\mathbb{Z}[x]/(x^n+1))^2$

1: **if** n = 1 **then**
2: $\quad\sigma' \leftarrow$ Normalization($g_{00}$)
3: $\quad z_0 \leftarrow$ **SamplerZ**($t_0, \sigma'$)
4: $\quad z_1 \leftarrow$ **SamplerZ**($t_1, \sigma'$)
5: $\quad$ return z = ($z_0, z_1$)
6: **end if**
7: (L, D) $\leftarrow$**LDL***(G)
8: $l \leftarrow L$
9: $d_{00}, d_{01} \leftarrow$ **split**($D_{00}$)
10: $d_{10}, d_{11} \leftarrow$ **split**($D_{11}$)
11: $G_0 \leftarrow \begin{bmatrix} d_{00} & d_{01} \\ d_{01}^* & d_{00} \end{bmatrix}, G_1 \leftarrow \begin{bmatrix} d_{10} & d_{11} \\ d_{11}^* & d_{10} \end{bmatrix}$
12: $t_1 \leftarrow$ **split**($t_1$)
13: $z_1 \leftarrow$ **ffSampling_dyntree**$_{n/2}$ ($t_1, G_1$)
14: $z_1 \leftarrow$ **merge**($z_1$)
15: $t_0' \leftarrow t_0 + (t_1 - z_1) \odot l$
16: $t_0 \leftarrow$ **split**($t_0'$)
17: $z_0 \leftarrow$ **ffSampling_dyntree**$_{n/2}$ ($t_0, G_0$)
18: $z_0 \leftarrow$ **merge**($z_0$)
19: **return** z = ($z_0, z_1$)

### C. FAST FOURIER SAMPLING (FFSAMPLING)

Fast Fourier sampling (ffSampling) is a central procedure in FALCON signature generation. As the name suggests, ffSampling improves performance by splitting polynomials like Fast Fourier transform (FFT). The ffSampling function includes two *recursive* calls, and each recursive call performs ffSampling recursively in half dimensions. Algorithm 2 illustrates the sampling procedure [4], [25]. Beginning with the matrix $G$, LDL decomposition is performed to decompose into a lower triangular $L$ and a diagonal matrix $D$. The internal calculation of LDL decomposition involves complex FP operations.

### D. CORTEX-M4 ARM MICROCONTROLLER

Cortex-M4 ARM Microcontroller is a small yet powerful microcontroller developed by ARM, widely used in embedded IoT devices and industrial automation systems. Unlike several other microcontrollers as MSP430 which only offers short integer datatypes, Cortex-M4 includes a Floating-Point Arithmetic Unit, offering the fast and precise computation required for PQC-DSA. It also offers an extensive range of peripherals, making it suitable for data-demanding applications. Our HW/SW co-design targets a platform using Cortex-M4 for two major reasons. First, it is one of the HW platforms that NIST designated as the primary microcontroller to be used for PQC standard benchmarks. Accordingly, numerous prior works on PQC implementations of low-end HW considered the Cortex-M4 as the target platform or baseline [26]. The other reason is that we aim to provide a fair SW baseline to compare with our HW accelerator. Generally, running the SW baseline using a weak platform makes the effect of an accelerator more distinguished in terms of diverse performance metrics. We therefore sought for a high-performance chip among microcontrollers to be used as our SW baseline, to evaluate our accelerator conservatively. Cortex-M4 is regarded as a premium and high-performance chip among microcontrollers, equipped with comparatively powerful resources that satisfy our needs.

## III. DESIGN OVERVIEW OF EFX

This section describes the overall design of EFX. We first explain how we identify the target sub-functions to be implemented as HW part, while the rest will be computed as SW on Cortex-M4. We then explain how we map those parts efficiently as a dedicated HW.

**TABLE 2. Latency breakdown of FALCON signature generation. - Listing in order of the most execution time.**

| Function name | Exec. time (%) | COBs | | | | Non-COBs | |
|---|---|---|---|---|---|---|---|
| | | Op1 | Op2 | Op3 | Op4 | Op5 | Others |
| SamplerZ | 24.3 | ✓ | ✓ | ✓ | ✓ | ✓ | |
| FFT | 21.5 | | ✓ | ✓ | | | |
| poly_split | 12.0 | | ✓ | ✓ | | | |
| poly_LDL | 11.4 | | ✓ | ✓ | ✓ | | |
| complete_private | 5.3 | | | | | | ✓ |
| IFFT | 4.9 | | ✓ | ✓ | | | |
| poly_merge | 4.9 | | ✓ | ✓ | | | |
| poly_mul | 4.8 | | ✓ | ✓ | | | |
| poly_add/sub | 3.5 | | ✓ | | | | |
| smallints_to_fpr | 0.8 | ✓ | | | | | |
| poly_mulselfadj | 0.7 | | ✓ | ✓ | | | |
| poly_muladj | 0.7 | | ✓ | ✓ | | | |
| fpr_sqrt | 0.5 | | | | | | ✓ |
| fpr_rint | 0.5 | ✓ | | | | | |
| poly_mulconst | 0.2 | | | ✓ | | | |
| fpr_of | 0.1 | ✓ | | | | | |
| etc | 3.8 | | | | | | |

Op is the operation type. (Op1: Type Converter. Op2: FP Addition. Op3: FP Multiplication. Op4: FP Division. Op5: Sampling.)

### A. PROFILING FALCON SIGNATURE GENERATION

We performed a profiling of the FALCON signature algorithm on the SW baseline to identify the target operations to offload to our HW component. Table 2 shows the analysis results of running the algorithm on Cortex-M4. As described before, our partitioning is handled at *operation-level*, not at function-level, to increase efficiency. Specifically, note that SamplerZ and FFT account for 45.8% of the overall execution time, which is less than half. Nevertheless, we could target them to accelerate using HW, but this would necessitate the HW to provide 5 types of operations

(described as Op1, Op2, …, Op5 in the table), requiring a large area. Analyzing at an operation scale, we could find out that the computation of 4 operations (Op1:`Type_converter`, Op2:`FP_ADD`, Op3:`FP_MUL`, and Op4:`FP_DIV`) takes account for approximately 90% of the execution time. Therefore, offloading the 4 operations would lead to more efficiency than the former function-level approach, in terms of both HW area and acceleration. Note that these 4 operations are repeatedly executed across various functions. We hereafter note these 4 operation blocks as *common operation blocks* (COBs), which are mapped and implemented in HW, whereas non-COBs are run on the SW side.

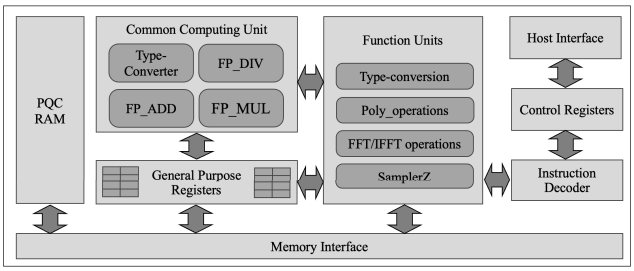



**FIGURE 1.** Overview of the HW part of EFX.

### B. ARCHITECTURE OVERVIEW OF EFX

Figure 1 depicts the overview of the HW part of EFX. Connected with the host processor via the Host Interface, the Control Signals are configured to initialize the HW. Then, the functions are offloaded from the host to the HW, decoded by the Instruction Decoder (ID), and then sent to the Function Unit (FU). FU plays the role of finite state machine, where each state represents a function that is computed by the HW. The micro-commands from FU are sent to the Common Computing Unit (CCU), General Purpose Registers (GPR), and also to Memory Interface (MI). The CCU contains COBs, and GPR is used to store the intermediate values. Once the HW finishes its task, it sends an interrupt signal to the host and waits for the next operations to be offloaded from the host.

### C. RESOURCE ALLOCATION FOR EFFICIENT DESIGN

To fulfill our goal of designing an efficient HW component, we used a single-port SRAM as PQC RAM (data buffer), which stores the inputs, outputs, and intermediate data during the computation. We use 64 bits as the bus width to compactly match our main data type, double-precision FP. Additionally, we implement registers that can each store the intermediate data during COB operations. To assess the minimum amount of registers to be used, we analyzed the cases where each COB operations within the CCU are performed simultaneously. As a result, we determined that we need to hold at least 12 64-bit values during the computation of the functions that are to be offloaded to our HW component – for which reason, we allocated 12 registers.

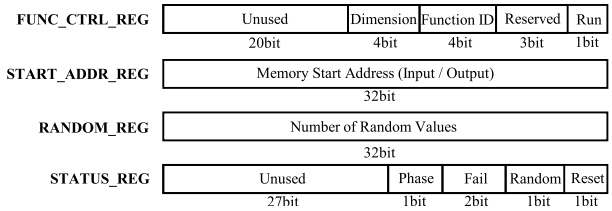



**FIGURE 2.** Control registers.

### D. HOST TO/FROM EFX INTERFACE

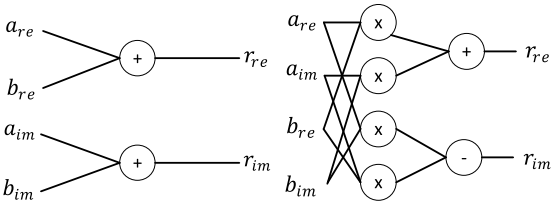
The interaction between the host CPU and our hardware accelerator, EFX, is designed for flexibility through the use of a control register via an AHB host interface. The configuration of this interface, as illustrated in Figure 2, allows for the setting of polynomial dimensions and the specific functions to be executed by EFX through the FUNC_CTRL_REG. This versatility enables the host to handle both FALCON-512 and FALCON-1024 parameters seamlessly with FUNC_CTRL_REG, as well as to facilitate key generation operations that mirror those of signature generations The memory address for input/output data is specified in START_ADDR_REG, ensuring proper data storage and retrieval. For the specialized coordination of `SamplerZ` function's execution between software and hardware, RANDOM_REG and STATUS_REG are employed. RAMDON_REG is tasked with holding the count of available random values, while STATUS_REG monitors the essential state for `SamplerZ` function's operation. Through STATUS_REG, the host can identify issues such as sampling errors and shortages of random numbers, which will be detailed later in section V.

## IV. COMMON COMPUTING UNIT (CCU)

The Common Computing Unit (CCU), as illustrated in Figure 1, is a HW component equipped with a series of common operation blocks (COBs) used for both FP operations and single data type operations such as type conversions. The 64-bit FP operations of COBs, which are integral to the generation of FALCON signatures and known for their extensive processing time, are accelerated through a specifically designed FP Unit (FPU). This FPU is engineered to function at a 300MHz frequency, optimizing the execution of COBs.

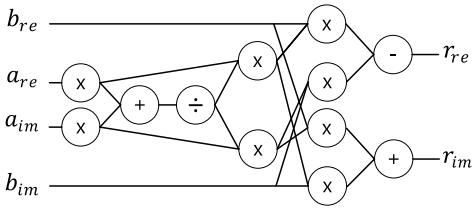
### A. FP ADDER, MULTIPLIER, AND DIVIDER

In our pursuit of designing an area-efficient hardware component for EFX, we opted to create separate modules for each FP operation after determining that a unified module handling addition, multiplication and division would significantly degrade performance. This decision was informed by the recognition that FP division is inherently more complex and involves many more processing stages than FP addition, as depicted in Figure 3. A combined module, operating in a pipelined manner, would force FP addition operations to unnecessarily traverse multiple stages, causing delays without justifiable cause. Moreover, FP division is utilized in only two




(a) Complex floating-point addition (FPC_ADD)

(b) Complex floating-point multiplication (FPC_MUL)




(c) Complex floating-point division (FPC_DIV)

**FIGURE 3.** Complex number calculations using floating-point operations.

specific functions within the FALCON algorithm, whereas addition and multiplication are fundamental to the majority of operations. Given these considerations, we resolved to design the three modules–FP addition, FP multiplication and FP division.–independently, this streaming the processes for addition and multiplication. Our design includes a 64-bit FP adder configured with two pipeline stages and a multiplier set up with five stages, employing a classical 4-digit schoolbook multiplication technique.

For the FP division module, a non-pipelined architecture was deemed most suitable due to its infrequent invocation during the FALCON algorithm's execution. We implemented a series of three 53-bit signed subtractors that met our desired frequency (i.e., 300MHz), reusing these across 55 iterations to complete the division process within 19 cycles. In a FP division operation, the subtraction of FP exponents and the XOR manipulation of the sign bit are performed in parallel. The final steps involve normalization, clamping and rounding, which are executed over two cycles to ensure precision and efficiency.

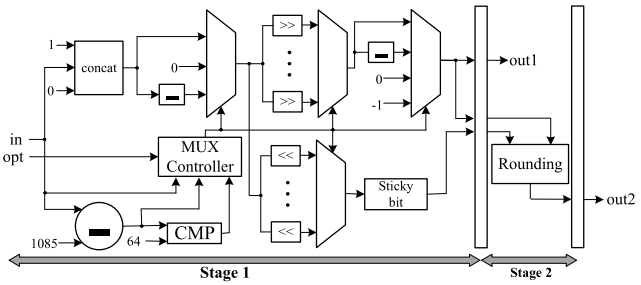



**FIGURE 4.** Pipeline of the type-converter (FP to integer).

### B. TYPE-CONVERTER MODULE

The type-converter COB facilitates the transition between integer and FP data types. When converting integer to FP numbers,a two-stage pipeline architecture is employed. This structure first normalizes the data before rounding it to achieve the desired FP representation. Conversely, the conversion from FP numbers to integers involves processes such as truncation, rounding and flooring, all of which depend on shifts in the mantissa guided by FP exponents. Given the nature of these operations, our approach prioritizes HW efficiency over direct performance enhancement through pipelining. We achieve this by employing a single, to-be-reused component capable of handling all three stages of the FP-to-integer conversion process. An additional component is integrated exclusively for the rounding stage, which necessitates a 63-bit addition, extending the operation by an additional cycle. As depicted in Figure 4, while the rounding process is structured as a two-stage pipeline, both truncation and flooring are designed to be completed within a single cycle to adhere to clock speed constraints.

**TABLE 3.** List of the functions which are implemented in HW.

| ID | Function Name | Description |
|---|---|---|
| 0 | smallints_to_fpr | Type-conversion (8bit Integer to 64bit FP) |
| 1 | N_fpr_of | Type-conversion (64bit Integer to 64bit FP) |
| 2 | N_fpr_rint | Type-conversion (64bit FP to 64bit Integer) |
| 3 | poly_add | Polynomial addition |
| 4 | poly_sub | Polynomial subtraction |
| 5 | poly_mul | Polynomial multiplication |
| 6 | poly_muladj | Polynomial multiplication with the adjoint of the polynomial |
| 7 | poly_mulconst | Polynomial multiplication with a real constant |
| 8 | poly_mulselfadj | Polynomial multiplication with its own adjoint |
| 9 | poly_LDL | LDL Decomposition |
| 10 | FFT | Fast Fourier Transform |
| 11 | FFT_neg | Fast Fourier Transform with negative output |
| 12 | poly_merge | Merge two polynomials into one |
| 13 | IFFT | Inverse Fast Fourier Transform |
| 14 | poly_split | Split a polynomial to two polynomials |
| 15 | SamplerZ | Discrete Gaussian sampling |

## V. FUNCTION UNIT (FU)

The Function Unit (FU) within the hardware part of EFX is responsible for coordinating the execution of operations offloaded from the software. Upon receiving these operations, the FU allocates them across the hardware modules of EFX for processing. The scope of functions managed by the FU, as detailed in Table 3, inlcudes a range of tasks that leverage shared resources within the EFX hardware, including CCU. Key resources such as GPRs, which hold both input/output and intermediate data, along with finite-state machines (FSMs) for resource management, are utilized collectively among the hardware functions. This subsection describes the strategy employed by FU to orchestrate the module operations described in section IV, facilitating the execution of advanced operations like polynomial computations and FFT/IFFTs.

### A. POLYNOMIAL OPERATIONS

In the context of FALCON signature generation, operations on polynomials, with the exception of multiplications, are conducted in a coefficient-wise fashion. This approach allows for sequential computation without the need for addressing data dependencies or specific access patterns across polynomial coefficients' indices. Operations such as type conversion, addition and subtraction among polynomial

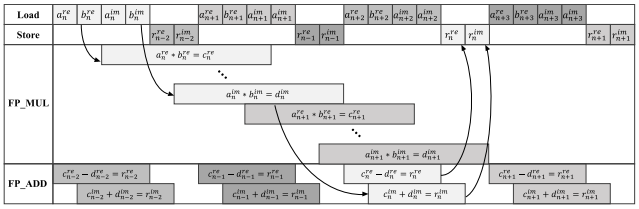
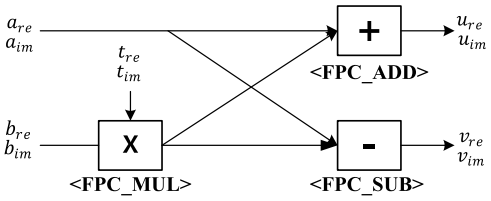



**FIGURE 5.** Scheduling of polynomial multiplication operation.

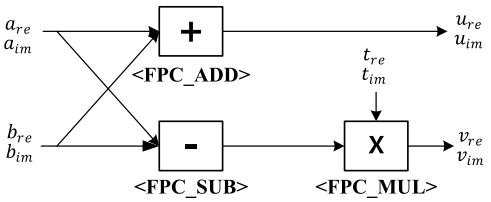
coefficients are thus scheduled sequentially by FU, resulting in each polynomial operation of degree $N$ averaging $N$ FP operations. It is critical to note, however, that the execution of such polynomial operations does not strictly translate to $N$ cycles, given the non-pipelined nature of our FP operation implementations which may introduce additional latency.

On the other hand, polynomial multiplication presents a more complex challenge due to the intricate data access patterns required, where every coefficient interacts with every other in a comprehensive manner. To address this complexity, we have implemented a specialized pipeline to efficiently manage FP operations for polynomial multiplications, as illustrated in Figure 5.

This setup allows for the sequential reception of four input data sets, with the `FP_MUL` operation processing these inputs in a pipelined fashion before forwarding the results to the `FP_ADD` operation, which similarly employs a pipelined computation process. During this computation, periods of inactivity are observed for both `FP_MUL` and `FP_ADD` COBs, alongside instances of idle memory bus time. To enhance optimization, we have strategized the overlapping of multiple input sets to ensure maximal utilization of COBs and the memory bus, effectively reducing idle times. The variation in data and operation colors within Figure 5 signifies the engagement of different input sets, depicting our approach to optimizing polynomial multiplication execution within the FALCON algorithm.




(a) Cooley-Tukey Algorithm




(b) Gentlman-Sande Algorithm

**FIGURE 6.** Butterfly operations in FFT/IFFT.

### B. FFT/IFFT OPERATIONS

The FALCON algorithm implementation submitted to NIST, the computation of Fast Fourier Transform (`FFT`) and its inverse (`IFFT`) employs the Cooley-Tukey method [27] and Gentleman-Sande method [28], respectively. Our hardware implementation adheres to these established methodologies for `FFT` and `IFFT`, with a focused optimization on their operational processes.

As shown in Figure 6, `FFT` and `IFFT` functions involve critical butterfly operations. To enhance efficiency and reduce latency, we have overlapped these butterfly operations across both functions, leveraging the pipelined architecture of CCU for optimal utilization. Similar to the `FFT`/`IFFT` operations, the execution of fast Fourier sampling operations, including the splitting and merging of polynomials, adopts a methodology paralleling the `FFT`/`IFFT`'s initial stage. The functions `poly_merge_fft` and `poly_split_fft` mimic the dataflow of `FFT` and `IFFT`'s butterfly operations, respectively, albeit being executed in a singular stage. The difference is the number of stages required to get the final result. The `poly_merge_fft` and `poly_split_fft` functions are performed only in a one-stage operation, but `FFT` and `IFFT` functions are performed in 9 or 10 stages according to FALCON-512 and FALCON-1024 parameters.

To efficiently integrate these operations, FU combines the stage operations of `FFT` and `IFFT` with those of `poly_merge_fft` and `poly_split_fft`. This integration is facilitated by a dedicated module designed for calculating memory addresses, effectively halving the value by multiplexing the output signals of FP_ADD. This approach enables the seamless unification of `poly_merge_fft` with `FFT` and `poly_split_fft` with `IFFT`, streamlining the process and optimizing performance.

**Algorithm 3** SamplerZ

**Input:** Floating point values $\mu$,
$\sigma' \in R$ such that $\sigma' \in [\sigma_{min}, \sigma_{max}]$
**Output:** An integer $z \in \mathbb{Z}$ sampled from a distribution very close to $D_{\mathbb{Z},\mu,\sigma'}$

$r \leftarrow \mu - \lfloor \mu \rfloor$
$ccs \leftarrow \sigma_{min}/\sigma'$
**while** *true* **do**
  $z_0 \leftarrow$ **GaussianSampler**() ▷ Gaussian sampling
  $b \leftarrow$ UniformBits(8) & 0x1
  $z \leftarrow b + (2 \cdot b - 1)z_0$
  $x \leftarrow \frac{(z-r)^2}{2\sigma'^2} - \frac{z_0^2}{2\sigma_{max}^2}$
  if(**BerExp**(x,ccs) = 1) ▷ Rejection sampling
    **return** z + $\lfloor \mu \rfloor$

### C. SAMPLERZ

`SamplerZ` is identified as the most time-consuming function in the signature generation process, accounting for 24.3% of

the total execution time, as detailed in the breakdown (see Table 2). This function is distinct from others in that it involves specialized operations for *Gaussian* and *rejection sampling*, as outlined in Algorithm 3. To enhance its efficiency, we have implemented a series of optimization techniques, as depicted in Figure7. As both *Gaussian sampling* and uniformly random generation can be executed independently of other operations, they are initially handled by the software and then dispatched their results to PQC RAM for simultaneous hardware usage (❶). Specifically, the outcomes of *Gaussian sampling* ($z_0$) and uniformly random generation ($b$) are pre-calculated and then fed into `SamplerZ` on HW part. Simultaneously, a dedicated buffer in the PQC RAM is allocated for random value storage, with STATUS_REG ensuring synchronization of these available random values ($z_0$, $b$) between the host and the HW accelerator.

Algorithm 3 shows the operation flow of the entire `SamplerZ` function, aiming to find a random integer $z$ derived from two random values, ensuring it passes a specified test (line 8), where $z_0$ is sampled from a discrete Gaussian distribution centered on $\mu$ with standard deviation $\sigma'$ and $b$ is a uniformly sampled bit. On the procedure, $\mu$ is divided into the integer part $\lfloor \mu \rfloor$ and the fractional remainder $r$. If passed the test calculated by the *rejection sampling* function (line 8), the final sampled result $z$ is made by adding the integer part $\lfloor \mu \rfloor$ (line 9). Given the iterative nature of finding $z$ in lines 3-9 in Algorithm 3, failures in the testing phase can extend latency. To mitigate the penalty due to this failure, we designed the HW part of `SamplerZ` to parallelize the evaluation of multiple $z_0$ and $b$ candidate pairs (❷).

To be specific, `SamplerZ` is processed somewhat *speculatively* in the sense that SW generates abundant random $z_0$ and $b$ pairs in idle time, assuming that the test would fail. Asynchronously to SW generation, HW part finds $z$ by multiple tests on these pre-generated pairs (❸). Our empirical analysis revealed that the `SamplerZ` function was invoked 102,400 times during signature generation process, with approximately 42.4% of operations failing more than once and an average number of failure equal to 0.74 (Table 4). By testing four candidate pairs in parallel based on this analysis, the likelihood of requiring more than one iteration drops to 1.31%, insignificantly affecting performance.

The integration of the Bernoulli approximate exponential function within *rejection sampling* test is accomplished using our FP multiplier (❹). The primitive operation of the Bernoulli approximation exponential function is 64-bit multiplications that typically demand substantial HW space. By assigning these multiplications to the shared COB, specifically the FP multiplier, we significantly reduce the hardware footprint. Upon completion of the sampling process, STATUS_REG is utilized to send an interrupt signal to the host (❺), prompting the SW to verify the process status. In the case of lacking random values on PQC RAM, as indicated by RANDOM_REG, HW part awaits SW's replenishment of random values.

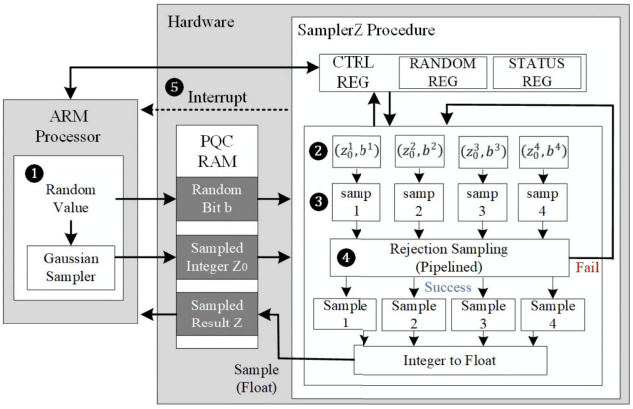



**FIGURE 7.** Sampler function optimization.

**TABLE 4.** SamplerZ execution failure statistics.

| # of failures | 0 | 1 | 2 | 3 | 4 | 5 | 6 ≥ | Total |
|---|---|---|---|---|---|---|---|---|
| # of SamplerZ executions | 58957 | 24978 | 10725 | 4465 | 1931 | 742 | 602 | 102400 |
| ratio (%) | 57.58 | 24.39 | 10.47 | 4.36 | 1.89 | 0.72 | 0.59 | 100.00 |

## VI. EXPERIMENTAL RESULTS

In this section, we evaluate the performance, HW area and power requirements of EFX. We compare EFX against the other papers to show the efficiency of our work. We assess the execution time using a cycle-accurate simulator and determine the required HW area by physically synthesizing the proposed design using Samsung's 28*nm* and 45*nm* process technology at 300 *MHz*. We implement SW baseline (SWB) of FALCON on the Cortex-M4 board (STM32F407VG) [29] that operates at a frequency of 168 MHz. This SW board setting is considered as the standard environment for the performance evaluation of PQC, recommended by NIST. To count the cycles and measure the time, we used Data Watchpoint and Trace (DWT) [30] which is implemented within ARM Cortex M series providing some additional debug information including clock cycle counts.

**TABLE 5.** Performance comparison of EFX with SWB [31].

| Function name | FALCON-512 | | | FALCON-1024 | | |
|---|---|---|---|---|---|---|
| | SWB (ms) | EFX (ms) | speed-up | SWB (ms) | EFX (ms) | speed-up |
| smallints_to_fpr | 3.3 | 0.005 | 702.3 | 6.6 | 0.009 | 707.7 |
| N_fpr_of | 0.4 | 0.001 | 406.8 | 0.8 | 0.002 | 407.7 |
| N_fpr_rint | 2.3 | 0.002 | 1093.7 | 4.5 | 0.004 | 1096.9 |
| FFT | 48.5 | 0.059 | 823.4 | 109.1 | 0.133 | 822.8 |
| FFT_neg | 39.8 | 0.047 | 843.9 | 89.2 | 0.106 | 841.1 |
| iFFT | 20.3 | 0.024 | 834.8 | 45.4 | 0.054 | 836.9 |
| poly_add | 7.8 | 0.026 | 305.8 | 16.8 | 0.054 | 309.0 |
| poly_sub | 6.5 | 0.018 | 363.4 | 14.4 | 0.039 | 370.7 |
| poly_mul | 19.9 | 0.032 | 613.8 | 42.3 | 0.068 | 625.5 |
| poly_muladj | 2.7 | 0.003 | 882.2 | 5.5 | 0.006 | 887.1 |
| poly_mulconst | 0.8 | 0.002 | 396.2 | 1.6 | 0.004 | 396.6 |
| poly_mulselfadj | 2.9 | 0.004 | 697.5 | 5.8 | 0.008 | 699.2 |
| poly_LDL | 46.9 | 0.093 | 503.3 | 104.1 | 0.200 | 521.5 |
| poly_merge | 20.3 | 0.032 | 631.4 | 45.4 | 0.070 | 647.9 |
| poly_split | 49.2 | 0.061 | 808.9 | 110.5 | 0.133 | 830.3 |
| SamplerZ | 100.0 | 0.102 | 978.2 | 195.6 | 0.212 | 923.1 |

**TABLE 6. Performance comparison with other papers in the signature generation.**

| Works | Parameters | | Clock Cycle | Relative Speedup | Clock Speed (MHz) | Time (ms) |
|---|---|---|---|---|---|---|
| M4(C-opt) [26] | FALCON-512 | Total(SW) | 62,225,400 | ×9.44 | 168 | 370.39 |
| | FALCON-1024 | Total(SW) | 136,596,407 | ×9.70 | 168 | 813.07 |
| M4(Asm-opt) [26] | FALCON-512 | Total(SW) | 38,090,446 | ×5.78 | 168 | 226.73 |
| | FALCON-1024 | Total(SW) | 83,482,883 | ×5.93 | 168 | 496.92 |
| ARM+FPGA [17] | FALCON-512 | - | - | - | - | - |
| | FALCON-1024 | Total(HW+SW) | 50,406,717 | ×3.58 | | 144.37 |
| | | HW | 10,074,112 | | 121 | 83.26 |
| | | SW | 40,332,605 | | 660 | 61.11 |
| **Ours (M4+ASIC)** | FALCON-512 | **Total(HW+SW)** | **6,590,698** | ×1.00 | | **37.82** |
| | | HW | 540,204 | | 300 | 1.80 |
| | | SW | 6,050,494 | | 168 | 36.01 |
| | FALCON-1024 | **Total(HW+SW)** | **14,078,075** | ×1.00 | | **80.86** |
| | | HW | 1,120,125 | | 300 | 3.73 |
| | | SW | 12,957,950 | | 168 | 77.13 |

### A. PERFORMANCE

The performance evaluation of our EFX, as compared to a purely software-based implementation (SWB) on a Cortex-M4 board, is presented in Table 5. The SWB represents the performance metrics of FALCON when executed entirely in software on a resource-constrained device, which lacks high precision arithmetic units, including the double-precision FPU essential for FALCON computations. To optimize performance within these constraints, emulated FP operations were executed using optimized ARM assembly code in the NIST submitted version. The EFX design achieved acceleration factors between 305.8 to 1093.7× for FALCON-512 and 309 to 1096.9× for FALCON-1024, in comparison to SWB. Further comparisons with prior studies on the signature generation function of the FALCON algorithm are detailed in Table 6. The study by Kannwischer et al. [26], which also targeted the Cortex-M4 board, proposed two versions of design, C optimization and assembly optimization. Our design significantly outperforms their results, reducing clock cycles by at least 5.78× for FALCON-512 and up to 9.7× for FALCON-1024. In terms of HW implementations, previous efforts [32], [33] have focused on accelerating specific functions of the FALCON algorithm without supporting the entire algorithm. The only accelerator (named by ARM+FPGA in the table) for the full signature generation function, by Karabulut and Aysu [17], are compared. Karabulut et al.'s study, which also employs a SW/HW co-design approach focusing on the SamplerZ function for FALCON-1024, demonstrates that our EFX design requires 3.58× fewer clock cycles in comparison.

### B. AREA AND POWER

Table 7 shows the comparison of HW area and power consumption with previous studies that implemented HW accelerator for PQC algorithms. To the best of our knowledge, there has been no previous research focused on the comprehensive hardware implementation of the FALCON signature generation, which the primary objective of our work. Given this context, making direct comparisons with prior studies is not straightforwad. Nontheless, we seek to indirectly assess the area and power efficiency of our EFX design by apposing it with other PQC research efforts that execute functions analogous to those performed by FALCON. As indicated in Table 7, our EFX design demonstrates a notably compact hardware footprint relative to other HW implementations in the field. To facilitate a fair evaluation of area and power consumption across varying silicon processes, we utilized both the 28*nm* and 45*nm* processes for synthesis. Our design was synthesized to occupy areas of 38K$\mu m^2$ and 74K $\mu m^2$ using Samsung 28*nm* and 45*nm* technology at 300 MHz, respectively. Our design strategy emphasized the efficient reuse of resources, such as CCU, GPRs, FU across multiple hardware-accelerated functions. Moreover, we integrated the Bernoulli approximate exponential function used in `SamplerZ` with a FP multiplier to further optimize resource utilization.

In terms of power consumption, our results indicate lower or comparable levels relative to those reported in existing DSA studies. Although there are differences in the algorithms and security levels supported between our work and that of Soni et al. [34], which also utilized the FALCON algorithm but focused on signature verification, our signature generation implementation demonstrates somewhat higher power use. This can be attributed to the greater computational complexity involved in signature generation as opposed to verification. Indeed, signature generation has been shown to require between 150× to 160× more clock cycles than verification on the Cortex M4 board [37], validating our design's power efficiency for such a complex task. While the lack of directly comparable studies on FALCON signature generation poses challenges for hardware comparisons, our investigation underscores the EFX design's advantages in terms of both

**TABLE 7. Area and power comparison of EFX with other papers.**

| Works | Categories | Algorithms | Security Level | Funtions | $nm$ | Freq. (MHz) | Area ($\mu m^2$) | Power ($mW$) HW | SW |
|---|---|---|---|---|---|---|---|---|---|
| Soni *et al.* [34] (HW) | DSA | qTesla | 1 | Signing | 65 | 200 | 840,169 | 6.1689 | - |
| | | Dilithium | 1 | Signing | 65 | 200 | 790,495 | 4.9968 | - |
| Soni *et al.* [35] (HW) | DSA | FALCON | 1 | Verifying | 65 | 123 | 387,027 | 2.4400 | - |
| | | FALCON | 5 | Verifying | 65 | 173 | 380,350 | 3.2700 | - |
| Imran *et al.* [36] (HW) | KEM | SABER | 3 | Keygen/Encap/Decap | 65 | 936 | 1,026,000 | 860.9000 | - |
| | | SABER | 3 | Keygen/Encap/Decap | 40 | 1095 | 767,000 | 137.0000 | - |
| | | SABER | 3 | Keygen/Encap/Decap | 28 | 2500 | 255,000 | 608.4000 | - |
| Karl *et al.* [24] (HW/SW) | DSA | Dilithium | 1,3,5 | Keygen/Signing/Verifying | 22 | 800 | 166,991 | 8.1030 | - |
| | | FALCON | 1,5 | Verifying | | | | 3.4500 | - |
| **Ours** (HW/SW) | DSA | FALCON | 1,5 | Signing | 45 | 300 | **74,639** | **9.0487** | 229 |
| | | FALCON | 1,5 | Signing | 28 | 300 | **38,085** | **5.7972** | 229 |

DSA: digital signature algorithm. KEM: key encapsulation mechanism.

reduced area and power consumption within the area of PQC hardware accelerators.

## VII. RELATED WORK

Various PQC-DSA HW accelerator implementations were reported during the standardization of NIST PQC-DSA. Different design goals were considered, e.g. high performance or compactness. It is hard to compare between the algorithms as they are very distinct from each other. We note that FALCON uses less amount of data (up to 1793 bytes for public key and 1280 bytes for signature key) compared to Dilithium, another PQC-DSA algorithm (up to 2595 bytes and 4595 bytes for public key and signature key, respectively). As embedded devices cannot hold huge amounts of data, we deduced that the algorithm with less data involved would be more efficient. The main difference in terms of computation was the bottleneck for FALCON as it is more sequential than Dilithium, but we successfully reduced its overhead as described throughout our work. For Dilithium, a lattice-based PQC-DSA, several HW accelerators have been proposed [9], [10], [11], [12], [38]. In [9] and [11], high-performance HW accelerators were developed using parallelism. In [10], compact HW accelerators were designed using a SW-HW co-design method. An efficient HW implementation for Dilithium was presented in [38] by reusing digital signal processors. In [12], an efficient implementation of Dilithium was reported using a segmented pipelined processing method reducing memory resources and latency. Several works also targeted to accelerate multiple PQC-DSA algorithms at the same time [35], [39] using HLS-based HW designs. A high-performance HW accelerator was proposed for the multivariate algorithm, Rainbow, in [40].

## VIII. CONCLUSION

This paper presented an efficient FALCON accelerator, EFX designed for superior performance with minimal HW area usage. By implementing a shared resource strategy within FALCON's HW and designing a highly pipelined CCU, we achieved significant performance optimization with reduced hardware footprint. In addition, we optimized `SamplerZ`, which has a relatively large proportion of non-COBs in entire operations, by overlapping COB and non-COB pairs that can be executed concurrently, given no data dependencies.

Performance comparisons with recent SW/HW co-design paper for FALCON revealed that our EFX significantly outperforms the previous work in signature generation, achieving lower cycles of 3.58× for FALCON-1024 parameter, while occupying lower area than other hardware-only and SW/HW co-design PQC papers. In addition, our EFX included a significant proportion of the overall execution of signature generation. When implemented in different embedded systems, our EFX is expected to have a similar performance. Given FALCON's recognition by NIST as one of the standards for PQC-DSA, this work's insights into hardware-efficient FALCON implementations are poised to advance further research and development in secure cryptographic practices for the post-quantum era.

## ACKNOWLEDGMENT

*(Yongseok Lee and Jonghee Youn contributed equally to this work.)* The EDA tool was supported by the IC Design Education Center (IDEC), South Korea.

## REFERENCES

[1] N. J. G. Saho and E. C. Ezin, "Survey on asymmetric cryptographic algorithms in embedded systems," *IJISRT*, vol. 5, pp. 544–554, Dec. 2020.

[2] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Rev.*, vol. 41, no. 2, pp. 303–332, Jan. 1999.

[3] D. J. Bernstein, A. Hülsing, S. Kölbl, R. Niederhagen, J. Rijneveld, and P. Schwabe, "The SPHINCS + signature framework," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 2129–2146.

[4] P. A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang, *Falcon: Fast-Fourier Lattice-based Compact Signatures Over NTRU*, document Specification Version 1. 2, NIST Post-Quantum Cryptography Standardization Round, Mar. 2020, p. 67.

[5] S. Bai, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, *Crystals-dilithium Algorithm Specifications and Supporting Documentation*, document Version 3.1, NIST Post-Quantum Cryptography Standardization Round, Mar. 2021.

[6] NIST. (2016). *Announcing Request for Nominations for Public-Key Post-Quantum Cryptographic Algorithms*. [Online]. Available: https://www.federalregister.gov/documents/2016/12/20/2016-30615/announcing-request-for-nominations-for-public-key-post-quantum-cryptographic-algorithms

[7] P. He, T. Bao, Y. Tu, and J. Xie, "HPMA-Saber: High-performance polynomial multiplication accelerator for KEM saber," in *Proc. IEEE 40th Int. Conf. Comput. Design (ICCD)*, Oct. 2022, pp. 525–528.

[8] J. Xie, P. He, and C.-Y. Lee, "CROP: FPGA implementation of high-performance polynomial multiplication in saber KEM based on novel cyclic-row oriented processing strategy," in *Proc. IEEE 39th Int. Conf. Comput. Design (ICCD)*, Oct. 2021, pp. 130–137.

[9] S. Ricci, L. Malina, P. Jedlicka, D. Smékal, J. Hajny, P. Cibik, P. Dzurenda, and P. Dobias, "Implementing CRYSTALS-Dilithium signature scheme on FPGAs," in *Proc. 16th Int. Conf. Availability, Rel. Secur.*, Aug. 2021, pp. 1–11.

[10] Z. Zhou, D. He, Z. Liu, M. Luo, and K.-K.-R. Choo, "A software/hardware co-design of CRYSTALS-Dilithium signature scheme," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 14, no. 2, pp. 1–21, Jun. 2021.

[11] L. Beckwith, D. T. Nguyen, and K. Gaj, "High-performance hardware implementation of CRYSTALS-Dilithium," in *Proc. Int. Conf. Field-Programmable Technol. (ICFPT)*, Dec. 2021, pp. 1–10.

[12] C. Zhao, N. Zhang, H. Wang, B. Yang, W. Zhu, Z. Li, M. Zhu, S. Yin, S. Wei, and L. Liu, "A compact and high-performance hardware architecture for CRYSTALS-Dilithium," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, pp. 270–295, Nov. 2021.

[13] D. Amiet, A. Curiger, and P. Zbinden, "FPGA-based accelerator for post-quantum signature scheme SPHINCS-256," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, pp. 18–39, Feb. 2018.

[14] D. Amiet, L. Leuenberger, A. Curiger, and P. Zbinden, "FPGA-based SPHINCS+ implementations: Mind the glitch," in *Proc. 23rd Euromicro Conf. Digit. Syst. Design (DSD)*, Aug. 2020, pp. 229–237.

[15] Q. Berthet, A. Upegui, L. Gantel, A. Duc, and G. Traverso, "An area-efficient SPHINCS+ post-quantum signature coprocessor," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, Jun. 2021, pp. 180–187.

[16] P. Mohan, W. Wang, B. Jungk, R. Niederhagen, J. Szefer, and K. Mai, "ASIC accelerator in 28 nm for the post-quantum digital signature scheme XMSS," in *Proc. IEEE 38th Int. Conf. Comput. Design (ICCD)*, Oct. 2020, pp. 656–662.

[17] E. Karabulut and A. Aysu, "A hardware–software co-design for the discrete Gaussian sampling of falcon digital signature," *Cryptol. ePrint Arch.*, vol. 2023/908, p. 9, Jun. 2023. [Online]. Available: https://ia.cr/2023/908

[18] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *J. ACM*, vol. 56, no. 6, pp. 1–40, Sep. 2009.

[19] M. Ajtai, "Generating hard instances of lattice problems," in *Proc. 28th ACM Symp. Theory Comput.*, 1996, pp. 99–108.

[20] L. Ducas and T. Prest, "Fast Fourier orthogonalization," in *Proc. ACM Int. Symp. Symbolic Algebr. Comput.*, Jul. 2016, pp. 191–198.

[21] D. Sikeridis, P. Kampanakis, and M. Devetsikiotis, "Post-quantum authentication in TLS 1.3: A performance study," *Cryptol. ePrint Arch.*, vol. 2020/071, p. 16, Feb. 2020. [Online]. Available: https://ia.cr/2020/071

[22] N. Bindel, S. McCarthy, H. Rahbari, and G. Twardokus. (Jun. 2021). *Suitability of 3rd Round Signature Candidates for Vehicle-to-Vehicle Communication*. [Online]. Available: https://csrc.nist.gov/Presentations/2021/suitability-of-3rd-round-signature-candidates-for

[23] L. Beckwith, D. T. Nguyen, and K. Gaj, "Hardware accelerators for digital signature algorithms Dilithium and FALCON," *IEEE Design Test*, p. 7, Aug. 2023, doi: 10.1109/MDAT.2023.3305156.

[24] 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, Mar. 2024.

[25] T. Pornin, "New efficient, constant-time implementations of falcon," Cryptol. ePrint Arch., Tech. Rep. 2019/893, 2019. [Online]. Available: https://ia.cr/2019/893

[26] M. J. Kannwischer, J. Rijneveld, P. Schwabe, and K. Stoffelen, "pqm4: Testing and benchmarking NIST PQC on ARM Cortex-M4," in *Proc. 2nd PQC Standardization Conf.*, 2019, p. 22. [Online]. Available: https://hdl.handle.net/2066/210214

[27] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, no. 90, p. 297, Apr. 1965.

[28] W. M. Gentleman and G. Sande, "Fast Fourier transforms: For fun and profit," in *Proc. Fall Joint Comput. Conf.*, 1966, pp. 563–578.

[29] *STM32F4DISCOVERY Board*. Accessed: Mar. 13, 2022. [Online]. Available: https://www.st.com/en/evaluation-tools/stm32f4discovery.html

[30] ARM Ltd. *Data Watchpoint and Trace Unit*. [Online]. Available: https://developer.arm.com/documentation/ddi0439/b/Data-Watchpoint-and-Trace-Unit

[31] P. A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang, "Falcon-software," NIST, Tech. Rep. FALCON, 2020. [Online]. Available: https://csrc.nist.gov/projects/post-quantum-cryptography/selected-algorithms-2022

[32] A. Sarker, M. Mozaffari Kermani, and R. Azarderakhsh, "Efficient error detection architectures for postquantum signature Falcon's sampler and KEM SABER," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 30, no. 6, pp. 794–802, Jun. 2022.

[33] M. Imran, Z. Ul Abideen, and S. Pagliarini, "A systematic study of lattice-based NIST PQC algorithms: From reference implementations to hardware accelerators," 2020, *arXiv:2009.07091*.

[34] D. Soni, M. Nabeel, K. Basu, and R. Karri, "Power, area, speed, and security (PASS) trade-offs of NIST PQC signature candidates using a C to ASIC design flow," in *Proc. IEEE 37th Int. Conf. Comput. Design (ICCD)*, Nov. 2019, pp. 337–340.

[35] D. Soni, K. Basu, M. Nabeel, N. Aaraj, M. Manzano, and R. Karri, *Hardware Architectures for Post-Quantum Digital Signature Schemes*. Cham, Switzerland: Springer, 2021, doi: 10.1007/978-3-030-57682-0.

[36] M. Imran, A. Aikata, and S. S. Roy, "Towards high-speed asic implementations of post-quantum cryptography," *Cryptol. ePrint Arch.*, vol. 2023/716, p. 5, May 2023. [Online]. Available: https://ia.cr/2023/716

[37] T. Oder, J. Speith, K. Höltgen, and T. Güneysu, "Towards practical microcontroller implementation of the signature scheme falcon," in *Post-Quantum Cryptography* (Lecture Notes in Computer Science), vol. 11505, J. Ding and R. Steinwandt, Eds. Cham, Switzerland: Springer, 2019, doi: 10.1007/978-3-030-25510-7_4.

[38] G. Land, P. Sasdrich, and T. Güneysu, "A hard crystal-implementing Dilithium on reconfigurable hardware," *Cryptol. ePrint Arch.*, Sep. 2021.

[39] K. Basu, D. Soni, M. Nabeel, and R. Karri, "NIST post-quantum cryptography—A hardware evaluation study," Cryptol. ePrint Arch., Tech. Rep. 2019/047, 2019. [Online]. Available: https://ia.cr/2019/047

[40] A. Ferozpuri and K. Gaj, "High-speed FPGA implementation of the NIST round 1 rainbow signature scheme," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig)*, Dec. 2018, pp. 1–8.

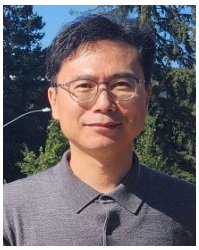

**YONGSEOK LEE** received the B.S. and M.S. degrees in electronic materials engineering from Kwangwoon University, South Korea, in 2015 and 2017, respectively. He was a SoC Designer with Korea Electronics Technology Institute, South Korea, from 2017 to 2021. He is currently pursuing the Ph.D. degree in electrical and computer engineering with Seoul National University, South Korea. His research interests include privacy-preserving computation and hardware acceleration solutions.

**JONGHEE YOUN** received the Ph.D. degree in EECS from Seoul National University, South Korea, in 2011. He is currently an Associate Professor of computer science and engineering with Yeungnam University. His research interests include compiler, software optimization, security, malware analysis, system software, and embedded systems (architecture/software).

**KEVIN NAM** received the B.S. degree in electrical and computer engineering from Seoul National University, South Korea, in 2020, where he is currently pursuing the Ph.D. degree in electrical and computing engineering. His research interests include privacy-preserving computation and hardware-backed system security against various types of threats.

**HEON HUI JUNG** received the B.S. degree from the Department of Electrical and Computer Engineering, University of Seoul, South Korea, in 2022. He is currently pursuing the M.S. degree in electrical and computing engineering with Seoul National University, South Korea. His research interests include privacy-preserving computation and hardware acceleration solutions and architectures.

**MYUNGHYUN CHO** received the B.S. degree from the Department of Electronic and Electrical Engineering, Hongik University, South Korea, in 2020, and the M.S. degree in electrical and computer engineering from Seoul National University, South Korea, in 2022. He is currently an Engineer with Samsung Electronics System LSI. His research interests include crypto IP and secure algorithms.



**JIMYUNG NA** received the B.S. and M.S. degrees from the Department of Electronics and Electrical Engineering, Chonnam National University, Gwangju, South Korea, in 1998 and 2000, respectively. In 2001, he joined Samsung Electronics System LSI, Hwaseong-si, South Korea, where he is currently a Principle Engineer with the Security Group, IP Development Team. His research interests include post-quantum cryptography, hardware security, high-performance crypto IP, and security systems.

**JONG-YEON PARK** received the master's degree in mathematics from Kookmin University, in 2012. He was a Researcher with the Electronics and Telecommunications Research Institute (ETRI), Daejeon, South Korea, from 2012 to 2014. He was also a Research Engineer with Korea Telecom (KT) Convergence Laboratory, Seoul, South Korea, from 2015 to 2017. He is currently a Staff Engineer with Samsung Electronics System LSI. His research interests include most of the cryptographer's topics, especially mathematical structures related to secure algorithms and SCA.

**SEUNGSU JEON** received the B.S. and M.S. degrees from the Department of Electronics and Electrical Engineering, Sungkyunkwan University, Suwon-si, South Korea, in 2015 and 2017, respectively. In 2017, he joined Samsung Electronics System LSI, Hwaseong-si, South Korea, where he is currently an Engineer with the Security Group, IP Development Team. His research interests include post-quantum cryptography, hardware security, high-performance crypto IP, and security systems.

**BO GYEONG KANG** received the B.S. degree in mathematics education from Seoul National University, South Korea, in 1999, and the M.S. and Ph.D. degrees in mathematics from Korea Advanced Institute of Science and Technology, KAIST, in 2001 and 2005, respectively. She is currently the Security Group Leader of the IP Development Team, Samsung Electronics System LSI. Her research interests include device security systems with software and hardware, secure processor design integrated in SoC which satisfies security certification common criteria (CC), and new algorithm implementation.

**HYUNYOUNG OH** received the B.S. and M.S. degrees in electrical and electronic engineering from Yonsei University, South Korea, in 2005 and 2007, respectively, and the Ph.D. degree in electrical and computing engineering from Seoul National University. He was a SoC Designer with Samsung Electronics, South Korea, from 2007 to 2017. He is currently a Professor with the Department of AI Software, Gachon University. His research interests include privacy-preserving computation and hardware-backed system security against various types of threats.

**YUNHEUNG PAEK** (Member, IEEE) received the B.S. and M.S. degrees in computer engineering from Seoul National University, South Korea, in 1988 and 1990, respectively, and the Ph.D. degree in computer science from the University of Illinois at Urbana–Champaign, in 1997. Currently, he is a Professor with the Department of Electrical and Computer Engineering, Seoul National University. His research interests include system security with hardware, secure processor design against various types of threats, and machine learning-based security solutions.