# Post-Quantum Signatures on RISC-V with Hardware Acceleration

PATRICK KARL, Technical University of Munich, Department of Electrical and Computer Engineering, Chair of Security in Information Technology, Germany
JONAS SCHUPP, Technical University of Munich, Department of Electrical and Computer Engineering, Chair of Security in Information Technology, Germany
TIM FRITZMANN, Technical University of Munich, Department of Electrical and Computer Engineering, Chair of Security in Information Technology, Germany
GEORG SIGL, Technical University of Munich, Department of Electrical and Computer Engineering, Chair of Security in Information Technology, Germany; Fraunhofer Institute for Applied and Integrated Security, Germany

CRYSTALS-Dilithium and Falcon are digital signature algorithms based on cryptographic lattices, that are considered secure even if large-scale quantum computers will be able to break conventional public-key cryptography. Both schemes are third round candidates in the ongoing NIST post-quantum competition. In this work, we present a RISC-V HW/SW codesign that aims to combine the advantages of software- and hardware implementations, i.e. flexibility and performance. It shows the use of flexible hardware accelerators, which have been previously used for Public-Key Encryption (PKE) and Key-Encapsulation Mechanism (KEM), for post-quantum signatures. It is optimized for Dilithium as a generic signature scheme but also accelerates applications that require fast verification of Falcon's compact signatures. We provide a comparison with previous works showing that for Dilithium and Falcon, cycle counts are significantly reduced, such that our design is faster than previous software implementations or other HW/SW codesigns. In addition to that, we present a compact Globalfoundries 22 nm ASIC design that runs at 800 MHz. By using hardware acceleration, energy consumption for Dilithium is reduced by up to 92.2%, and up to 67.5% for Falcon's signature verification.

Additional Key Words and Phrases: Post-Quantum, NIST PQC, Digital Signatures, HW/SW Codesign, CRYSTALS-Dilithium, Falcon, RISC-V

## 1 INTRODUCTION

In 1994, Peter Shor presented an algorithm that severely threatens today's cryptography under the assumption that large-scale quantum computers are available [28]. This algorithm, simply known as Shor's algorithm, allows to solve the problems of discrete logarithm and integer factorization within feasible time. These problems are the basis for today's public-key cryptography and thus, new ways must be found to provide secure communication. One class of algorithms that fulfills these requirements is called lattice-based cryptography. Due to their efficiency with respect to performance and parameter sizes, these algorithms seem to be well suited for constrained devices.

In order to evaluate the different options and define new standards for post-quantum secure algorithms, the National Institute of Standards and Technology (NIST) initiated the Post-Quantum Cryptography (PQC) competition[1]. The process started in 2016 and consists of two tracks. The

[1]https://csrc.nist.gov/projects/post-quantum-cryptography

Authors' addresses: Patrick Karl, patrick.karl@tum.de, Technical University of Munich, Department of Electrical and Computer Engineering, Chair of Security in Information Technology, Arcisstr. 21, Munich, 80333, Bavaria, Germany; Jonas Schupp, jonas.schupp@tum.de, Technical University of Munich, Department of Electrical and Computer Engineering, Chair of Security in Information Technology, Arcisstr. 21, Munich, 80333, Bavaria, Germany; Tim Fritzmann, tim.fritzmann@tum.de, Technical University of Munich, Department of Electrical and Computer Engineering, Chair of Security in Information Technology, Arcisstr. 21, Munich, 80333, Bavaria, Germany; Georg Sigl, sigl@tum.de, Technical University of Munich, Department of Electrical and Computer Engineering, Chair of Security in Information Technology, Munich, Germany; and Fraunhofer Institute for Applied and Integrated Security, Garching, Bavaria, Germany.

first track contains PKE and KEMs and is currently in its third and final round. It is expected that NIST will announce the winners of this track in early 2022. The second track contains Digital Signature Algorithms (DSAs) and NIST decided to extend this competition to a fourth round, whose candidates will also be expected to be announced early 2022.

Among the third round DSA candidates are two lattice-based schemes, i.e. CRYSTALS-Dilithium [22] and Falcon [10]. Dilithium is built on the Fiat-Shamir with Aborts [21] principle and bases its security on the Module Learning with Errors (M-LWE) and Module Short Integer Solution (M-SIS) problem. Falcon, however, is a hash-and-sign signature scheme that operates on NTRU lattices and requires a trapdoor sampler for signature generation. Similar to Dilithium, Falcon's security is based on the Shortest Integer Solution (SIS) problem for NTRU lattices. Both schemes have different advantages. An advantage of Dilithium is that its operations are quite simple compared to Falcon's operations. More concretely, Dilithium performs polynomial multiplication with integer coefficients, whereas Falcon also operates on complex polynomials. In addition to that, Dilithium only requires uniform sampling, whereas Falcon requires Gaussian sampling with varying center and standard deviation. However, an advantage of Falcon is its compactness and performance of signature verification, which can be crucial for certain applications like authenticated firmware updates. Therefore, a unified design with Dilithium as a signature scheme and Falcon for fast signature verification is a desirable goal for a wide range of applications.

*Related Works:* Several designs implementing Dilithium on embedded systems have been presented in the past. Round 2 versions of the NIST submissions have been implemented in software [15, 16, 26], in hardware [3, 4, 27, 29, 30] or as a HW/SW codesign [32]. For round 2 Falcon versions, the verification procedure has been implemented in [29]. For the final round versions, several implementations for Dilithium [1, 2, 5, 6, 17, 20, 31] and Falcon [6, 24, 25] were published. In [7], Bos et al. optimized a platform independent C implementation of Dilithium with the goal of low memory consumption. Most of these designs are either pure SW implementations benefiting from its flexibility, or HW implementations benefiting from high performance, but lacking flexibility. Combining both advantages by integrating specific hardware accelerators into RISC-V platforms has previously been applied to several lattice-based KEMs e.g. [11, 12]. Inspired by this, Nannipieri et al. integrate a Dilithium-tailored ALU that supports several operations required for polynomial arithmetic into an open-source 64 bit RISC-V processor [23].

*Contribution:* In this work, we present a hardware accelerated RISC-V platform for efficient signature generation and verification. The design integrates accelerators that fully support the Dilithium signature scheme but also speed up signature verification for Falcon signatures. This combines the advantages of both schemes, i.e. an overall efficient signature scheme (Dilithium) with application-specific fast verification of compact signatures (Falcon). Additionally, supporting flexible signature verification of two different schemes allows for authenticated firmware updates even if the security of one scheme is threatened. This increases the confidence in the system when migrating towards post-quantum cryptography. In contrast to the 64 bit design in [23], our design is based on a 32 bit RISC-V platform and also accelerates the Keccak-based shake functions that are used in several lattice-based NIST submissions. We therefore provide:

- An assembled HW/SW codesign supporting all parameter sets of 1) the full Dilithium scheme and 2) Falcon verification in a unified design.
- A step towards a crypto-agile system by supporting accelerated verification of two different signature schemes for authentication.
- A comparison of cycle counts with previous work showing improvements over previous SW implementations for embedded systems and HW/SW codesigns.

- An analysis of resource cost in terms of memory consumption using hardware acceleration for signature schemes.
- A 22nm Globalfoundries chip design running at 800 MHz including numbers for area, power, and energy consumption. The design shows significant reduction in energy consumption by making use of hardware acceleration.

*Organization:* Section 2 gives an introduction of the signature schemes and some mathematical notation is introduced. The system architecture and integrated accelerators are presented in Section 3. The results with respect to performance and area numbers are discussed in Section 4 and the 22 nm ASIC design is presented. Finally, Section 5 concludes this work.

## 2 PRELIMINARIES

### 2.1 Polynomial Notation and Number Theoretic Transform (NTT)

Let $\mathcal{R}_q = \mathbb{Z}_q/\phi(x)$ denote a polynomial ring with an integer modulus $q$ and the cyclotomic polynomial $\phi(x)$. Dilithium and Falcon are both specified for $\phi(x) = x^n + 1$. Let $a \in \mathcal{R}_q$ be a single polynomial. Vectors of polynomials are written in bold lowercase letters, i.e. $\boldsymbol{b} \in \mathcal{R}_q^l$ and matrices of polynomials in bold uppercase $\boldsymbol{A} \in \mathcal{R}_q^{k \times l}$ for dimensions $k$ and $l$. We denote with $\leftarrow \mathcal{U}$ the sampling of uniformly distributed random bits.

Polynomial arithmetic is often performed in Number Theoretic Transform (NTT) domain, which is efficient for polynomial multiplication. It can be seen as a variant of the Fast Fourier Transform (FFT) operating in the finite field $\mathbb{Z}_q$ instead of $\mathbb{C}$. Using the NTT transformation effectively reduces the complexity of polynomial multiplication from $O(n^2)$ down to $O(n \log_2 n)$. That is, for two polynomials $a, b \in \mathcal{R}_q/\phi(x)$, the product $c = a \cdot b$ can be efficiently calculated as $c = NTT^{-1}(NTT(a) \odot NTT(b))$, where $\odot$ denotes the coefficient wise multiplication and $NTT()$ and $NTT^{-1}()$ the transformation to and from NTT domain, respectively.

Let $a$ be a polynomial of degree $n-1$, the transformation $\hat{a} = NTT(a)$ and $a_i$ (resp. $\hat{a}_i$) denote the $i$-th coefficient of $a$ (resp. $\hat{a}$). Then the functions $NTT()$ and $NTT^{-1}()$ are defined as follows:

$$NTT(a): \quad \hat{a}_i = \sum_{j=0}^{n-1} \gamma^j \cdot \omega_n^{ij} \cdot a_j \qquad NTT^{-1}(\hat{a}): \quad a_i = \frac{1}{n} \cdot \gamma^{-i} \sum_{j=0}^{n-1} \omega_n^{-ij} \cdot \hat{a}_j \tag{1}$$

$\omega_n \in \mathbb{Z}_q$ is called the $n$-th root of unity such that $\omega_n^n = 1 \ mod\ q$ and $\omega_n^k \neq 1 \ mod\ q$, $\forall k \in [1, n-1]$. The powers of $\omega_n$ are usually called twiddle factors and can either be precomputed and stored in memory, or can be computed on-the-fly when needed. Furthermore, $\gamma_n$ is the $2n$-th root of unity and allows to use a length-$n$ NTT instead of a length-$2n$ NTT. They can also be precomputed and merged into the twiddle factors.

For realizing the NTT transformations, there exist mainly two algorithms, i.e. Cooley-Tukey [8] and Gentleman-Sande [14]. Due to the structure of their equations as shown in Eqs. (2) and (3), they are usually referred to as butterfly units.

$$\begin{aligned} x' &= x + y\omega_n \\ y' &= x - y\omega_n \end{aligned} \tag{2}$$

Eq. 2. Cooley-Tukey butterfly

$$\begin{aligned} x' &= x + y \\ y' &= (x - y)\omega_n \end{aligned} \tag{3}$$

Eq. 3. Gentleman-Sande butterfly

Both signature schemes Dilithium and Falcon have parameters chosen such that NTT can be efficiently used. Therefore, the rest of this work assumes NTT multiplications whenever polynomials or vectors of polynomials are multiplied.

### 2.2 CRYSTALS-Dilithium

In the following, an introduction of Dilithium according to the official NIST submission specification is given [22]. The signature scheme is based on the *Fiat Shamir with Aborts* principle [21] and is a third round candidate in the ongoing NIST PQC competition. The latest version provides parameter sets for three NIST security levels: Dilithium-II, Dilithium-III and Dilithium-V for security levels 2, 3 and 5. The underlying security is based on the M-LWE and M-SIS problem operating on vectors of polynomials. In Dilithium, the M-LWE distribution is denoted as $(\boldsymbol{A},\ \boldsymbol{t} = \boldsymbol{A}\boldsymbol{s_1} + \boldsymbol{s_2})$ for $\boldsymbol{A} \in \mathcal{R}_q^{k\times l}$, $\boldsymbol{s_1} \in \mathcal{R}_q^l$ and $\boldsymbol{s_2} \in \mathcal{R}_q^k$, whereas all elements $\boldsymbol{A}, \boldsymbol{s_1}$ and $\boldsymbol{s_2}$ are uniformly distributed and $\boldsymbol{s_1}$. The M-SIS problem is defined as finding an $\boldsymbol{x} \in \mathcal{R}_q^l$ such that $\boldsymbol{A}\boldsymbol{x} = 0$ for $\boldsymbol{A} \in \mathcal{R}_q^{k\times l}$ and the norm of $\boldsymbol{x}$ is below some predefined boundary $\beta$.

The polynomials have dimension $n = 256$ for all security levels. Similarly, the modulus $q = 2^{23} - 2^{13} + 1 = 8,380,417$ is the same for each parameter set. All operations for seed extension, sampling and hashing are performed using the Keccak based functions `shake128` or `shake256`. The dimensions $k$ and $l$ are set to $(4, 4)$, $(6, 5)$ and $(8, 7)$ for the three security levels.

*Key Generation:* Algorithm 1 provides a simplified version of the key generation procedure. In a first step, the seeds $\rho$ and $\rho'$ as well as the secret $K$ are sampled uniformly random. Note, that in the official document, they are not directly sampled, but are derived from a hash function seeded with a truly random seed. As these details are not necessarily relevant for our work, we omit them from Algorithm 1 for simplicity. In line 2 of Algorithm 1, the public matrix $\boldsymbol{A} \in R_q^{k\times l}$ is expanded from $\rho$. All sampled coefficients that are not in the range of $[0, q-1]$ are rejected. As $\boldsymbol{A}$ is used for multiplications later on, it is directly sampled and stored in NTT representation. The secret polynomial vectors $\boldsymbol{s_1}$ and $\boldsymbol{s_2}$ are expanded from $\rho'$ via rejection sampling, i.e. only the coefficients in the range $[-\eta, \eta]$ are kept. The multiplication $\boldsymbol{A}\boldsymbol{s_1}$ is performed in NTT domain and as $\boldsymbol{A}$ is already sampled in NTT domain, only $\boldsymbol{s_1}$ has to be converted. Finally, `Power2Round()` splits $\boldsymbol{t}$ into an upper part and lower part, which are then part of public key and secret key. This also serves as a compression of the public key, as only the upper part $\boldsymbol{t_1}$ of $\boldsymbol{t}$ has to be transmitted. The lower part $\boldsymbol{t_0}$ is then recovered during verification by the use of a hint. $tr$ is the hash of the public key $(\rho, \boldsymbol{t_1})$ computed by the hash function $H$.

**Algorithm 1** Dilithium KeyGen

**Require:** -
**Ensure:** $(pk, sk)$

1: $\rho, \rho', K \leftarrow \mathcal{U}$ ▷ Uniformly sampled
2: $\boldsymbol{A} \leftarrow \mathsf{ExpandA}(\rho)$ ▷ $\boldsymbol{A} \in R_q^{k\times l}$, sampled in NTT domain
3: $(\boldsymbol{s_1}, \boldsymbol{s_2}) \leftarrow \mathsf{ExpandS}(\rho')$ ▷ $\boldsymbol{s_1}, \boldsymbol{s_2} \in S_\eta^l \times S_\eta^k$
4: $\boldsymbol{t} \leftarrow \boldsymbol{A}\boldsymbol{s_1} + \boldsymbol{s_2}$
5: $(\boldsymbol{t_1}, \boldsymbol{t_0}) \leftarrow \mathsf{Power2Round}(\boldsymbol{t})$
6: $tr \leftarrow H(\rho \parallel \boldsymbol{t_1})$
7: **return** $pk = (\rho, \boldsymbol{t_1})$, $sk = (\rho, K, tr, \boldsymbol{s_1}, \boldsymbol{s_2}, \boldsymbol{t_0})$

*Signature Generation:* Algorithm 2 provides an overview of the signature generation process in Dilithium. The core part is the rejection loop starting at line 5. On an abstract level, a candidate signature $\boldsymbol{z}$ is generated for a challenge $c$ and has to pass two security checks in line 13. The first security check verifies that the norm of $\boldsymbol{z}$ is sufficiently small and does not leak information about the secret $\boldsymbol{s_1}$. The second check is required for security but also for correctness: As $\boldsymbol{t_0}$ is not part of

the public key due to compression, a hint must specify which parts of $\boldsymbol{t_1}$ require carry bits during signature verification. If too many carry bits would be required, no hint can be generated. As a result, if one of these checks fail, i.e. the $if$-branch is taken, the rejection loop must restart and a new candidate signature is generated. On success, the hint $\boldsymbol{h}$ is generated, and the signature tuple $(\tilde{c}, \boldsymbol{z}, \boldsymbol{h})$ is returned.

Inspecting Algorithm 2, it shows that there are several hash operations and seed expansions again using `shake128` and `shake256`. Furthermore, the multiplications $\boldsymbol{Ay}$, $c\boldsymbol{s_1}$, $c\boldsymbol{s_2}$ and $c\boldsymbol{t_0}$ are performed in NTT domain. For definitions of the functions `HighBits()`, `LowBits()`, `SampleInBall()` or `MakeHint`, we refer to the official specification [22].

**Algorithm 2** Dilithium Sign

**Require:** Secret key $sk$, message $M$
**Ensure:** Signature $\sigma$

1: $\boldsymbol{A} \leftarrow \texttt{ExpandA}(\rho)$ ▷ $\boldsymbol{A} \in R_q^{k \times l}$, sampled in NTT domain
2: $\mu \leftarrow H(tr \parallel M)$
3: $\rho' \leftarrow H(K \parallel \mu)$
4: $(\boldsymbol{z}, \boldsymbol{h}) = \perp, \kappa = 0$
5: **while** $(\boldsymbol{z}, \boldsymbol{h}) = \perp$ **do** ▷ Start of rejection loop
6: $\quad \boldsymbol{y} \leftarrow \texttt{ExpandMask}(\rho', \kappa)$ ▷ $\boldsymbol{y} \in \tilde{S}_{\gamma_1}^l$
7: $\quad \boldsymbol{w} \leftarrow \boldsymbol{Ay}$
8: $\quad \boldsymbol{w_1} \leftarrow \texttt{HighBits}(\boldsymbol{w})$
9: $\quad \tilde{c} \leftarrow H(\mu \parallel \boldsymbol{w_1})$
10: $\quad c \leftarrow \texttt{SampleInBall}(\tilde{c})$
11: $\quad \boldsymbol{z} \leftarrow \boldsymbol{y} + c\boldsymbol{s_1}$
12: $\quad \boldsymbol{r_0} \leftarrow \texttt{LowBits}(\boldsymbol{w} - c\boldsymbol{s_2})$
13: $\quad$ **if** $\|z\|_\infty \geq \gamma_1 - \beta$ OR $\|\boldsymbol{r_0}\|_\infty \geq \gamma_2 - \beta$ **then**
14: $\quad\quad (\boldsymbol{z}, \boldsymbol{h}) = \perp$ ▷ Bad signature, reiterate
15: $\quad$ **else**
16: $\quad\quad \boldsymbol{h} \leftarrow \texttt{MakeHint}(-c\boldsymbol{t_0}, \boldsymbol{w} - c\boldsymbol{s_2} + c\boldsymbol{t_0})$ ▷ Good signature
17: $\quad$ **end if**
18: $\quad \kappa \leftarrow \kappa + l$
19: **end while**
20: **return** $\sigma = (\tilde{c}, \boldsymbol{z}, \boldsymbol{h})$

*Signature Verification:* Finally, Algorithm 3 provides an overview of the Dilithium signature generation. In lines 3-4, the challenge $c$ is recovered from the signature and the hint $\boldsymbol{h}$ is used to recreate $\boldsymbol{w_1'}$. Both parts are then used to perform several checks in line 5. At first, it is checked whether the norm of the received signature part $\boldsymbol{z}$ is within its defined boundary $\gamma_1 - \beta$. In addition to that, $\boldsymbol{w_1'}$ is hashed with $\mu$ and the result is compared to match the transmitted challenge $\tilde{c}$. Finally, the correct format of the hint $\boldsymbol{h}$ is verified. Only if all of these checks are valid, the signature is accepted, otherwise it is rejected.

### 2.3 Falcon

Falcon is a hash-and-sign signature scheme that operates on NTRU lattices and makes use of a trapdoor sampler. It also operates on polynomial rings. However, Falcon partially computes on complex numbers and requires double-precision floating point arithmetic. This makes the scheme

**Algorithm 3** Dilithium Verify

**Require:** Public Key $pk$, message $M$, signature $\sigma = (\tilde{c}, \boldsymbol{z}, \boldsymbol{h})$
**Ensure:** Accept or reject

1: $\boldsymbol{A} \leftarrow \text{ExpandA}(\rho)$ ▷ $\boldsymbol{A} \in R_q^{k \times l}$, sampled in NTT domain
2: $\mu \leftarrow H(\, H(\rho \parallel \boldsymbol{t_1}) \parallel M)$
3: $c \leftarrow \text{SampleInBall}(\tilde{c})$
4: $\boldsymbol{w_1'} \leftarrow \text{UseHint}(\boldsymbol{h}, \boldsymbol{Az} - c\boldsymbol{t_1} \cdot 2^d)$
5: **if** $\|\boldsymbol{z}\|_\infty < \gamma_1 - \beta$ AND $\tilde{c} = H(\mu \parallel \boldsymbol{w_1'})$ AND # of 1's in $\boldsymbol{h} \leq \omega$ **then**
6: **return** *accept*
7: **else**
8: **return** *reject*
9: **end if**

less efficient for embedded devices that usually don't provide a Floating Point Unit (FPU), let alone with double-precision. In such cases, floating point operations must be emulated with integer arithmetic, which is even less efficient. The effect of the FPU emulation for Falcon has been analyzed in [18] in more detail. Falcon is specified for the two parameter sets Falcon-512 and Falcon-1024 for NIST security levels 1 and 5. The lattice dimensions are $n = 512$ for Falcon-512 and $n = 1024$ for Falcon-1024. The modulus $q = 12,289$ is the same for both parameter sets. In contrast to Dilithium, Falcon does require Gaussian sampling in addition to uniform sampling.

Algorithm 4 provides a simplified version of the signature verification procedure according to [10]. Note, that in the original specification, the Falcon signature is $(r, s)$ where $s$ is a compressed version of $s_2$. Therefore, $s$ has to be decompressed before being processed. We omitted this step as it is not relevant for the remainder of this work and assume that $s_2$ is directly part of the signature. Falcon's verification routine is only operates on integer polynomials and is compact by design. It consists of basically three steps: At first, the message $M$ is hashed with the salt $r$ to a point $c$. Then, the signature polynomial $s_1$ is recovered from $c$, $s_2$ and the public key polynomial $h$. Finally, it is checked whether the norm of the signature tuple $(s_1, s_2)$ is within its defined boundary $\lfloor \beta^2 \rfloor$. Therefore, the verification only consists of hashing, a single NTT multiplication and some straightforward polynomial operations (i.e. addition/subtraction, norm computation).

**Algorithm 4** Falcon Verify

**Require:** Public key $pk = h$, message $M$, signature $\sigma = (r, s_2)$, acceptance bound $\lfloor \beta^2 \rfloor$
**Ensure:** Accept or reject

1: $c \leftarrow \text{HashToPoint}(r \parallel M)$
2: $s_1 \leftarrow c - s_2 h \bmod q$
3: **if** $\parallel (s_1, s_2) \parallel^2 \leq \lfloor \beta^2 \rfloor$ **then**
4: *accept*
5: **else**
6: *reject*
7: **end if**

Inspecting Algorithms 1 to 4 indicates, that there are mainly two operations frequently used. One common operation in lattice-based cryptography is the use of a hash function for hashing or sampling. For Dilithium and Falcon this is both realized by the Keccak based `shake128` and

shake256 functions, which are part of the SHA-3 standard. The PQM4 benchmark [2] has evaluated, that hashing makes up about 67.1%/37.3%/60.1% of the overall computation time of Dilithium-2's key generation, signature generation and verification, respectively. For the higher security levels, this percentage even increases. For Falcon, the benchmark reports 29.3% and 26.9% of execution time accounted for hash operations for security level 1 and 5.

The second frequent operation is the polynomial multiplication. Although in Falcon's verify function only one multiplication is performed, it is a frequent operation in Dilithium. For instance, the matrix-vector multiplications $\boldsymbol{As_1}$, $\boldsymbol{Ay}$ and $\boldsymbol{Az}$ in the keygen, sign and verify procedure involve several polynomial multiplications.

As a result, it is an obvious choice to accelerate the NTT transformation and pointwise multiplications, as well as the functions shake128 and shake256 using dedicated hardware. The accelerators and the resulting HW/SW system will be subject of Section 3.

### 2.4 Parameter Comparison of Dilithium and Falcon

A summary of the parameters of Dilithium and Falcon is provided in Table 1. It depicts the remarkable compactness of Falcon with respect to the size of the public key and signature. Comparing Dilithium-V with Falcon-1024, it shows that Falcon's combined size of the public key and signature is less then half of Dilithium's size for the same security level. This underlines the suitability of Falcon for applications, that mostly require fast and efficient signature verification. Therefore, our

Table 1. Parameter comparison between Dilithium and Falcon

| | Dilithium-II | Dilithium-III | Dilithium-V | Falcon-512 | Falcon-1024 |
|---|---|---|---|---|---|
| NIST level | 2 | 3 | 5 | 1 | 5 |
| Ring degree $n$ | 256 | 256 | 256 | 512 | 1024 |
| Modulus $q$ | 8, 380, 417 | 8, 380, 417 | 8, 380, 417 | 12, 289 | 12, 289 |
| $\lceil log_2(q) \rceil$ | 23 | 23 | 23 | 14 | 14 |
| $\lvert pk \rvert$ (B) | 1, 312 | 1, 952 | 2, 592 | 897 | 1, 793 |
| $\lvert sig \rvert$ (B) | 2, 420 | 3, 293 | 4, 595 | 666 | 1, 280 |
| $\lvert pk \rvert + \lvert sig \rvert$ (B) | 3, 732 | 5, 245 | 7, 187 | 1, 563 | 3, 073 |

goal is to provide a generic HW/SW codesign platform that accelerates 1) Dilithium as a generic and efficient signature scheme and 2) fast verification for compact Falcon signatures.

## 3 SYSTEM DESIGN AND ACCELERATORS

As a baseline for our design we chose the PULPino[1] microcontroller from the Parallel Ultra-Low-Power (PULP) project, originally developed in collaboration between ETH Zurich and the University of Bologna. This microcontroller instantiates CV32E40P (formerly known as RI5CY), a single core 4-stage pipeline RISC-V processor [13]. As the core and the Instruction Set Architecture (ISA) is fully open-source, it can be easily extended with further instructions and hardware accelerators. For software compilation the corresponding PULP compilation toolchain[2] with flag -O3 has been used. A Xilinx UltraScale+ FPGA (xczu9eg-ffvb1156-2e) is used as test platform to implement the RISC-V core.

[2]https://github.com/mupq/pqm4/blob/master/benchmarks.md
[1]https://github.com/pulp-platform/pulpino
[2]https://github.com/pulp-platform/pulp-riscv-gnu-toolchain

When designing hardware accelerators, there are basically two design options. One can add custom ISA instructions to the system and integrate the corresponding accelerators directly into the pipeline of the processor. This form is commonly referred to as *tightly-coupled* accelerators and is very well suited for lightweight operations where the accelerators have relatively low resource consumption. A second option is to implement a dedicated, standalone accelerator that is connected to the system bus and has its own address space. This type is often called *loosely-coupled* and is suitable for large accelerators that do a lot of processing for one chunk of data. However, the speed-up of these computations must be large enough to compensate for the communication overhead between the processor and the accelerator.

In this work, we chose the tightly-coupled approach for accelerating the `shake128/256` functions and the loosely-coupled approach for acceleration of the NTT transformations and polynomial arithmetic as explained in the following.

### 3.1 Keccak Accelerator

The functions `shake128/256` are part of the SHA-3 [9] standard and use the Keccak primitive. Keccak computes on a 1600 bit state that is permuted by a non-linear round function called `Keccak-f1600`. In contrast to implementing the whole Keccak primitive with its 1600 bit state as a separate accelerator that is connected to the system bus, we opt to take the approach presented in [12]. In this work, Fritzmann et al. propose to implement only the round function `Keccak-f1600` as a tightly-coupled accelerator that is connected to the processors register file. A custom RISC-V instruction is implemented that performs a single round of the permutation. For the final `shake128/256` functions, a designer can use the corresponding C implementation and simply replace the round function by the corresponding assembly instruction.

For the state, 50 registers of 32 bit are required. Therefore, [12] proposes to use the 32 Floating Point Registers (FPRs) as well as 18 additional General Purpose Registers (GPRs). In our system, however, we do not make use of the FPU and thus we have to enable only the FPR and corresponding load/store instructions. Under these circumstances, one can say that the accelerator comes with 32 additional FPR registers (1024 bit) overhead, but still uses the 18 registers from the GPR (576 bit) without area overhead. In case the complete FPU is used in the system anyway, saving the state does not impose additional resource cost at all. Therefore, we consider the tightly-coupled accelerator for `shake128/256` as an appropriate compromise between performance and resource consumption. Table 2 states the resource overhead introduced by the Keccak accelerator.

Table 2. Resource overhead of the Keccak accelerator measured for a Xilinx UltraScale+ FPGA

| | LUTs | FFs | BRAMs | DSPs |
|---|---|---|---|---|
| Keccak | 4,782 | 1,050 | 0 | 0 |

Using this tightly-coupled approach, the rejection sampling of the matrix $\boldsymbol{A}$ can be further optimized. As the state is stored directly in the registers, the rejection sampling can be performed without first storing the Keccak squeeze to memory, and afterwards load it again to perform rejection sampling.

### 3.2 NTT Accelerator

In order to accelerate the NTT transformations for both signature schemes, a design with a certain amount of flexibility is required. Table 1 shows that a unified design must provide support for ring dimensions $n = 256$, $n = 512$ and $n = 1024$ as well as support for the prime moduli $q = 8,380,417$

and $q = 12,289$. One suitable approach would be to integrate a generic butterfly unit and modular multiplier tightly-coupled into the processor pipeline and perform the control logic in pure software. Such an approach has been chosen for instance in [12, 23]. A second option is to integrate a generic accelerator for NTT transformation and polynomial arithmetic as a loosely-coupled solution connected to the system bus. Such an approach was presented in [11], where a generic NTT-based accelerator has been designed with multiple features. More concretely, it has runtime configurable support for the following:

- Dimension $n$ up to 4096, modulus $q$ up to 39 bit
- Positive and negative wrapped convolutions
- Early abort functionality for incomplete NTTs
- Pointwise multiplication, addition and subtraction

The authors of [11] state, that their design goal was to support a wide range of parameter sets, such that it can be used for all the lattice-based candidates of the NIST competition.

For our purpose, we decided to opt for the second approach with the loosely-coupled, generic NTT accelerator. The reason is, that a standalone accelerator comes with dedicated memory for storing twiddle factors and polynomial coefficients. That means, that the twiddle factors can be written once into the accelerator's memory and reside there, even for multiple consecutive transformations. With a tightly-coupled solution, however, the twiddle factors reside in the system memory and single elements must be loaded over again in consecutive computations. For Dilithium's signing procedure shown in Algorithm 2, there are several occasions where this effect becomes visible. That is, in line 7, $\boldsymbol{y}$ is transformed into NTT domain for multiplication and afterwards, the result $\omega$ is transformed back into normal domain. The challenge $c$ that is used for multiplication in line 11 and 12 is transformed to NTT domain and the results $\boldsymbol{z}$ and $\boldsymbol{r_0}$ of line 11 and 12 are transformed back. All these transformations are computed inside the rejection loop until a valid signature is found. Therefore, we prefer having a dedicated memory for the twiddle factors inside the accelerator and a hardware controller for reading them from the dedicated memory. Furthermore, the standalone solution allows to keep Dilithium's challenge polynomial $c$, that is processed several times within one loop iteration, inside the accelerator.

Nevertheless, we do not require all the features that were integrated into the NTT accelerator of [11]. In order to reduce the resource consumption, we modify the accelerator as follows:

(1) *Reducing parameter support:* The presented accelerator supports $n$ up to 4094 and a modulus $q$ up to 39 bit. However, we only require support for $n$ up to 1024 and a 23 bit modulus $q$ as shown in Table 1. The reason for the 39 bit prime support in [11] is, that non NTT-friendly moduli can be lifted to NTT-friendly primes $q'$, such that $q' > nq^2$. In this context, [11] chose a 39 bit Solinas prime which allows easy reduction. As Dilithium and Falcon use NTT-friendly parameters, we do not require this feature. Therefore, the reduction circuit for the Solinas prime can be removed.
(2) *Removing support for positive wrapped convolution:* Both algorithms in this work use the reduction polynomial $\phi(x) = x^n + 1$, i.e. negative wrapped convolutions.
(3) *Removing support for incomplete NTTs:* Dilithium and Falcon both use parameters where the $n$-th root as well as the $2n$-th root of unity exist. Therefore, complete NTT transformations can be performed.
(4) *Reducing memory size:* Due to its generic character, the accelerator features two memory blocks (one for coefficients, one for twiddle factors) each of dimension $39 \times 4096$. For our use case, we resize the twiddle memory to $32 \times 2048$, as we need only $2n$ twiddle factors, i.e. at most 2048 in the case of Falcon-1024 (1024 for forward and reverse NTT each). The coefficient memory is configured to $32 \times 3584$. This allows to pack two 14 bit coefficients (stored as

16 bit variables) of Falcon in one memory word. Dilithium's matrix-vector multiplication with $\boldsymbol{A}$ is an iterative multiplication of two vectors of size $l$. For efficiency reasons, we want to have enough space in the accelerator memory to store both vectors. As Dilithium has a 23 bit modulus, we require $2 \times l \times n$ coefficients in the accelerator. For Dilithium-5 it is $l = 7$ and thus, $2 \times 7 \times 256 = 3584$ coefficients/words must be stored.

With these modifications, the overall size of the NTT accelerator in [11] can be reduced. Table 3 depicts the savings in terms of FPGA resources. For the baseline version, we synthesized the accelerator as stated in [11] for our platform. The savings are mainly caused by removing the support for the 39 bit Solinas prime, which requires some extra reduction circuit. Furthermore, the memory consumption is reduced by a third, which represents a saving of more than 12 kB.

Table 3. Resource consumption for baseline and modified NTT on a Xilinx UltraScale+ FPGA

| Version | LUTs | FFs | BRAMs | DSPs |
|---|---|---|---|---|
| Baseline [11] | 2,475 | 1,940 | 9 | 7 |
| **Modified** | 1,402 | 1,192 | 6 | 7 |

### 3.3 System Overview

A system overview of the PULPino microcontroller is given in Fig. 1. The additional accelerators are colored blue. It shows the tightly-coupled Keccak accelerator integrated into the pipeline and directly connected to the processor's GPR and FPR. The NTT accelerator is connected to the system bus and can be reached via its corresponding address space. Furthermore, it is connected to the two additional memories for storing the twiddle factors and the polynomial coefficients. The

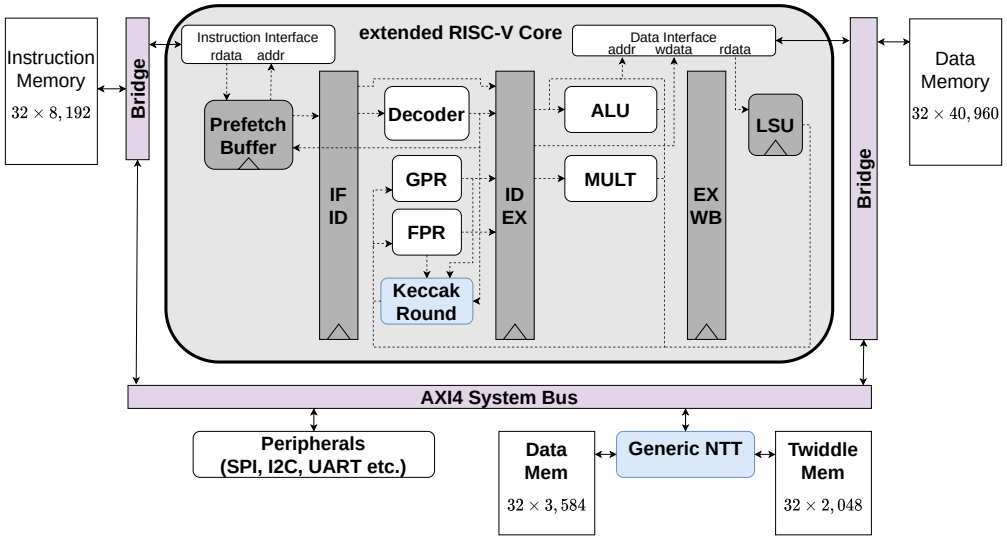



Fig. 1. System overview of the PULPino microcontroller and integrated accelerators in blue.

system's instruction memory is configured to 32 kbit and the data memory is set to store 160 kbit. As discussed later in Section 4.3.1, this is sufficient memory for both Dilithium and Falcon and also compares to a widely used commercial microcontroller.

# 4 RESULTS

## 4.1 Performance Gain

Table 4 compares our baseline implementation with the accelerated version and the PQM4 benchmark [19]. The functions have been measured over 100 iterations for a 59 B message, just as in PQM4. For the PQM4 numbers, the *clean* version has been chosen as it has the same code base as our baseline implementation, i.e. the code from PQClean[3] and therefore allows for a fair comparison. Unfortunately, PQM4 does not state results for Dilithium-5. Comparing our baseline and accelerated implementations, we obtain speed-up factors of up to 6.31 for Dilithium. For the Falcon signature verification, this factor is in the range of 2.6 to 2.7. This decreased speed-up is caused by the reduced computational complexity of the verification. As there is less to compute, the speed-up also decreases. Compared to the PQM4 numbers, the presented design is also faster, although the PULPino platform itself is less performant in general, as can be seen by comparing our baseline implementation with the PQM4 numbers.

Table 4. Average cycle count for 100 iterations and a 59 B message.

| | | Keygen | Sign | Verify |
|---|---|---|---|---|
| Dilithium-II | [19] (clean) | 1,976,311 (×3.33) | 7,465,108 (×3.92) | 2,109,292 (×3.24) |
| | baseline | 3,566,442 (×6.01) | 11,242,911 (×5.90) | 3,854,303 (×5.92) |
| | **accelerated** | 593,403 (×1.00) | 1,905,872 (×1.00) | 651,217 (×1.00) |
| Dilithium-III | [19] (clean) | 3,414,513 (×3.20) | 11,722,059 (×3.60) | 3,499,388 (×3.11) |
| | baseline | 6,432,671 (×6.02) | 20,523,503 (×6.31) | 6,458,078 (×5.73) |
| | **accelerated** | 1,067,824 (×1.00) | 3,253,378 (×1.00) | 1,126,938 (×1.00) |
| Dilithium-V | baseline | 10,679,856 (×5.98) | 25,912,136 (×5.95) | 11,016,121 (×5.96) |
| | **accelerated** | 1,784,767 (×1.00) | 4,357,249 (×1.00) | 1,848,324 (×1.00) |
| Falcon-512 | [19] (clean) | – | – | 765,394 (×2.43) |
| | baseline | – | – | 830,597 (×2.64) |
| | **accelerated** | – | – | 314,639 (×1.00) |
| Falcon-1024 | [19] (clean) | – | – | 1,526,901 (×2.49) |
| | baseline | – | – | 1,660,838 (×2.71) |
| | **accelerated** | – | – | 613,911 (×1.00) |

## 4.2 Comparison to Previous Work

Several works implemented Dilithium with the first- and second-round parameter sets of the NIST competition targeting embedded systems. Pure software implementations for embedded systems were presented in [15, 16, 26], hardware implementations in [3, 4, 27, 29, 30] as well as a HW/SW codesign in [32]. For Falcon, the verification procedure of first- and second-round parameters has been implemented in [29].

With the start of third round of the NIST competition, there has been a change in the parameter sets for Dilithium as well as for Falcon. Therefore, we want to compare our results with the most recent works that implemented the parameter sets of the round three NIST competition. As our focus is on constrained devices for embedded systems, we omit the comparison with high performance processors. Table 5 provides a cycle count comparison of our design and several previous round

[3]https://github.com/PQClean/PQClean, as of 2021-12-17

three implementations. Optimized versions sometimes divide the signing procedures of Dilithium into offline and online stages, as the expansion of $\boldsymbol{A}$ can be pre-computed if a static key is assumed. For a better comparison among all results, however, we took the numbers that perform all operations online according to the official specification document, assuming a new key for every signing process. For every algorithm, Table 5 is divided into three sections listing pure SW implementations, HW/SW codesigns and pure HW implementations.

For Dilithium, it shows that our accelerated design improves cycle counts by a factor of $\approx 2.4$ for keygen, $\approx 1.9$ for signing and $\approx 2.2$ for verification compared to the fastest pure SW implementation running on an ARM Cortex-M7, as presented in [18]. Note however, that we used the non-optimized reference C code for our evaluation whereas [18] used the C code from PQM4, that highly optimize the implementations for the Cortex-M4. The implementation in [7] also used a platform independent C code, but optimized it with respect to memory consumption. Pure hardware implementations are of course still much faster than our accelerated design, but are also less flexible. In addition to that, comparing the pure cycle count of a hardware implementation is not very meaningful without considering also the maximum frequency it can run at, as well as the resource consumption of the design. Nevertheless, we stated the cycle counts for hardware implementation as a reference and overview of current state-of-the-art. As our implementation runs software and only accelerates computationally intensive operations like the Keccak round function and polynomial arithmetic, we combine the advantages of both implementation strategies, i.e. the flexible character of SW implementations and the performance gain by HW acceleration. In [23], Nannipieri et al. presented a HW/SW codesign with tightly coupled accelerators for polynomial arithmetic on a 64 bit RISC-V platform. They obtained speed-up factors of about 2.05 for NTT transformations and even less for the whole algorithm, whereas our NTT accelerator measurements yield factors of 8.4 for NTT transformation, including the communication overhead to and from the accelerator. Furthermore, they did not accelerate the `shake128/256` operations, which are frequently used in Dilithium. As a result, our design is more than 3 times faster for signature generation.

For Falcon, our verification procedure is roughly 1.5 times faster than the best SW implementation listed in the PQM4 benchmark [19]. This moderate increase is due to the fact that Falcon's verification is inherently compact (only single hash and polynomial multiplication). The accelerators benefit comes from the computational speed-up. In this case, the computational effort is inherently small and therefore less operations can benefit from the accelerators. As expected, pure HW implementations again yield substantially higher speed-ups. We want to note, however, that comparing clock cycles for HW implementations must be taken with care without also considering their resource consumption. For completeness and a general overview, we nevertheless decided to provide their numbers in Table 5.

Table 5. Cycle count comparison with previous work for third round parameters.

| | | Platform | Keygen | Sign | Verify |
|---|---|---|---|---|---|
| **Dilithium-II** | | | | | |
| SW | [7][3] | Cortex-M4F | 2,927,000 (×4.93) | 18,470,000 (×9.69) | 4,036,000 (×6.20) |
| | [1] | Cortex-M4 | 1,598,000 (×2.69) | 4,083,000 (×2.14) | 1,572,000 (×2.41) |
| | [18] | Cortex-M7 | 1,437,000 (×2.42) | 3,658,000 (×1.92) | 1,429,000 (×2.19) |
| HW/SW | [23] | CVA6 SoC | 1,592,325 (×2.68) | 5,884,266 (×3.09) | 1,700,679 (×2.61) |
| | **This** | PULPino | 593,403 (×1.00) | 1,905,872 (×1.00) | 651,217 (×1.00) |
| HW | [20] | Artix-7 | 18,761 | 76,613 | 19,687 |
| | [2] | UltraScale+ | 14,183 | 30,358 | 15,044 |
| | [5, 6] | UltraScale+ | 4,875 | 29,876 | 6,582 |
| | [31] | Artix-7 | 4,172 | 31,600 | 4,422 |
| **Dilithium-III** | | | | | |
| SW | [7][3] | Cortex-M4F | 5,112,000 (×4.79) | 36,303,000 (×11.16) | 7,249,000 (×6.43) |
| | [1] | Cortex-M4 | 2,830,000 (×2.65) | 6,624,000 (×2.04) | 2,692,000 (×2.39) |
| | [18] | Cortex-M7 | 2,566,000 (×2.40) | 6,009,000 (×1.85) | 2,453,000 (×2.18) |
| HW/SW | [23] | CVA6 SoC | 2,974,897 (×2.79) | 10,211,677 (×3.14) | 2,963,936 (×2.63) |
| | **This** | PULPino | 1,067,824 (×1.00) | 3,253,378 (×1.00) | 1,126,938 (×1.00) |
| HW | [20] | Artix-7 | 33,102 | 123,218 | 32,050 |
| | [2] | UltraScale+ | 22,957 | 47,418 | 25,535 |
| | [5, 6] | UltraScale+ | 8,291 | 49,437 | 9,724 |
| | [31] | Artix-7 | 5,851 | 49,496 | 6,181 |
| **Dilithium-V** | | | | | |
| SW | [7][3] | Cortex-M4F | 8,609,000 (×4.82) | 44,332,000 (×10.17) | 12,616,000 (×6.83) |
| | [1] | Cortex-M4 | 4,828,000 (×2.71) | 8,726,000 (×2.00) | 4,707,000 (×2.55) |
| | [18] | Cortex-M7 | 4,368,000 (×2.45) | 8,157,000 (×1.87) | 4,287,000 (×2.32) |
| HW/SW | [23] | CVA6 SoC | 5,001,302 (×2.80) | 13,339,255 (×3.06) | 5,132,776 (×2.78) |
| | **This** | PULPino | 1,784,767 (×1.00) | 4,357,249 (×1.00) | 1,848,324 (×1.00) |
| HW | [17][4] | Artix-7 | 63,200 | 113,900 | 67,900 |
| | [20] | Artix-7 | 50,982 | 145,912 | 52,712 |
| | [2] | UltraScale+ | 38,841 | 68,460 | 45,789 |
| | [5, 6] | UltraScale+ | 14,037 | 55,070 | 13,642 |
| | [31] | Artix-7 | 8,765 | 55,321 | 9,039 |
| **Falcon-512** | | | | | |
| SW | [18] | Cortex-M7 | – | – | 559,000 (×1.78) |
| | [24] | Cortex-M4F | – | – | 530,900 (×1.69) |
| | [25] | Cortex-M4 | – | – | 504,051 (×1.60) |
| | [19][1] | Cortex-M4 | – | – | 473,061 (×1.50) |
| HW/SW | **This** | PULPino | – | – | 314,639 (×1.00) |
| HW | [6] | UltraScale+ | – | – | 2,399 |
| **Falcon-1024** | | | | | |
| SW | [18] | Cortex-M7 | – | – | 1,136,000 (×1.85) |
| | [24] | Cortex-M4F | – | – | 1,046,700 (×1.70) |
| | [25] | Cortex-M4 | – | – | 1,032,261 (×1.68) |
| | [19][2] | Cortex-M4 | – | – | 977,058 (×1.59) |
| HW/SW | **This** | PULPino | – | – | 613,911 (×1.00) |
| HW | [6] | UltraScale+ | – | – | 4,687 |

[1] Version denoted as m4-ct
[2] Version denoted as opt-leak
[3] Platform independent C code optimized for low memory consumption
[4] Only numbers for best case given. Design also presented on UltraScale+ and TSCM 65nm ASIC

### 4.3 Resource Consumption

*4.3.1 Memory Requirements.* In Table 6 the memory requirements of the baseline and accelerated versions of the algorithms are listed. For Dilithium, the memory consumption refers to the peak value of all three operations, i.e. key generation, signature generation and signature verification. On the contrary for Falcon, the consumption only indicates the requirements for the signature verification. As one can see, the required instruction size (code size) generally decreases as numerous operations implemented in software in the reference implementation are offloaded to the hardware accelerators and called by single custom instructions. This results in a code size decrease of $\approx 25\%$ for all three Dilithium security levels and $\approx 4\%$ for the Falcon implementations. The reduction in required data memory on the other hand is rather moderate for all three Dilithium parameter sets, as the total amount of computational results does not change due to the used accelerators. The small decrease is mainly caused by data representations in the NTT domain which are only stored in the NTT accelerator, thus not contributing to the core's data memory consumption. Concerning Falcon, the reduction in data memory requirement is more significant ($\approx 25\%$) as the signature verification primarily consists of one calculation in NTT domain and the corresponding transformations, as discussed in Section 2.3. As the data in NTT domain does not need to be stored within the data memory when using the NTT accelerator, the resulting reduction in memory consumption is more significant. These numbers, as shown in Table 6, define the requirements for our implementation and follows: The core has 32 KiB instruction memory and 160 KiB of data memory, while the NTT accelerator has 3584 B data and 2048 B twiddle factor memory. The total memory requirement of the core (excluding the NTT accelerator) is equal to the memory required by the popular STM32F407VG microcontroller[4].

Taking the entire PULPino microcontroller and not only RISC-V core into account, the reductions in data memory are partially canceled out by the corresponding overhead in memory necessary for the loosely-coupled NTT accelerator. This overhead can be summed up to 5632 B.

Concurrent to this work, a memory optimized implementation of Dilithium has been published [7]. Sacrificing performance as shown in Table 5, they achieved significant memory reduction by analyzing the lifetime of different variables. They targeted implementations with $< 7$ KiB (data) memory consumption and managed to run almost all Dilithium parameter sets and functions under this restriction on their ARM Cortex-M4F platform. Only the Dilithium-V signature generation required slightly more memory.

*4.3.2 Resource Consumption on an FPGA.* The resource consumption of our design synthesized for a Xilinx UltraScale+ FPGA is shown in Table 7. The number of LUTs and FFs increase by a factor of $\approx 1.48$ and $\approx 1.33$. The additional BRAM and DSP consumption is solely caused by the standalone NTT accelerator as stated in Table 3. One may notice that the sum of the overheads stated in Tables 2 and 3 is significantly smaller than the overhead of the assembled design stated in Table 7. In fact, there is an additional overhead of roughly 1k LUTs and FFs each when integrating the accelerators. This is mainly caused by the increased complexity of the AXI4-Interconnect to which the NTT accelerator is connected. Nevertheless, we see this as a moderate overhead considering the speed-up shown in Table 4.

Table 7 also compares our design with the tightly-coupled Dilithium design of [23]. It shows that their design comes with a smaller absolute overhead caused by the acceleration in terms of LUTs and FFs. This is expected as they 1) only implement small, tightly-coupled functions inside their ALU and 2) our design does not only accelerate the NTT operations and polynomial arithmetic, but also the Keccak-based functions `shake128/256`, which also require the whole FPR for storing

[4]https://www.st.com/resource/en/data_brief/stm32f4discovery.pdf

Table 6. Memory Consumption of the different algorithms for a 59 B message.

| | | Instr. Mem. | Data Mem. | NTT Data | NTT Twiddle |
|---|---|---|---|---|---|
| Dilithium-II | baseline | 27,500 | 62,560 | – | – |
| | **accelerated** | 20,624 | 61,216 | 2,048 | 512 |
| Dilithium-III | baseline | 26,780 | 94,064 | – | – |
| | **accelerated** | 20,052 | 92,720 | 2,560 | 512 |
| Dilithium-V | baseline | 27,204 | 141,168 | – | – |
| | **accelerated** | 20,324 | 139,840 | 3,584 | 512 |
| Falcon-512 | baseline | 7,864 | 10,604 | – | – |
| | **accelerated** | 7,556 | 6,508 | 1,024 | 1,024 |
| Falcon-1024 | baseline | 7,864 | 16,212 | – | – |
| | **accelerated** | 7,560 | 12,116 | 2,048 | 2,048 |

the state. Furthermore, the additional overhead caused by the interconnect as mentioned above does not apply to [23], as their design does not connect accelerators to the system bus.

Table 7. Resource consumption for baseline and modified PULPino on a Xilinx UltraScale+ FPGA (xczu9eg-ffvb1156-2e) and comparison with CV6A-based SoC [23].

| **Version** | **LUTs** | **FFs** | **BRAMs** | **DSPs** |
|---|---|---|---|---|
| PULPino base | 15,137 | 9,943 | 48 | 6 |
| **PULPino accel.** | 22,356 | 13,181 | 54 | 13 |
| Overhead | 7,219 | 3,238 | 6 | 7 |
| SoC base [23] | 61,349 | 60,278 | 77 | 19 |
| SoC accel. [23] | 64,855 | 60,349 | 77.5 | 29 |
| Overhead | 3,506 | 71 | 0.5 | 10 |

### 4.4 ASIC Design

Besides the evaluation on a FPGA platform, we also implemented the accelerated design as an ASIC. For that, we used the Globalfoundries 22 nm FDSOI technology, which enables high performance designs with still reasonable low energy consumption. As the main goal of this work was to implement a high performance design, we primarily used the fastest cells with the lowest threshold voltage available for this technology node. Nevertheless, the actual choice whether to use a faster cell with higher leakage or a slower cell with less leakage was left to the design tools as long as the constraints were met. The maximum frequency of the design is 800 MHz and mainly constrained by the access latency of the implemented memory macros. This design leads to Dilithium-II signature generations within an average of 2.38 ms and Falcon-512 signature verifications within 0.393 ms.

*4.4.1 Area overhead.* By including the loosely-coupled NTT as well as the tightly-coupled Keccak accelerator, the number of gates increased by 38%, i.e. from 52,788 to 85,778, when comparing to the original PULPino design without accelerators. Numbers for logical synthesis as well as for the placed and routed design are shown in Table 8. While this increase in gates is significant, its impact on the actual area of the design is limited, as the total area consumption is dominated by

the required instruction and data memories. All in all, our hardware acceleration results in an area increase of 23 715 µm$^2$ for the logic gates, plus an additional 66 362 µm$^2$ for the memories of the NTT accelerator, summing up to an increase in area requirement of 21%. As one can see in Table 8, this increase is mainly caused by the necessary memories for the NTT (i.e. column Cell Area Memory), which contribute 74% of the total area overhead.

Table 8. Area and gate footprints of the design for Globalfoundries' 22 nm FDSOI node

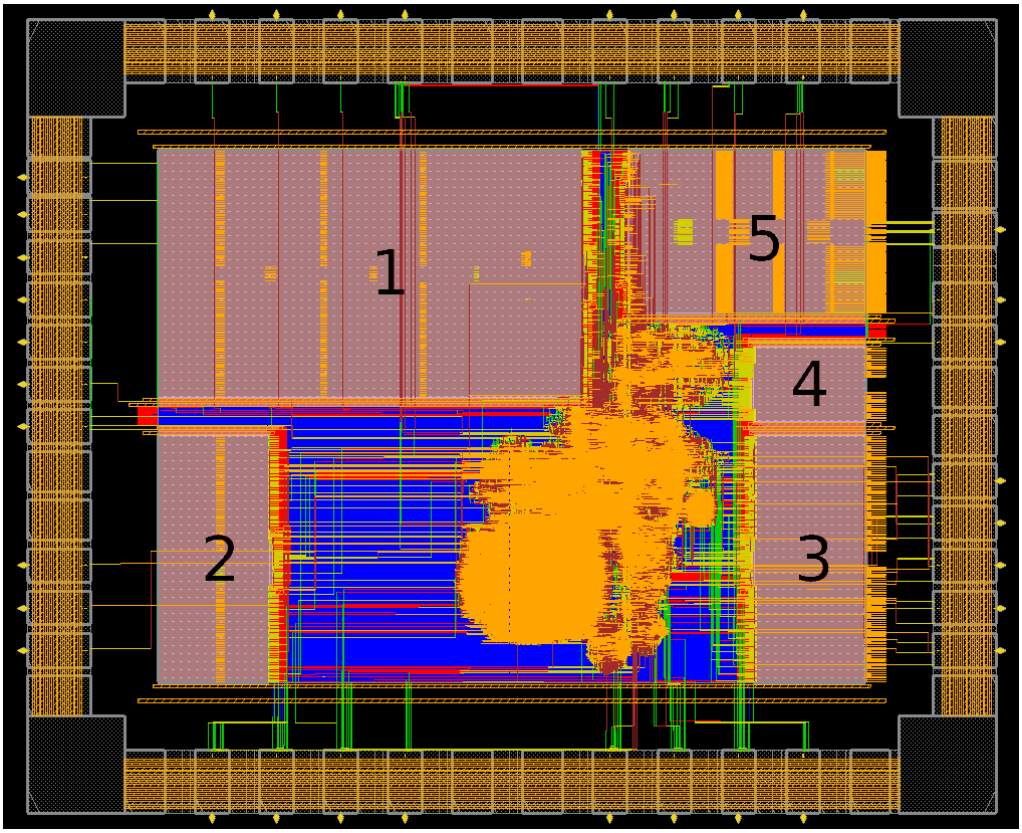
| | Cell Count | Cell Area Combinatorial [µm$^2$] | Cell Area Sequential [µm$^2$] | Cell Area Memory [µm$^2$] |
|---|---|---|---|---|
| **Logical Synthesis** | | | | |
| PULPino baseline | 43, 984 | - | - | - |
| PULPino accelerated | 65, 968 | - | - | - |
| **Post Place & Route** | | | | |
| PULPino baseline | 52, 788 | 120, 393 | 22, 883 | 223, 451 |
| PULPino accelerated | 85, 778 | 133, 058 | 33, 933 | 289, 813 |

The final layout of the design after place and route is depicted in Fig. 2. The top metal layers, which are primarily used for power routing, are omitted from the image for clarity. The blocks marked with numbers are the different memories used in the design. Number 1 and 2 are the data/system memories of the actual RISC-V core. They are split into two blocks to fit into a more compact design. Memory 3 is the instruction memory of the RISC-V core storing the program code. Memory 4 is the NTT's twiddle factor memory and 5 is the NTT's data/coefficient memory. The overall size of the design is 1144 µm x 1144 µm for the core and the IO-ring. For a tapeout, the design including additional margins for manufacturing, yields a size of 1250 µm x 1250 µm. As one can see, the total size of the ASIC as well as the shape is primarily defined by the size of the memories which prohibit further shrinking, although some space for logic would still be available.

*4.4.2 Power and energy consumption.* We furthermore compare the power and energy consumption of the original core with our accelerated design using toggle count analysis. The toggle counts for average runs were generated in simulation with Cadence Xcelium and further analyzed with Cadence Joules. The operating frequency for these simulated measurements is 800 MHz at an operating temperature of 25 °C and a core voltage of 0.8 V. The power and energy consumption of the original as well as the accelerated core are shown in Table 9. As for the aforementioned evaluation metrics, we calculated the power and energy consumption for the complete Dilithium scheme, i.e. key generation, signing and signature verification, while for Falcon, we only evaluated the signature verification. As can be seen, the energy consumption for the complete scheme is reduced by more than a factor of 10 for all three Dilithium sets, which corresponds to savings of $\approx$ 91% up to $\approx$ 92.9%. The energy reduction is less significant for the Falcon parameters sets, i.e. 57% and 67.5% as we only evaluated the signature verification. In this case, there are only few operations benefiting from the acceleration and the static leakage dominates the total power consumption. The work in [23] also measured the savings of energy consumption for their optimizations. When combining all the savings for key generation, signing and verification, they end up with total savings of $\approx$ 27.4%[5] in total for Dilithium-II, which is much less. However, a direct comparison is quite difficult, as they implemented their design not on ASIC technology, but on a Xilinx ZCU106

[5]Computed by dividing the sum of energy for the accelerated case by the sum of energy in the baseline case.

Fig. 2. ASIC layout of the presented design after place and route implemented using the Globalfoundries 22 nm FDSOI technology.




FPGA board running at a clock frequency of 100 MHz. With respect to gate optimization and routing capabilities, FPGAs are of course less flexible.

In [3], a flexible co-processor for a variety of lattice-based schemes, called Sapphire, was presented. The design was also evaluated for Dilithium and implemented using a 40nm TSMC low-power technology. For their power measurements, the design was running at a frequency of 72 MHz. A comparison of latency, power and energy consumption for Dilithium running on our design as well as Sapphire is shown in Fig. 3. For comparison, we used combined values for one key generation, signature generation and signature verification. That is, the energy values given in [3] for all three functions were summed-up. Similarly for the latency, the added cycle counts and corresponding frequency of 72 MHz were taken to compute the time in ms. Both values were then used to compute an average estimate for power consumption, which is roughly in the range of the values given in [3]. Note, however, that Sapphire was evaluated for the second round parameter sets of the NIST competition, which slightly differ from the third round parameter sets we evaluated. In Fig. 4a, the total latency for the different parameter sets is compared. Sapphire requires less clock cycles but also runs much slower (72 MHz compared to 800 MHz), such that our implementation is faster w.r.t. time when considering the different security levels. Although our design has slightly higher power consumption as shown in Fig. 4b, we again require less total energy as shown in Fig. 4c. That is

Table 9. Power and Energy of the Design for Globalfoundries' 22 nm FDSOI node for 800 MHz.

| | | **Leakage** [µW] | **Internal** [µW] | **Switching** [µW] | **Tot. Power** [µW] | **Cycles** [$\times 10^3$] | **Energy** [µJ] |
|---|---|---|---|---|---|---|---|
| Dilithium-II | base | 994 | 7,264 | 6,536 | 14,794 | 19,036 | 352 (×13.2) |
| | accel. | 1,116 | 5,515 | 312 | 6,943 | 3,068 | 26.6 (×1.00) |
| Dilithium-III | base | 1031 | 12,055 | 3,674 | 16,759 | 33,259 | 697 (×11.2) |
| | accel. | 1,116 | 6,580 | 382 | 8,078 | 6,144 | 62.0 (×1.00) |
| Dilithium-V | base | 1,031 | 14,472 | 3,988 | 19,490 | 47,713 | 1162 (×14.1) |
| | accel. | 1,116 | 6,698 | 289 | 8,103 | 8,150 | 82.5 (×1.00) |
| Falcon-512 | base | 992 | 591 | 683 | 2,267 | 831 | 2.35 (×2.33) |
| | accel. | 1,113 | 1,403 | 53 | 2,569 | 315 | 1.01 (×1.00) |
| Falcon-1024 | base | 992 | 1,083 | 1,221 | 3,927 | 1,661 | 8.15 (×3.08) |
| | accel. | 1,137 | 2,247 | 89 | 3,450 | 614 | 2.65 (×1.00) |

mainly caused by the fact that our design is faster while consuming roughly the same amount of power.

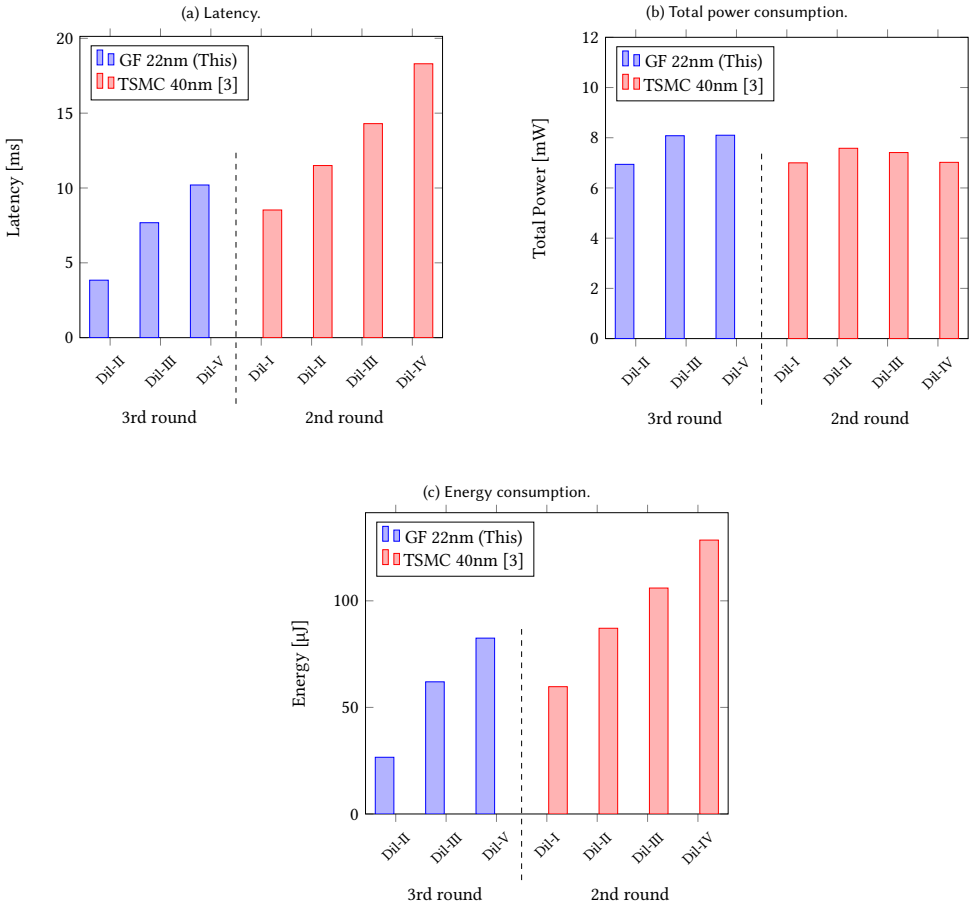
## 5 CONCLUSION

Large-scale quantum computers pose a threat to cryptographic systems as they can efficiently solve fundamental mathematical problems in cryptography. Therefore, devices must be prepared to provide means of secure communication and switch to algorithms that are considered secure in these scenarios. As the migration towards post-quantum secure systems will take time and the security of corresponding schemes must be further analyzed, agile cryptosystems will play an important role during migration. In this work, we presented a flexible, hardware accelerated RISC-V design for lattice-based digital signature algorithms. Our design accelerates the full Dilithium scheme as well as the verification of Falcon signatures. This allows for secure firmware updates in case security concerns for one of the schemes arise. Computational intensive operations in lattice-based cryptography are usually the generation of uniformly distributed polynomials and polynomial arithmetic. Therefore, we have shown how using accelerators, that were previously used for acceleration of PKEs and KEMs for these computational intensive operations improve the performance of signature schemes. In fact, our design is faster than optimized embedded software implementations or previous HW/SW codesigns. We have further shown how the code size decreases for the signature schemes by using hardware accelerators. Using a 22nm Globalfoundries technology, we presented an ASIC design of our accelerated system that can run at 800 MHz. This leads to Dilithium signature generation in only a few microseconds and Falcon signature verification in less than half of a microsecond for the lowest security levels. Besides the performance benefits, hardware acceleration also significantly reduces the overall power and energy consumption of the investigated signature schemes. This is an important aspect especially for resource constrained devices in the Internet-of-Things (IoT) context.

## ACKNOWLEDGMENTS

The authors acknowledge the financial support by the Federal Ministry of Education and Research of Germany in the program of "Souverän. Digital. Vernetzt.". Joint project 6G-life, project identification number: 16KIS002.

Fig. 3. Comparison of latency, power and energy comparison between this work and the TSMC 40nm design of [3].



## REFERENCES


[1] Amin Abdulrahman, Vincent Hwang, Matthias J. Kannwischer, and Daan Sprenkels. 2022. Faster Kyber and Dilithium on the Cortex-M4. Cryptology ePrint Archive, Report 2022/112. https://ia.cr/2022/112.

[2] Aikata, Ahmet Can Mert, David Jacquemin, Amitabh Das, Donald Matthews, Santosh Ghosh, and Sujoy Sinha Roy. 2021. A Unified Cryptoprocessor for Lattice-based Signature and Key-exchange. Cryptology ePrint Archive, Report 2021/1461. https://ia.cr/2021/1461.

[3] Utsav Banerjee, Tenzin S. Ukyab, and Anantha P. Chandrakasan. 2019. Sapphire: A Configurable Crypto-Processor for Post-Quantum Lattice-based Protocols. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (aug 2019), 17–61. https://doi.org/10.46586/tches.v2019.i4.17-61

[4] Kanad Basu, Deepraj Soni, Mohammed Nabeel, and Ramesh Karri. 2019. NIST Post-Quantum Cryptography - A Hardware Evaluation Study. Cryptology ePrint Archive, Report 2019/047. https://ia.cr/2019/047 https://ia.cr/2019/047.

[5] Luke Beckwith, Duc Tri Nguyen, and Kris Gaj. 2021. High-Performance Hardware Implementation of CRYSTALS-Dilithium. In *2021 International Conference on Field-Programmable Technology (ICFPT)*. IEEE. https://doi.org/10.1109/icfpt52863.2021.9609917

[6] Luke Beckwith, Duc Tri Nguyen, and Kris Gaj. 2022. High-Performance Hardware Implementation of Lattice-Based Digital Signatures. Cryptology ePrint Archive, Report 2022/217. https://ia.cr/2022/217.

[7] Joppe W. Bos, Joost Renes, and Daan Sprenkels. 2022. Dilithium for Memory Constrained Devices. Cryptology ePrint Archive, Report 2022/323. https://ia.cr/2022/323.

[8] James W. Cooley and John W. Tukey. 1965. An algorithm for the machine calculation of complex Fourier series. *Math. Comp.* 19, 90 (1965), 297–301. https://doi.org/10.1090/s0025-5718-1965-0178586-1

[9] Morris J. Dworkin. 2015. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions.* Technical Report. https://doi.org/10.6028/nist.fips.202

[10] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. 2018. Falcon: Fast-Fourier lattice-based compact signatures over NTRU. *Submission to the NIST post-quantum cryptography standardization process* (2018). https://falcon-sign.info/falcon.pdf

[11] Tim Fritzmann, Michiel Van Beirendonck, Debapriya Basu Roy, Patrick Karl, Thomas Schamberger, Ingrid Verbauwhede, and Georg Sigl. 2021. Masked Accelerators and Instruction Set Extensions for Post-Quantum Cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (nov 2021), 414–460. https://doi.org/10.46586/tches.v2022.i1.414-460

[12] Tim Fritzmann, Georg Sigl, and Johanna Sepúlveda. 2020. RISQ-V: Tightly Coupled RISC-V Accelerators for Post-Quantum Cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems* Volume 2020 (2020), Issue 4. https://doi.org/10.13154/TCHES.V2020.I4.239-280

[13] Michael Gautschi, Pasquale Davide Schiavone, Andreas Traber, Igor Loi, Antonio Pullini, Davide Rossi, Eric Flamand, Frank K. Gurkaynak, and Luca Benini. 2017. Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25, 10 (oct 2017), 2700–2713. https://doi.org/10.1109/tvlsi.2017.2654506

[14] W. M. Gentleman and G. Sande. 1966. Fast Fourier Transforms. In *Proceedings of the November 7-10, 1966, fall joint computer conference on XX - AFIPS '66 (Fall)*. ACM Press. https://doi.org/10.1145/1464291.1464352

[15] Denisa O. C. Greconici, Matthias J. Kannwischer, and Daan Sprenkels. 2020. Compact Dilithium Implementations on Cortex-M3 and Cortex-M4. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (dec 2020), 1–24. https://doi.org/10.46586/tches.v2021.i1.1-24

[16] Tim Güneysu, Markus Krausz, Tobias Oder, and Julian Speith. 2018. Evaluation of Lattice-Based Signature Schemes in Embedded Systems. In *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE. https://doi.org/10.1109/icecs.2018.8617969

[17] Naina Gupta, Arpan Jati, Anupam Chattopadhyay, and Gautam Jha. 2022. Lightweight Hardware Accelerator for Post-Quantum Digital Signature CRYSTALS-Dilithium. Cryptology ePrint Archive, Report 2022/496. https://eprint.iacr.org/2022/496/20220428:080014 https://ia.cr/2022/496.

[18] James Howe and Bas Westerbaan. 2022. Benchmarking and Analysing the NIST PQC Finalist Lattice-Based Signature Schemes on the ARM Cortex M7. Cryptology ePrint Archive, Report 2022/405. https://eprint.iacr.org/2022/405/20220331:072640 https://ia.cr/2022/405.

[19] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. [n.d.]. PQM4: Post-quantum crypto library for the ARM Cortex-M4. https://github.com/mupq/pqm4, as of 2022-01-19.

[20] Georg Land, Pascal Sasdrich, and Tim Güneysu. 2021. A Hard Crystal - Implementing Dilithium on Reconfigurable Hardware. Cryptology ePrint Archive, Report 2021/355. https://ia.cr/2021/355.

[21] Vadim Lyubashevsky. 2009. Fiat-Shamir with Aborts: Applications to Lattice and Factoring-Based Signatures. In *Advances in Cryptology – ASIACRYPT 2009*. Springer Berlin Heidelberg, 598–616. https://doi.org/10.1007/978-3-642-

10366-7_35

[22] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. 2017. CRYSTALS-Dilithium. *Submission to the NIST post-quantum cryptography standardization process* (2017). https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf

[23] Pietro Nannipieri, Stefano Di Matteo, Luca Zulberti, Francesco Albicocchi, Sergio Saponara, and Luca Fanucci. 2021. A RISC-V Post Quantum Cryptography Instruction Set Extension for Number Theoretic Transform to Speed-Up CRYSTALS Algorithms. *IEEE Access* 9 (2021), 150798–150808. https://doi.org/10.1109/access.2021.3126208

[24] Tobias Oder, Julian Speith, Kira Höltgen, and Tim Güneysu. 2019. Towards Practical Microcontroller Implementation of the Signature Scheme Falcon. In *Post-Quantum Cryptography*. Springer International Publishing, 65–80. https://doi.org/10.1007/978-3-030-25510-7_4

[25] Thomas Pornin. 2019. New Efficient, Constant-Time Implementations of Falcon. Cryptology ePrint Archive, Report 2019/893. https://eprint.iacr.org/2019/893/20190918:144441

[26] Prasanna Ravi, Sourav Sen Gupta, Anupam Chattopadhyay, and Shivam Bhasin. 2019. Improving Speed of Dilithium's Signing Procedure. Cryptology ePrint Archive, Report 2019/420. https://eprint.iacr.org/2019/420/20191018:013921 https://ia.cr/2019/420.

[27] Sara Ricci, Lukas Malina, Petr Jedlicka, David Smekal, Jan Hajny, Petr Cibik, and Patrik Dobias. 2021. Implementing CRYSTALS-Dilithium Signature Scheme on FPGAs. Cryptology ePrint Archive, Report 2021/108. https://ia.cr/2021/108.

[28] P.W. Shor. 1994. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*. IEEE Comput. Soc. Press, 124–134. https://doi.org/10.1109/sfcs.1994.365700

[29] Deepraj Soni, Kanad Basu, Mohammed Nabeel, Najwa Aaraj, Marc Manzano, and Ramesh Karri. 2021. *Hardware Architectures for Post-Quantum Digital Signature Schemes*. Springer International Publishing. https://doi.org/10.1007/978-3-030-57682-0

[30] Deepraj Soni, Kanad Basu, Mohammed Nabeel, and Ramesh Karri. 2019. A hardware evaluation study of nist post-quantum cryptographic signature schemes. In *Second PQC Standardization Conference*. NIST. https://csrc.nist.gov/Events/2019/second-pqc-standardization-conference

[31] Cankun Zhao, Neng Zhang, Hanning Wang, Bohan Yang, Wenping Zhu, Zhengdong Li, Min Zhu, Shouyi Yin, Shaojun Wei, and Leibo Liu. 2021. A Compact and High-Performance Hardware Architecture for CRYSTALS-Dilithium. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (nov 2021), 270–295. https://doi.org/10.46586/tches.v2022.i1.270-295

[32] Zhen Zhou, Debiao He, Zhe Liu, Min Luo, and Kim-Kwang Raymond Choo. 2021. A Software/Hardware Co-Design of Crystals-Dilithium Signature Scheme. *ACM Transactions on Reconfigurable Technology and Systems* 14, 2 (jun 2021), 1–21. https://doi.org/10.1145/3447812