A Monolithic Hardware Implementation of Kyber: Comparing Apples to Apples in PQC Candidates

Mojtaba Bisheh-Niasar[1(✉)], Reza Azarderakhsh[1,2], and Mehran Mozaffari-Kermani[3]

[1] Department of Computer and Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton, FL, USA
{mbishehniasa2019,razarderakhsh}@fau.edu
[2] PQSecure Technologies, LLC, Boca Raton, FL, USA
[3] Department of Computer Science and Engineering, University of South Florida, Tampa, FL, USA
mehran2@usf.edu

Abstract. With the advent of large-scale quantum computers, factoring and discrete logarithm problems could be solved using the polynomial-time quantum algorithms. To ensure public-key security, a transition to quantum-resistant cryptographic protocols is required. Performance of hardware accelerators targeting different platforms and diverse application goals plays an important role in PQC candidates' differentiation. Hardware accelerators based on FPGAs and ASICs also provide higher flexibility to create a very low area or ultra-high performance implementations at the high cost of the other. While the hardware/software co-design development of PQC schemes has already received an increasing research effort, a cost analysis of efficient pure hardware implementation is still lacking. On the other hand, since FPGA has various types of hardware resources, evaluating and making the accurate and fair comparison of hardware-based implementations against each other is very challenging. Without a common foundation, apples are compared to oranges. This paper demonstrates a pure hardware architecture for Kyber as one of the finalists in the third round of the NIST post-quantum cryptography standardization process. To enable real, realistic, and comparable evaluations in PQC schemes over hardware platforms, we compare our architecture over the ASIC platform as a common foundation showing that it outperforms the previous works in the literature.

Keywords: ASIC · Hardware architecture · Kyber · Lattice-based cryptography · NTT · Post-quantum cryptography

1 Introduction

The hard problems of traditional public-key cryptosystems, e.g., RSA and ECC, can be easily solved using Shor's algorithm [1], so current cryptographic algorithms cannot be secure anymore against quantum attacks. To prepare for

P. Longa and C. Ràfols (Eds.): LATINCRYPT 2021, LNCS 12912, pp. 108–126, 2021.
https://doi.org/10.1007/978-3-030-88238-9_6

security concerns caused by building large-scale quantum computers, in 2016, the National Institute of Standards and Technology (NIST) started the post-quantum cryptography (PQC) standardization process for the quantum-safe cryptographic algorithm. After several rounds, NIST announced finalist candidates in July 2020, including four key encapsulation mechanisms (KEM), i.e., Classic-McEliece, Kyber, NTRU, and Saber. The majority of the finalists are based on lattice-based cryptography offering a high-performance scheme and relatively small ciphertext and key sizes. Kyber KEM [2] is one of the PQC finalists, which is constructed on the hardness of the module learning-with-errors problem (M-LWE) in module lattices [3].

Performance of hardware accelerators plays an important role in the NIST standardization process because the overall complexity of the winner schemes will have to be minimal to be implemented in widely-deployed cryptosystems [4]. As a consequence, hardware benchmarking of PQC candidates is crucial considering the advantages of hardware-based designs to exploit parallelism, which leads to improvements in the efficiency of the overall system. While software (SW) implementations for embedded systems have more flexibility than hardware-based approaches, they have a lower performance. Hardware/Software (HW/SW) co-design approaches increase the performance but keeping the flexibility of a SW solution to cope with embedded constraints. Although the HW/SW approach offers flexibility and a shorter design cycle than pure HW schemes, they may not lead to the best performance. Simultaneously, pure hardware implementation of PQC schemes is extremely challenging due to their high algorithmic complexity, considering both algorithmic and architectural alternatives, and also the lack of hardware description language libraries for the basic building blocks.

An accurate and fair comparison between hardware accelerators is very challenging. One of the main challenges is they target different optimization perspectives, including performance, required resources, power consumption, and energy usage. To address this challenge, one can consider efficiency as an area-time product for a comprehensive comparison. Additionally, another challenge for a fair comparison is that most HW/SW designs proposed a unified core for several schemes [5–7]. Therefore, there appear to be very few hardware implementations that focus only on a specific scheme and make the best of all its features. Moreover, the comparisons are too complicated and cannot indicate the advantage of one architecture over another, especially when they do not belong to the same platform. NIST's recommendation to use the Xilinx Artix-7 FPGA family for hardware prototyping is in an effort to improve the accuracy of comparison in the same chip architecture family. Nevertheless, the effect of different resources, e.g., DSPs and BRAMs, has not been taken into consideration in the calculation of the total area. Consequently, ASIC results can be chosen as a benchmark to have a fair comparison with existing implementations with respect to efficiency.

In this paper, we propose a monolithic hardware implementation, including polynomial sampling, NTT, and point-wise multiplication, that is parallelized by virtue of the Kyber algorithm that is naturally parallelizable to accelerate

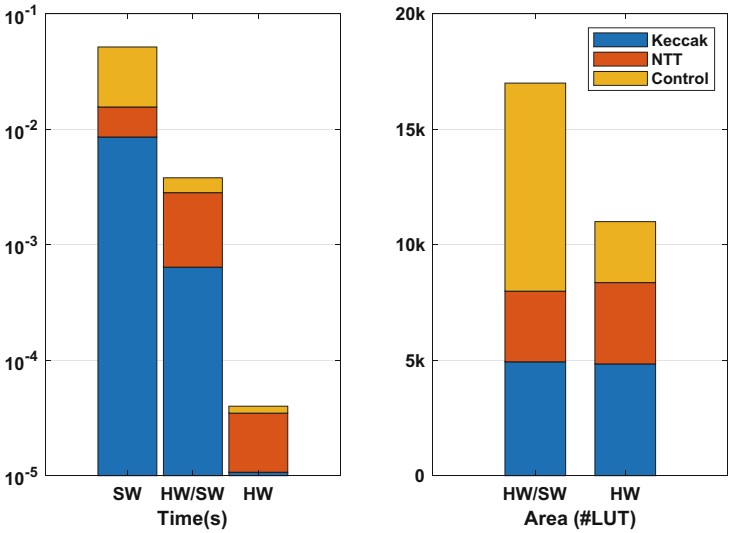

Fig. 1. Performance (in $\log_{10}$) and resource utilization comparison in three different Kyber-512 implementation approaches: software (SW), hardware/software (HW/SW), and hardware (HW). Kyber architecture is breakdown into three main cores, including Keccak (hashing and sampling), NTT (polynomial multiplication), and Control (controller and all other required functions).

lattice-based PQC exploiting fewer resources. The efficiency of our proposed PQC implementation has performance levels comparable to or even significantly better than ECC-based schemes [8–10].

The first hardware implementation of Kyber was proposed in [11] employing the high-level synthesis (HLS) approach. In this work, the authors designed a map of high-level C specifications of round 2 candidates into FPGA and ASIC implementations. Although some dedicated optimizations, particularly for Kyber, are applied, the results are significantly less efficient from other hardware-based implementations. The authors in [5] presented a configurable core based on RISC-V architecture over ASIC targeting power consumption optimization. To provide FPGA results, the authors extended their work in [12]. In [6], another RISC-V accelerator called RISQ-V was introduced synthesized for an FPGA prototype and an ASIC. The authors in [7] proposed a lightweight design for NewHope and Kyber based on the RISC-V processor integrated with a finite field multiplier for FPGA. A vector processor was proposed in [13] for ASIC implementations targeting a high-performance architecture. The first pure hardware implementation of Kyber is reported in [14] based on RTL methodology in FPGA. The work of [15] also presented a pure hardware approach for Kyber. However, this design heavily relies on memory units between components. Moreover, a compact architecture was proposed in [16] to reduce the required block RAMs. In [17], the authors proposed a highly optimized NTT core to achieve

significant speedup for Kyber KEM. The hardware architecture for polynomial multiplication targeting Kyber parameters has been studied by several implementations [18–21]; however, these architectures cannot perform the complete Kyber protocols.

Figure 1 illustrates the comparison between different development approaches, i.e., SW, HW/SW, and HW implementations of Kyber KEM based on the required time and resources. The reported cycle counts for SW implementation in [22–24] show 60–80% of the overall computation time is spent on hashing and sampling. Thus, Keccak is the most performance-critical part of SW implementation. However, this core can be accelerated in a hardware architecture since Keccak is a hardware-friendly design of SHA. For HW/SW co-design approach, a wide range of results was reported for polynomial multiplication. While the work of [13] occupied 55% of the total area for vectorized NTT core, in [6], only 12% of resources were utilized for NTT. Additionally, implementing a software-based processor, e.g., RISC-V architecture, increases the occupied resources for the controller in HW/SW compared to the HW approach. For example, the controller requires 31% and 71% of total resources in [5] and [6], respectively. However, in pure HW implementation of lattice-based PQC, the controller cost was reduced to 5% [25]. Accordingly, the pure HW can significantly accelerate the Kyber KEM by parallelizing NTT and hiding the Keccak latency on the one hand, and reduces the required resources compared to HW/SW approach on the other hand.

1.1 Contributions

Polynomial multiplication computations take a significant portion of Kyber KEM latency on hardware implementation. To improve the efficiency of Kyber, one should increase efficiency on the NTT core, providing higher throughput using fewer hardware resources. This paper proposes an efficient hardware implementation of the module lattice-based post-quantum KEM CRYSTALS-Kyber on the application-specific integrated circuit (ASIC) platform. Our proposed architecture provides a monolithic hardware implementation to accelerate lattice-based PQC exploiting compact resources. The contributions of this paper are itemized in the following:

1. We propose a compact hardware architecture for NTT and INTT, supporting both decimation-in-frequency (DIF) and decimation-in-time (DIT) NTT algorithms. Our proposed reconfigurable architecture avoids utilizing additional resources for the same computations while reduces the pre-processing cost of NTT and post-processing cost of INTT. The proposed architecture significantly reduces the overall area and memory consumption with no impact on performance.
2. We highly parallelize the operations in polynomial sampling cores through tightly coupling with Keccak core to decrease the required cycles. The performance of proposed parallel scheduling for binomial sampler indicates a significant improvement, while our rejection sampler latency can be completely absorbed by the Keccak core.

3. We propose a high-performance coprocessor architecture for lattice-based public-key cryptography with Kyber KEM as a case study. Our result utilizes the proposed high-speed NTT core and outperforms all reported implementations by reducing the total time.

The rest of the paper is organized as follows. In Sect. 2, we discuss the preliminaries of lattice-based cryptography and the relevant mathematical background based on the Kyber algorithm. In Sect. 3, our proposed algorithms and architectures for implementing a high-performance Kyber KEM are discussed. We discuss our results and compare them to the counterparts in Sect. 4. Finally, we conclude the paper in Sect. 5.

2 Preliminaries

In this section, employed notation, Kyber protocols and relevant mathematical background are briefly described.

2.1 Symbol Definition

In this paper, to make the paper more readable, regular font lower-case letters (a) shows polynomials, bold lower-case letters ($\mathbf{a}$) determines vectors of polynomials, bold upper-case letters ($\mathbf{A}$) indicates matrices of polynomials, and their NTT-domain representation are referred by ($\hat{a}$), ($\hat{\mathbf{a}}$) and ($\hat{\mathbf{A}}$), respectively. For a vector $\mathbf{a}$ (or matrix $\mathbf{A}$), its transpose is $\mathbf{a}^T$ (or $\mathbf{A}^T$). Also, the lower-case Greek letters ρ, σ, and μ stand for random bit-strings. The polynomial ring $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n+1)$ is defined over the field of $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$ in which n is the dimension and q is the prime modulo. Let $\mathbf{a}$ and $\mathbf{b}$ be polynomial vectors in $\mathcal{R}_q$, we denote point-wise multiplication by $\mathbf{a} \circ \mathbf{b} \in \mathcal{R}_q$. The $\circ$ product between a matrix and a vector is the natural generalization of point-wise multiplication between their polynomial vectors.

2.2 The Kyber Protocol

Kyber is an IND-CCA secure KEM [26], including three algorithms, i.e., key generation, encryption, and decryption. In key generation, a matrix $\mathbf{A}$ and a secret key $\mathbf{s}$ are sampled from a uniform and binomial distribution, respectively. Then a public key is computed by multiplication between $\mathbf{A}$ and $\mathbf{s}$ in the NTT domain and adding noise to the product. In encryption, a message m should be added to the product of the public key and a sampled random $\mathbf{r}$ in the normal domain to generate a vector v. Additionally, another polynomial multiplication is performed between $\mathbf{r}$ and uniform distribution matrix $\hat{\mathbf{A}}$ to compute matrix $\mathbf{u}$. The encryption output, called ciphertext ct, is composed of compression of $\mathbf{u}$ and v, while the message can then be decrypted by recovering an approximation of v by computing the product of secret key and $\mathbf{u}$.

Table 1. Parameter sets for Kyber Implementation [2]

Algorithm	NIST-Level	Parameters					Size (in Bytes)		
		n	k	q	(η_1, η_2)	(d_u, d_v)	Secret key (sk)	Public key (pk)	Ciphertext (ct)
Kyber-512	1 (AES-128)	256	2	3,329	(3, 2)	(10, 3)	1,632	800	768
Kyber-768	3 (AES-192)	256	3	3,329	(2, 2)	(10, 4)	2,400	1,184	1,088
Kyber-1024	5 (AES-256)	256	4	3,329	(2, 2)	(11, 5)	3,168	1,568	1,568

All polynomials in the Kyber scheme have 256 coefficients over k-dimensional vectors, where $k = 2, 3, 4$ indicates the three different post-quantum security levels. Kyber parameter sets corresponding to these levels are reported in Table 1 to construct a Chosen Plaintext Attack (CPA) secure public-key encryption scheme. Moreover, a CCA-secure Kyber KEM can be constructed using an adapted Fujisaki-Okamoto transformation [27]. For details, we refer interested readers to [2].

2.3 Polynomial Multiplication

The most important operation in lattice-based cryptography is polynomial multiplication, which can be performed using different methods, e.g., NTT or schoolbook polynomial multiplication algorithm. Polynomial multiplication in NTT domain can be computed efficiently over a polynomial ring $\mathbb{Z}_q[X]/\langle X^n + 1\rangle$ when the modulus provides m-th primitive roots of unity for a sufficiently high power of two m (ideally, $m = 2n$ or $m = n$). The NTT is defined as a fast Fourier transform (FFT) in a finite field. Let f be a polynomial of degree n, where $f = \sum_{i=0}^{n-1} f_i X^i$ and $f_i \in \mathbb{Z}_q$, and ω_n be n-th primitive root of unity such that $\omega_n^n = 1 \bmod q$. The forward NTT is defined by $\hat{f} = NTT(f)$, such that:

$$\hat{f}_i = \sum_{j=0}^{n-1} f_j \omega_n^{ij} \bmod q \tag{1}$$

The inverse NTT, shown by INTT, as a back transformation form NTT domain to normal domain is shown by $f = INTT(\hat{f})$, such that:

$$f_i = n^{-1} \sum_{j=0}^{n-1} \hat{f}_j \omega_n^{-ij} \bmod q \tag{2}$$

Accordingly, a polynomial multiplication between polynomial vectors f and g employing NTT and INTT results in a polynomial vector which can be performed such that:

$$f \cdot g = \mathrm{INTT}(\mathrm{NTT}(f) \circ \mathrm{NTT}(g)) \tag{3}$$

To avoid the overhead of zero padding in the polynomial multiplication over $\mathbb{Z}_q[X]/\langle X^n + 1\rangle$, the negative wrapped convolution (NWC) was proposed in

[28] at the cost of pre-processing of NTT and post-processing of INTT. Let $\psi = \sqrt{\omega_n}$ be a primitive $2n$-th root of unity. Pre-processing of NTT includes multiplication between the coefficients of the input polynomials and ψ^i, while the post-processing of INTT is multiplication between the coefficients of the output polynomial and ψ^{-i}.

However, the work of [29] merged the pre-processing of NTT into butterfly operations. The work of [30] presented an algorithm to avoid the post-processing overhead of INTT. The KRED reduction algorithm was proposed in [31] to accelerate the NTT and reduce the post-processing overhead of INTT. NTT computation can be implemented by Cooley-Turkey (CT) [32] or Gentleman-Sande (GS) [33] butterfly configuration. Employing the CT as NTT and the GS as INTT [31,34] is a well-known trick in the literature to avoid the bit-reverse permutation.

Algorithm 1 presents the NTT computation. Figure 2 illustrates an 8-point NTT-based multiplication employing both CT and GS butterfly operations. The matrix-vector multiplication $\hat{\mathbf{A}} \circ \hat{\mathbf{s}}$ in NTT domain for Kyber-512 is shown in (4).

$$\hat{\mathbf{A}} \circ \hat{\mathbf{s}} = \begin{bmatrix} \hat{\mathbf{A}}_{\mathbf{00}} & \hat{\mathbf{A}}_{\mathbf{01}} \\ \hat{\mathbf{A}}_{\mathbf{10}} & \hat{\mathbf{A}}_{\mathbf{11}} \end{bmatrix} \circ \begin{bmatrix} \hat{\mathbf{s}}_{\mathbf{0}} \\ \hat{\mathbf{s}}_{\mathbf{1}} \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{A}}_{\mathbf{00}} \circ \hat{\mathbf{s}}_{\mathbf{0}} + \hat{\mathbf{A}}_{\mathbf{01}} \circ \hat{\mathbf{s}}_{1} \\ \hat{\mathbf{A}}_{\mathbf{10}} \circ \hat{\mathbf{s}}_{\mathbf{0}} + \hat{\mathbf{A}}_{\mathbf{11}} \circ \hat{\mathbf{s}}_{\mathbf{1}} \end{bmatrix} \tag{4}$$

A point-wise multiplication includes 128 multiplications of polynomial of degree 2 modulo $X^2 - \zeta^{2\mathsf{br}_7(i)+1}$. For example, multiplication between two coefficients $\hat{\mathbf{A}}_{\boldsymbol{j,i}} \circ \hat{\mathbf{s}}_{\boldsymbol{i}}$ can be performed as shown in (5) where $\zeta = 17$ is the first primitive 256-th root of unity modulo q, and br_7 is the bit reversal function.

$$\begin{aligned} &(\hat{a}_{j,2i} + \hat{a}_{j,2i+1}X) \cdot (\hat{s}_{2i} + \hat{s}_{2i+1}X) \\ &= (\hat{a}_{j,2i}\hat{s}_{2i} + \hat{a}_{j,2i+1}\hat{s}_{2i+1}\zeta^{2\mathsf{br}_7(i)+1}) \\ &+ (\hat{a}_{j,2i}\hat{s}_{2i+1} + \hat{a}_{j,2i+1}\hat{s}_{2i})X \pmod{X^2 - \zeta^{2\mathsf{br}_7(i)+1}} \end{aligned} \tag{5}$$

3 Proposed Architecture

In this section, the proposed components to design a high-performance and efficient Kyber KEM are described.

3.1 Keccak Core

Keccak core is required in KEM to compute four different functions, including two hash functions SHA3-256 and SHA3-512, SHAKE-128 as an extendable output function (XOF), and SHAKE-256 as a pseudo random function (PRF) and key-derivation function (KDF). Keccak is a hardware-oriented design based on bit-oriented operations. Since we compute 24 rounds for Keccak-f[1600], various architectures can be employed considering different optimization perspectives, i.e., high-performance, lightweight, or anything in between. In our proposed high-performance core, we slightly modify the implementation of the high-speed core by the Keccak team [36] by designing a serial-in parallel-out (SIPO) buffer in

Algorithm 1. Iterative In-Place NTT Algorithm Based on Cooley-Tukey Butterfly [35]

Input: a polynomial $a(x) \in \mathbb{Z}_q[X]/(X_n + 1)$, n-th primitive root of unity $\omega_n \in \mathbb{Z}_q$, $n = 2^l$
Output: $\hat{a}(x) = \text{NTT}_{\omega_n}(a) \in \mathbb{Z}_q[X]/(X_n + 1)$

$\hat{a} \leftarrow$ bit-reverse(a)
for ($i = 1$; $i < l$; $i++$) **do**
  $m = 2^{l-i}$
  $\omega_m \leftarrow \omega_n^{n/m}$
  for ($j = 0$; $j < n$; $j = j + m$) **do**
    $\omega \leftarrow 1$
    for ($k = 0$; $k < m/2$; $k++$) **do**
      $T \leftarrow \omega \cdot \hat{a}[k + j + m/2] \bmod q$
      $U \leftarrow \hat{a}[k + j]$
      $\hat{a}[k + j] = U + T \bmod q$
      $\hat{a}[k + j + m/2] = U - T \bmod q$
      $\omega \leftarrow \omega \cdot \omega_m \bmod q$
    end for
  end for
end for
return $\hat{a}(x)$

Table 2. Failure probabilities in Kyber rejection sampling for performing different Keccak rounds

Total round	Keccak outputs (bit)	Total samples	Required valid sample	Failure probability
3	4,032	336	256	0.0083
4	5,376	448	256	2.2E−32
5	6,720	560	256	2.3E−79

input and parallel-in serial-out (PISO) buffer for accelerating the data transition with this core. The serial data width for these buffers is set to 64-bit, and the parallel line is 1344-bit. A maximum of 21 cycles is needed to read/write to/from these buffers in serial mode, while the Keccak sponge function takes 24 cycles. Therefore, data transforming can be hidden by simultaneously performing Keccak computation without resource conflict and data dependency to reduce clock cycles.

3.2 Rejection Sampling

This unit takes 64-bit data from the output of SHAKE-128 stored in a PISO buffer. The required cycles for this unit are variable due to the non-deterministic pattern of rejection sampling. Since the public key computed by rejection sampling is not secret, the SCA countermeasure against timing attack is not required. Nevertheless, since most SCA evaluation methods, e.g., t-test, can be performed

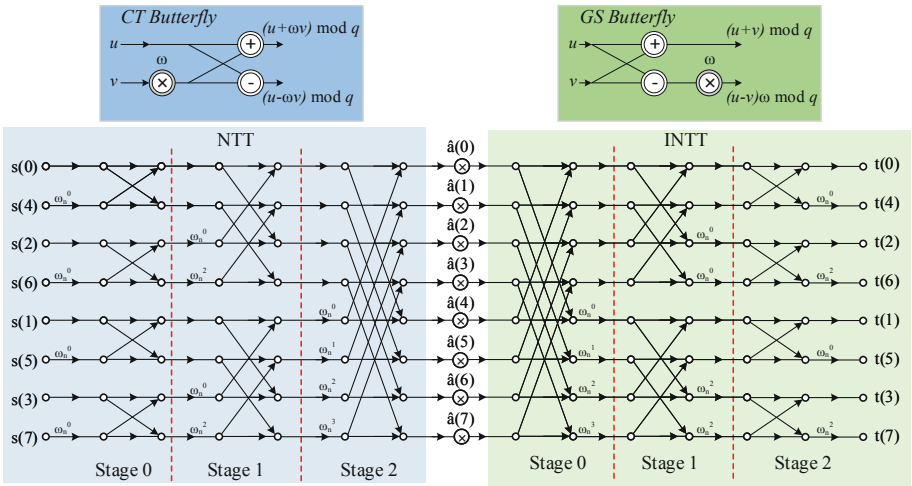

Fig. 2. An 8-point NTT-based polynomial multiplication. Dataflow graph includes CT butterfly-based NTT, point-wise multiplication, and GS butterfly-based INTT. Polynomial $\hat{a}$ is in NTT domain and s and t are in normal domain.

on a constant-time design, we perform constant rounds of SHAKE-128 to form a constant-time implementation. While 112 samples can be evaluated by rejection unit for each round of Keccak core, the failure probability that 256 valid coefficients can be sampled by performing different rounds of SHAKE-128 is listed in Table 2. As a result, four rounds are performed for each required vector of $\mathbf{A}$ while the failure probability is negligible. In our optimized architecture, this unit works in parallel with the Keccak core. Therefore, the latency for rejection sampling is completely absorbed within the latency for a concurrently running Keccak core. An alternative approach is to sample directly on the Keccak state; however, this approach increases the complexity of hardware-based architecture.

3.3 Binomial Sampling

There are two different configurations for binomial sampling unit corresponding to $\eta = 2$ and $\eta = 3$. We propose an optimized configurable architecture that can compute the Hamming weight for both values of η. To support both architectures, the data from PISO is buffered in a 96-bit register, of which only 64-bit is utilized in $\eta = 2$. The results are in $[-\eta, \eta]$ are presented in 13-bit signed representation to simplify the addressing. Our proposed binomial sampling architecture for two first sampled data is depicted in Fig. 3.

3.4 Configurable Butterfly Core

To avoid the bit-reverse cost in polynomial multiplication, two different butterfly configurations, i.e., CT and GS, are required for NTT and INTT, respectively.

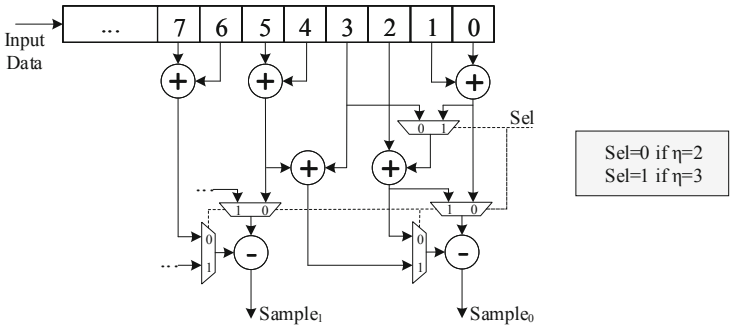

Fig. 3. Proposed configurable binomial sampling unit.

In our proposed architecture, a configurable butterfly core is proposed to support both CT and GS operations and reduce required hardware resources. In order to design a high-performance architecture, the resource sharing technique from [5,37] is extended by using compact storage for pre-computed twiddle factors from [35] and doubled bandwidth scheme from [18,30].

Since CT configuration is used in NTT, we assume that the input polynomials are in normal order, while the public and secret keys are in bit-reverse order. Hence, the point-wise multiplication works in bit-reverse order in the NTT domain, and the results are transformed back to the normal domain with normal order employing GS configuration. We take advantage of the NTT definition in the Kyber scheme to perform two independent NTT computations for odd and even coefficients based on (6) and (7).

$$\hat{a}_{2i} = \sum_{j=0}^{127} a_{2j} \zeta^{(2i+1)j} \tag{6}$$

$$\hat{a}_{2i+1} = \sum_{j=0}^{127} a_{2j+1} \zeta^{(2i+1)j} \tag{7}$$

Two butterfly cores are employed in parallel for NTT computation to reduce execution time to $\frac{N}{2} \log_2 \frac{N}{4}$. Each line of RAM block stores two consecutive coefficients, i.e., $\mathbf{s}_{i,2j}$ and $\mathbf{s}_{i,2j+1}$, in two columns to feed both butterfly cores. Reading from two addresses of memory provides four coefficients, i.e., $\mathbf{s}_{i,2j}$ and $\mathbf{s}_{i,2j+1}$ from address j, and $\mathbf{s}_{i,2k}$ and $\mathbf{s}_{i,2k+1}$from address k of memory. The lower columns storing the even coefficients, i.e., $\mathbf{s}_{i,2j}$ and $\mathbf{s}_{i,2k}$, are used for the first butterfly, while the higher columns including the odd coefficients, i.e., $\mathbf{s}_{i,2j+1}$ and $\mathbf{s}_{i,2k+1}$, are fed into the second core. The results should be stored similarly in the second RAM.

Our proposed NTT architecture includes five different main modules: two RAM blocks, an address generator (working in three modes for NTT, INTT, and point-wise multiplication), a pre-computed twiddle factor ROM, and an

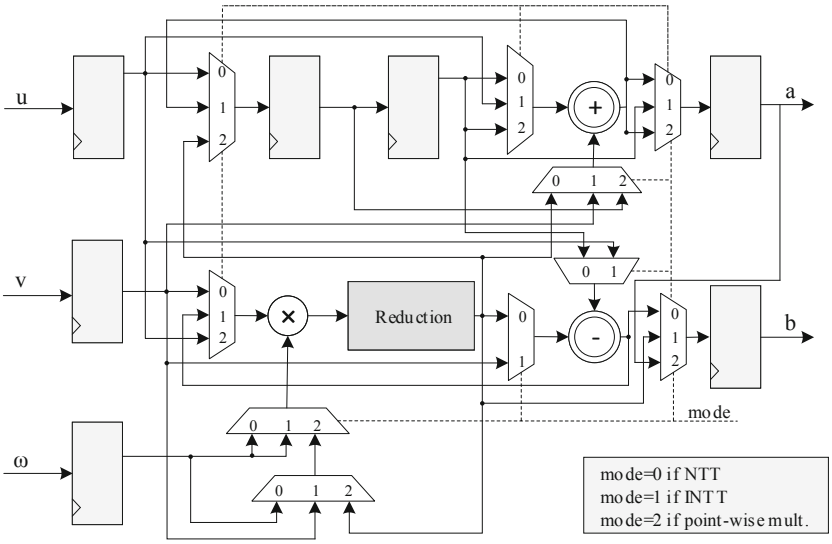

Fig. 4. Proposed configurable butterfly architecture.

arithmetic unit. The dual-port capabilities of the RAM blocks are exploited in our proposed design to increase efficiency. Moreover, the address generator computes the two read and write addresses to load and store the required coefficients as well as the corresponding address for the twiddle factor in each operation.

The arithmetic unit consists of a multiplier, a modular reduction, an addition, and a subtraction, while there are also some registers to balance the pipeline latency in different configurations. The proposed architecture is depicted in Fig. 4. Different reduction units have been studied in the literature. While Barrett reduction works in the normal domain, Montgomery reduction needs more resources and latency to perform the transformation into and out of the Montgomery domain. The proposed architecture employing Barrett reduction is implemented in a pipelined fashion to increase the throughput to 1 output per cycle.

3.5 Area/Performance Trade-Offs

The main goal of the proposed architecture is to achieve high-speed computation employing small area requirements. However, we can target different area/performance trade-offs by increasing the number of butterfly cores, taking advantage of polynomial vector structure in the Kyber algorithm. For example, in Kyber-512 having 2 polynomial vectors, increasing the number of implemented butterfly core from 2 to 4 can drastically reduce to a half of NTT/INTT latency. Nevertheless, implementing more arithmetic units needs higher bandwidth. In this case, the number of occupied RAM blocks will be doubled while they are implemented in parallel to provide the required bandwidth.

Table 3. Implementation results for Kyber KEM on 65-nm ASIC

Protocol	Area		Freq [MHz]	Cycles			Total time† [μs]
	Logic gates [kGE]	SRAM [kB]		KeyGen [CCs]	Encaps [CCs]	Decaps [CCs]	
Kyber-512	95	10	200	4,267	6,769	10,015	83.9
Kyber-768	93	22	200	6,641	9,683	13,569	116.3
Kyber-1024	104	24	200	9,971	13,278	17,676	154.8

† Total time includes Encaps + Decaps, as the key generation can be done offline.

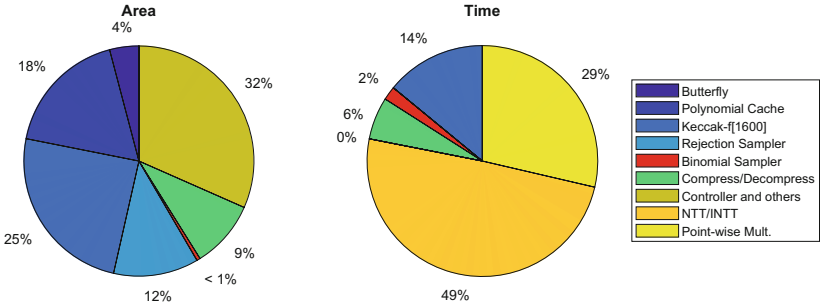

Fig. 5. Area breakdown (left) and time breakdown (right) during encapsulation of Kyber-512.

4 Implementation Results And Comparisons

Our proposed architecture is synthesized using a 65-nm TSMC cell library [38] to show the required area. VHDL has been used as the design entry to the Synopsys Design Compiler [38]. In addition, using the area of a NAND gate in the utilized 65-nm library, which is 1.35 μm^2, we have provided the gate equivalent (GE) so that area comparisons among different technologies are meaningful. Although similar to the previous work presented in [13], we have not fabricated a chip on silicon, our detailed results are intended for benchmarking the metrics for the previous and proposed research works. All the designs are synthesized with a 5 ns clock period.

Table 3 reports the required hardware resources and latency specifications for our proposed architecture in three different security levels. The total time is the summation of key encapsulation and key decapsulation (Encaps + Decaps), as the key generation can be done offline. We implement 2, 3, and 4 of our proposed NTT architecture for security levels 1, 3, and 5, respectively. As one can see, for NIST level 1 security, our proposed architecture occupies 95 kGE, and 10 kB SRAM. It also runs at 200 MHz and performs the whole Kyber protocol in 83.9 μs.

The area breakdown of our design and the latency breakdown in encapsulation of Kyber-512 is illustrated in Fig. 5. Our proposed Keccak, butterfly, and sampling units utilize 4%, 25%, and 13% of the total area. For reporting latency breakdown, when the butterfly is parallel with other units, the latency is consid-

Table 4. Comparisons with existing hardware-based implementations of NTT for Kyber KEM.

Work	Platform	Tech [nm]	Freq [MHz]	NTT [CCs]	INTT [CCs]	Point-wise Mult. [CCs]
Karabulut et al. [19]	Virtex-7	–	NA	43,756	NA	NA
Alkim et al. [7]	Artix-7	–	59	6,868	6,367	2,395
Chen et al. [18]	Artix-7	–	130	2,055	NA	7,197
Huang et al. [15]	Artix-7	–	155	1,834	NA	NA
Bisheh-Niasar et al. [17]	Artix-7	–	222	324	324	NA
Fritzmann et al. [21]	ASIC	65	25	2,056	NA	NA
Fritzmann et al. [6]	ASIC	65	45	1,935	1,930	NA
Banerjee et al. [5]	ASIC	40	72	1,289	NA	NA
Xin et al. [13]	ASIC	28	300	41	NA	NA
This work	**ASIC**	**65**	**200**	**474**	**602**	**1,289**

ered for NTT or point-wise multiplication. As one can see, the more expensive operation from a resource utilization point of view in Kyber KEM is Keccak core. However, using the compact version of Keccak core results in more delay in sampling units which reduces the total efficiency. In the timing breakdown, the point-wise multiplication and NTT are the first two time-consuming operations. It should be noted that although Kyber reduces the number of required stages for NTT computation from 8 stages to 7, the special form of multiplication in this scheme increases the computation overhead in point-wise multiplication.

Table 4 reports area and latency specifications for our NTT architecture which works in three different modes, i.e., NTT and INTT, and point-wise multiplication. Other state-of-the-art NTT designs for the Kyber scheme over hardware platforms are also listed. We report the results for both Kyber parameters in the previous rounds and round-3, i.e., $q = 3329$ and $q = 7681$, respectively.

Our results show a significant improvement requiring only 474, 602, and 1,289 clock cycles for NTT, INTT, and point-wise multiplication, respectively. We presented a highly optimized FPGA-based NTT core in our previous work [17] which shows 31% NTT performance improvement at the cost of occupying a 2×2 butterfly units. The work in [6] optimized an NTT core based on hardware/software approach over RISC-V architecture, while it works at 45 MHz on the ASIC platform. Our architecture results in 4× and 18× speedup in terms of required cycles and time for NTT computation, respectively. The works in [19] and [7] also presented an NTT architecture over RISC-V, which requires considerably greater cycle count, while our optimized design achieves 92× and 49× speedup, respectively. The FPGA-based design was proposed in [15] employing Montgomery reduction; however, our design reduces the required cycles achieving a speedup factor of 3.9. Our design also improves almost 6.7× and 8.6× the required latency for NTT and point-wise multiplication, respectively. In [5],

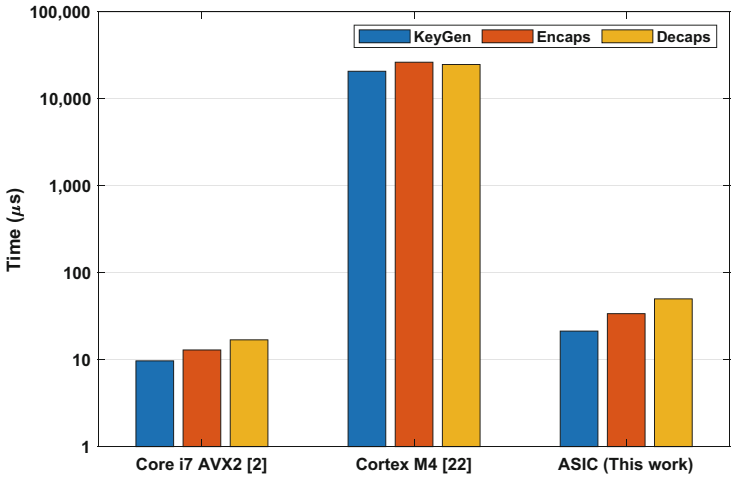

Fig. 6. Comparisons with other software implementations of Kyber-512.

a RISC-V-based architecture proposed working at 72 MHz over a 40-nm ASIC platform. Our proposed architecture achieves 7.5× better timing results compared to this work. The work of [13] employed the vectorized architecture at the cost of utilizing 521 kGE. Although the proposed architecture in [13] shows 17.3× speedup, it consumes 137× more resources compared to our design.

Figure 6 depicts the comparison of timing results for our proposed architecture and software implementations of Kyber on a mainstream desktop Intel Core i7 CPU with the optimization of AVX2 and an embedding Cortex-M4 CPU. The Core i7 CPU works at 3,492 MHz, while the results for the Discovery board are reported in 24 MHz. As one can see, our proposed ASIC architecture is 2.8× slower than Intel core [2]. However, ours achieves more than 600× speedup compared to [22]. Nevertheless, this speedup is not surprising since HW significantly reduces total time compared to SW employing parallel computation.

Table 5 lists the detailed resource consumption and performance results (frequency, required cycles, and execution time) of Kyber coprocessor designs for all NIST security levels. There are several hardware/software implementations targeting Kyber KEM in the literature. However, a direct comparison is not possible between the listed hardware implementations due to the varying techniques of different platforms, targeting different optimization goals, and using different design methodologies. The work in [5] implemented a configurable coprocessor based on a RISC-V architecture that can be used for multiple lattice-based schemes, including Kyber. Its architecture performs almost 263 KEM per second for Kyber-512, which is 45× slower than our design. In [6], another RISC-V-based architecture was proposed to accelerate NTT-based schemes. This design requires 23× more cycles for encapsulation and decapsulation while consuming 2.8× more resources. Additionally, in [13], a high-performance hardware architecture was proposed based on RISC-V. Nevertheless, our design achieves 5×

Table 5. ASIC Implementation results for Kyber KEM and comparison with state-of-the-art.

Work	Tech [nm]	Area		Total Area† [kGE]	Freq [MHz]	Latency			Total Time‡ [μs]	$A \times T$ [GE×s]
		Logic gates [kGE]	SRAM [kB]			KeyGen [kCCs]	Encaps [kCCs]	Decaps [kCCs]		
Kyber-512										
Basu et al. [11]	65	1,341	–	3,531	200	–	–	43	–	–
Fritzmann et al. [6]	65	170	465§	635	45	150	193	205	8,844	5,615
Banerjee et al. [5]	40	106	40.25	547	72	75	132	142	3,806	2,081
Xin et al. [13]	28	979	12	1,131	300	19	46	80	420	475
This work	**65**	**95**	**10**	**222**	**200**	**4**	**7**	**10**	**84**	18
Kyber-768										
Fritzmann et al. [6]	65	170	465§	635	45	273	326	340	14,800	9,398
Banerjee et al. [5]	40	106	40.25	547	72	112	178	191	5,125	2,803
This work	**65**	**93**	**22**	**372**	**200**	**7**	**10**	**14**	**116**	43
Kyber-1024										
Fritzmann et al. [6]	65	170	465§	635	45	350	405	425	18,444	11,711
Banerjee et al. [5]	40	106	40.25	547	72	149	223	241	6,444	3,524
Xin et al. [13]	28	979	12	1,131	300	40	82	136	727	822
This work	**65**	**104**	**24**	**409**	**200**	**10**	**14**	**18**	**155**	63

† The total area is calculated based on the reported fabric dimension corresponding to their technology. For non-fabricated results, a rough estimation of 55% area overhead is considered for implementing SRAM similar to [39].
‡ Total time includes Encaps + Decaps, as the key generation can be done offline.
§ The reported numbers are in kGE.

faster KEM and improves 80% resource utilization while occupying 5× fewer area compared to [13]. An HLS evaluation was proposed in [11] for Kyber-512 employing different implementations for encapsulation and decapsulation. However, this approach comes at a considerably far larger area consumption. Hence, our design achieves almost 16 and 4.3 times better are and timing results compared to HLS-based implementation. For the other security level, the same trend can be seen.

We also illustrated a summary of comparison with other ASIC implementation of Kyber-512 in Fig. 7. For different implementations, the total time (T) and the equivalent area (A), including the required logic gates and SRAM are shown. The efficiency in terms of $A \times T$ is also computed. As one can see, our proposed design achieves better latency using significantly fewer resources. Furthermore, the proposed Kyber-512 implementation improves 312×, 116×, and 26× efficiency compared to [5,6] and [13], respectively.

The experimental result shows that taking advantage of the proposed NTT architecture to implement lattice-based KEM schemes as full-hardware architecture results in high-speed and efficient design. For Kyber KEM, our coprocessor architecture outperforms the reported implementations in the literature in terms of efficiency. Furthermore, although one of the drawbacks of various post-quantum cryptosystems is requiring larger key sizes and more computational power than the current pre-quantum algorithms, the efficiency of our proposed implementation already has performance levels comparable to or even significantly better than pre-quantum algorithms [8,9,40].

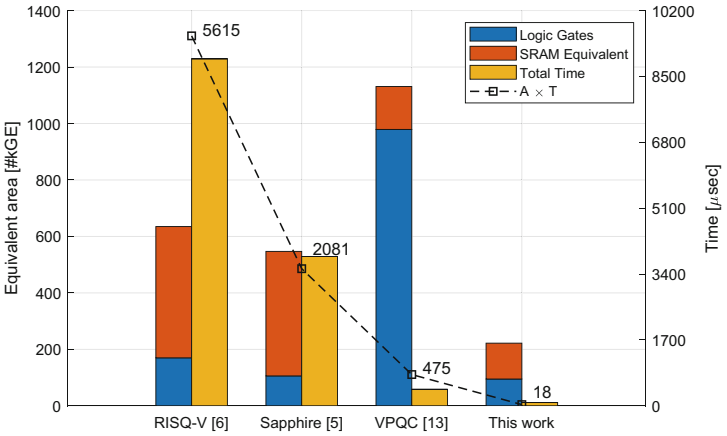

Fig. 7. Comparisons with other ASIC implementations of Kyber-512.

Table 6. Comparisons with existing FPGA-based PQC implementations of CCA-secure KEM schemes in NIST security level 1.

Protocol	Platform	Area (Gates Equivalent) or (LUTs/FFs/Slices/DSPs/BRAMs)	Freq [MHz]	Time [us]
SIKEp434 [43]	Virtex-7	12,818/18,271/5,527/195/32	249.6	8,800
Frodo-640 [44]†	Artix-7	6,881/5,081/1,947/16/12.5	149	2,621
LightSaber [45]	ASIC	742 kGE‡	400	5
Kyber-512 [This work]	**ASIC**	**222 kGE**	**200**	**84**

† Different architectures for Encaps and Decaps are used.
‡ The reported area is 0.38 mm^2 in 40 nm process.

Several performance optimizations of other PQC schemes were proposed recently [41–45]. Table 6 lists other PQC scheme results implemented on the hardware platform for NIST security level 1. Elkhatib *et al.* in [43] implemented a supersingular isogeny-based KEM performed in 8.8 ms. Howe *et al.* [44] presented a flexible FrodoKEM architecture that performs 825 and 710 encapsulations and decapsulation. The work of [45] proposed an energy-efficient architecture for Saber employing 8-level hierarchical Karatsuba, which consumes 859 and 1,075 clock cycles for encapsulation and decapsulation, respectively.

5 Conclusion

This paper proposed a high-performance and efficient architecture for NTT-based polynomial multiplication and lattice-based public-key cryptography coprocessor with Kyber KEM as a case study. We optimize the implementation of the NTT core by creating a configurable butterfly core. Besides, we propose a coprocessor architecture that can perform all KEM operations for Kyber. The proposed Kyber coprocessor architecture performs key generation, encapsulation, and decapsulation in 21.3, 33.8, and 50 µs for a security level comparable

to AES-128 respectively, by consuming only 95 kGE and 10 kB SRAM, on an 65-nm ASIC platform.

Acknowledgment. The authors would like to thank the reviewers for their conservative comments. This research is supported in parts by an award from NSF 1801341.

References

1. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: 35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20–22 November 1994, pp. 124–134 (1994)
2. Avanzi, R., et al.: CRYSTALS-Kyber: algorithm specification and supporting documentation (version 3.0). Submission to the NIST post-quantum cryptography standardization project (2020)
3. Langlois, A., Stehlé, D.: Worst-case to average-case reductions for module lattices. Des. Codes Cryptogr. **75**(3), 565–599 (2015)
4. NIST: Submission requirements and evaluation criteria for the post-quantum cryptography standardization process. National Institute of Standards and Technology (2016)
5. Banerjee, U., Ukyab, T.S., Chandrakasan, A.P.: Sapphire: a configurable crypto-processor for post-quantum lattice-based protocols. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2019**(4), 17–61 (2019)
6. Fritzmann, T., Sigl, G., Sepúlveda, J.: RISQ-V: tightly coupled RISC-V accelerators for post-quantum cryptography. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(4), 239–280 (2020)
7. Alkim, E., Evkan, H., Lahr, N., Niederhagen, R., Petri, R.: ISA extensions for finite field arithmetic accelerating Kyber and NewHope on RISC-V. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(3), 219–242 (2020)
8. Bisheh Niasar, M., Elkhatib, R., Azarderakhsh, R., Mozaffari Kermani, M.: Fast, small, and area-time efficient architectures for key-exchange on Curve25519. In: 27th IEEE Symposium on Computer Arithmetic, ARITH 2020, Portland, OR, USA, 7–10 June 2020, pp. 72–79 (2020)
9. Bisheh Niasar, M., Azarderakhsh, R., Kermani, M.M.: Efficient hardware implementations for elliptic curve cryptography over curve448. In: Bhargavan, K., Oswald, E., Prabhakaran, M. (eds.) INDOCRYPT 2020. LNCS, vol. 12578, pp. 228–247. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-65277-7_10
10. Bisheh-Niasar, M., Azarderakhsh, R., Mozaffari-Kermani, M.: Cryptographic accelerators for digital signature based on Ed25519. IEEE Trans. Very Large Scale Integr. Syst. **29**(7), 1297–1305 (2021)
11. Basu, K., Soni, D., Nabeel, M., Karri, R.: NIST post-quantum cryptography-a hardware evaluation study. IACR Cryptol. ePrint Arch. 2019, 47 (2019)
12. Banerjee, U., Ukyab, T.S., Chandrakasan, A.P.: Sapphire: a configurable crypto-processor for post-quantum lattice-based protocols (extended version). IACR Cryptol. ePrint Arch. 2019, 1140 (2019)
13. Xin, G., et al.: VPQC: a domain-specific vector processor for post-quantum cryptography based on RISC-V architecture. IEEE Trans. Circuits Syst. I Regul. Pap. **67-I**(8), 2672–2684 (2020)

14. Dang, V.B., Farahmand, F., Andrzejczak, M., Mohajerani, K., Nguyen, D.T., Gaj, K.: Implementation and benchmarking of round 2 candidates in the NIST post-quantum cryptography standardization process using hardware and software/hardware co-design approaches. IACR Cryptol. ePrint Arch. 2020, 795 (2020)
15. Huang, Y., Huang, M., Lei, Z., Wu, J.: A pure hardware implementation of CRYSTALS-Kyber PQC algorithm through resource reuse. IEICE Electronics Express advpub (2020)
16. Xing, Y., Li, S.: A compact hardware implementation of CCA-secure key exchange mechanism CRYSTALS-KYBER on FPGA. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(2), 328–356 (2021)
17. Bisheh-Niasar, M., Azarderakhsh, R., Mozaffari-Kermani, M.: High-speed NTT-based polynomial multiplication accelerator for CRYSTALS-Kyber post-quantum cryptography. IACR Cryptol. ePrint Arch. 2021, 563 (2021)
18. Chen, Z., Ma, Y., Chen, T., Lin, J., Jing, J.: Towards efficient Kyber on FPGAs: a processor for vector of polynomials. In: 25th Asia and South Pacific Design Automation Conference, ASP-DAC 2020, Beijing, China, 13–16 January 2020, pp. 247–252 (2020)
19. Karabulut, E., Aysu, A.: RANTT: A RISC-V architecture extension for the number theoretic transform. In: 2020 30th International Conference on Field-Programmable Logic and Applications (FPL), pp. 26–32 (2020)
20. Mert, A.C., Karabulut, E., Öztürk, E., Savas, E., Aysu, A.: An extensive study of flexible design methods for the number theoretic transform. IEEE Trans. Comput., 1–1 (2020)
21. Fritzmann, T., Sepúlveda, J.: Efficient and flexible low-power NTT for lattice-based cryptography. In: IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2019, McLean, VA, USA, 5–10 May 2019, pp. 141–150 (2019)
22. Botros, L., Kannwischer, M.J., Schwabe, P.: Memory-efficient high-speed implementation of kyber on cortex-M4. In: Buchmann, J., Nitaj, A., Rachidi, T. (eds.) AFRICACRYPT 2019. LNCS, vol. 11627, pp. 209–228. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-23696-0_11
23. Alkim, E., Bilgin, Y.A., Cenk, M., Gérard, F.: Cortex-m4 optimizations for R, M LWE schemes. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(3), 336–357 (2020)
24. Kannwischer, M.J., Rijneveld, J., Schwabe, P., Stoffelen, K.: PQM4: post-quantum crypto library for the ARM Cortex-M4 (2018)
25. Xing, Y., Li, S.: An efficient implementation of the NewHope key exchange on FPGAs. IEEE Trans. Circuits Syst. I Regul. Pap. **67-I**(3), 866–878 (2020)
26. Bos, J.W., et al.: CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM. In: 2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, 24–26 April 2018, pp. 353–367 (2018)
27. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 537–554. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_34
28. Pöppelmann, T., Güneysu, T.: Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware. In: Hevia, A., Neven, G. (eds.) LATINCRYPT 2012. LNCS, vol. 7533, pp. 139–158. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33481-8_8
29. Roy, S.S., Vercauteren, F., Mentens, N., Chen, D.D., Verbauwhede, I.: Compact ring-LWE cryptoprocessor. In: Batina, L., Robshaw, M. (eds.) CHES 2014. LNCS, vol. 8731, pp. 371–391. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44709-3_21

30. Zhang, N., Yang, B., Chen, C., Yin, S., Wei, S., Liu, L.: Highly efficient architecture of NewHope-NIST on FPGA using low-complexity NTT/INTT. IACR Trans. Cryptogr. Hardw. Embed. Syst. 2020(2), 49–72 (2020)
31. Longa, P., Naehrig, M.: Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In: Foresti, S., Persiano, G. (eds.) CANS 2016. LNCS, vol. 10052, pp. 124–139. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48965-0_8
32. Cooley, J., Tukey, J.W.: An algorithm for the machine calculation of complex Fourier series. Math. Comput. **19**, 297–301 (1965)
33. Gentleman, W.M., Sande, G.: Fast Fourier transforms: for fun and profit. In: American Federation of Information Processing Societies: Proceedings of the AFIPS 1966 Fall Joint Computer Conference, San Francisco, California, USA, 7–10 November 1966, pp. 563–578 (1966)
34. Pöppelmann, T., Oder, T., Güneysu, T.: High-performance ideal lattice-based cryptography on 8-bit ATxmega microcontrollers. In: Lauter, K., Rodríguez-Henríquez, F. (eds.) LATINCRYPT 2015. LNCS, vol. 9230, pp. 346–365. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-22174-8_19
35. Du, C., Bai, G.: Towards efficient polynomial multiplication for lattice-based cryptography. In: IEEE International Symposium on Circuits and Systems, ISCAS 2016, Montréal, QC, Canada, 22–25 May 2016, pp. 1178–1181 (2016)
36. Bertoni, G., Daemen, J., Hoffert, S., Peeters, M., Assche, G.V.: Keccak in VHDL (2020)
37. Kuo, P.C., et al.: High performance post-quantum key exchange on FPGAs. IACR Cryptology ePrint Archive, 690 (2017)
38. Synopsys. http://Synopsys.com
39. van der Leest, V., van der Sluis, E., Schrijen, G.-J., Tuyls, P., Handschuh, H.: Efficient implementation of true random number generator based on SRAM PUFs. In: Naccache, D. (ed.) Cryptography and Security: From Theory to Applications. LNCS, vol. 6805, pp. 300–318. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28368-0_20
40. Bisheh-Niasar, M., Azarderakhsh, R., Mozaffari-Kermani, M.: Area-time efficient hardware architecture for signature based on Ed448. IEEE Trans. Circ. Syst. II Express Briefs, **68**(8), 2942–2946 (2021). https://doi.org/10.1109/TCSII.2021.3068136
41. Anastasova, M., Azarderakhsh, R., Mozaffari Kermani, M.: Fast strategies for the implementation of SIKE round 3 on ARM Cortex-M4. IACR Cryptol. ePrint Arch. 2021, 115 (2021)
42. Seo, H., Anastasova, M., Jalali, A., Azarderakhsh, R.: Supersingular isogeny key encapsulation (SIKE) round 2 on ARM Cortex-M4. IACR Cryptol. ePrint Arch. 2020, 410 (2020)
43. Elkhatib, R., Azarderakhsh, R., Mozaffari Kermani, M.: Highly optimized montgomery multiplier for SIKE primes on FPGA. In: 27th IEEE Symposium on Computer Arithmetic, ARITH 2020, Portland, OR, USA, 7–10 June 2020, pp. 64–71 (2020)
44. Howe, J., Martinoli, M., Oswald, E., Regazzoni, F.: Exploring parallelism to improve the performance of frodokem in hardware. Cryptology ePrint Archive, Report 2021/155 (2021)
45. Zhu, Y., et al.: A high-performance hardware implementation of saber based on Karatsuba algorithm. IACR Cryptol. ePrint Arch. 2020, 1037 (2020)