Instruction-Set Accelerated Implementation of CRYSTALS-Kyber

Mojtaba Bisheh-Niasar, *Student Member, IEEE*, Reza Azarderakhsh, *Member, IEEE*, and Mehran Mozaffari-Kermani, *Senior Member, IEEE*

***Abstract*— Large scale quantum computers will break classical public-key cryptography protocols by quantum algorithms such as Shor's algorithm. Hence, designing quantum-safe cryptosystems to replace current classical algorithms is crucial. Luckily there are some post-quantum candidates that are assumed to be resistant against future attacks from quantum computers, and NIST is considering standardizing them. Among these candidates, lattice-based cryptography sounds more interesting than others due to the performance results as well as confidence in the security. There are few works in the literature evaluating the performance of lattice-based cryptography in hardware. In this paper, we focus on Cryptographic Suite for Algebraic Lattices (CRYSTALS) key exchange mechanisms known as Kyber and provide an instruction-set hardware architecture and implement on Xilinx Artix-7 FPGA for performance evaluation and testing. Our proposed architecture provides an efficient and high-performance set of components to perform polynomial sampling, number-theoretic transform (NTT), and point-wise multiplication to speed up lattice-based post-quantum cryptography (PQC). This architecture implemented on ASIC outperforms state-of-the-art implementations.**

***Index Terms*— ASIC, FPGA, hardware architecture, Kyber, lattice-based cryptography, post-quantum cryptography.**

I. Introduction

QUANTUM computing development constitutes a significant threat to classical public-key cryptography protocols based on Shor's algorithm [1]. Most current cryptosystems, i.e., RSA and Elliptic Curve Cryptography (ECC), are envisioned to be broken when large quantum computers will be built. Thus, designing the lattice-based cryptosystem as one of the most promising algorithms in Post-Quantum Cryptography (PQC) based on alternative mathematical features has become a fundamental research topic.

Recently, the National Institute of Standards and Technology (NIST) announced the third-round finalists, which includes 4 key encapsulation mechanisms (KEMs) and 3 signature schemes [2]. Among these KEM schemes, CRYSTALS-Kyber shares a common framework with the CRYSTALS-Dilithium signature scheme [2]. This scheme also supports efficient matrix-vector and vector-vector multiplication over a polynomial ring using the fast number-theoretic transform (NTT) [3]. Although the optimization of NTT-based multiplication is not a new idea and is used in countless applications, particularly in signal processing, it is still a performance bottleneck in the lattice-based cryptography implementation. Thus, several works have been done to optimize NTT from different perspectives, such as resource utilization, performance, efficiency, and energy consumption.

Recently, implementations of lattice-based cryptography have been investigated on various platforms. While software (SW) implementations offer programming capabilities, flexibility, and a shorter design cycle, the hardware (HW) platforms accelerate the computations and result in significantly higher throughput. Recently, there are considerable efforts to implement cryptosystems using hardware-software (HW/SW) co-design. This method makes the design smaller, slower, and more controllable/programmable compared to pure HW schemes at the cost of implementing a software-based processor. Furthermore, a HW/SW co-design requires a shorter design period; nevertheless, this method may not lead to the best performance. On the other hand, pure hardware implementations can be significantly accelerated using well-known optimization strategies, including register balancing, parallelization, and resource sharing, to increase the overall throughput of the hardware architectures. The main difficulty of this strategy is its hand-optimized design requiring a longer time and may be achieved at the cost of losing flexibility.

To transition to PQC, we must develop hybrid cryptosystems to maintain industry or government regulations, while PQC updates will be applied thoroughly. Therefore, classical cryptosystems, e.g. ECC, cannot be eliminated even if PQC will significantly be developed. The instruction-set processor builds an appropriate platform for accelerated implementation compared to SW and HW/SW. while the architecture remains flexible compared to highly optimized HW. Specifically, the flexible HW architecture is a promising solution for

Manuscript received December 28, 2020; revised April 13, 2021 and June 10, 2021; accepted August 11, 2021. Date of publication August 30, 2021; date of current version November 9, 2021. This work was supported by NSF under Grant 1801341. This article was recommended by Associate Editor S. Yin. *(Corresponding author: Mojtaba Bisheh-Niasar.)*

Mojtaba Bisheh-Niasar is with the Department of Computer and Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton, FL 33431 USA, and also with I-SENSE, Florida Atlantic University, Boca Raton, FL 33431 USA (e-mail: mbishehniasa2019@fau.edu).

Reza Azarderakhsh is with the Department of Computer and Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton, FL 33431 USA, also with I-SENSE, Florida Atlantic University, Boca Raton, FL 33431 USA, and also with PQSecure Technologies LLC, Boca Raton, FL 33431 USA (e-mail: razarderakhsh@fau.edu).

Mehran Mozaffari-Kermani is with the Department of Computer Engineering and Science, University of South Florida, Tampa, FL 33620 USA (e-mail: mehran2@usf.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSI.2021.3106639.

Digital Object Identifier 10.1109/TCSI.2021.3106639

1549-8328 © 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: University of South Florida. Downloaded on March 02,2022 at 23:49:46 UTC from IEEE Xplore. Restrictions apply.

integrating classic cryptosystems and PQC to move towards hybrid systems.

Kyber is notable for high speed and constant-time implementations. It has to be implemented in various platforms subject to the performance requirement. However, Kyber has not got sufficient study in the field of hardware implementation. Therefore, investigation of the hardware implementation is required considering the advantages of FPGA-based architectural designs to exploit parallelism, which leads to improvements in the efficiency of the overall system. In this paper, we implement a pure hardware design since it is faster and could be integrated into any HW/SW co-design solutions.

A. Related Work

Software implementation of Kyber has been studied by Botros *et al.* in [3], proposing a memory-efficient high-speed implementation on Cortex-M4. Recently, several PQC schemes have been implemented, targeting HW/SW co-design. The work of [4] was one of the first initiatives of post-quantum acceleration using high-level synthesis (HLS). Furthermore, Banerjee *et al.* in [5] proposed a flexible ASIC crypto-processor to support several lattice-based algorithms into a RISC-V architecture, including Frodo, NewHope, qTESLA, and CRYSTALS-Kyber/Dilithium. This work is extended in [6] to show FPGA validation results. Their design strategy targets reducing power consumption. The authors in [7] employ the RISC-V processor integrated with a finite field multiplier to accelerate polynomial multiplications in a lightweight architecture of NewHope and Kyber. In [8], performing vectorized modular arithmetic and NTT computations are proposed employing RISC-V for NewHope, Kyber, and Saber. The vector processor architecture based on the extensible RISC-V architecture has been studied in [9], which shows a remarkable speed up occupying 979k gate equivalent (GE) in ASIC implementations.

The pure hardware architectures of Kyber are proposed in [10]–[13]. The work of [10] heavily relies on BlockRAM primitives between components to perform arithmetic tasks and store intermediate results. We addressed the high-performance implementation of Kyber in our previous work [13] as the fastest Kyber design in the literature. The authors in [14] proposed a Kyber processor for computing NTT and point-wise multiplication. An instruction-set coprocessor for Saber is presented in [15] to design a flexible hardware architecture using the quadratic-complexity schoolbook polynomial multiplication algorithm. Schoolbook polynomial multiplication is also employed in [16].

Since NTT plays a central role in lattice-based cryptography, several hardware implementations focus on NTT from performance, efficiency, and flexibility perspectives. The work of [17], [18] introduced a scalable NTT architecture that can be used for various lattice-based schemes. Furthermore, the authors in [19] proposed a RISC-V architecture to increase efficiency and flexibility for NTT computation used in NewHope, qTESLA, CRYSTALS-Kyber, CRYSTALS-Dilithium, and Falcon. Additionally, Fritzmann and Sepúlveda [20] proposed an efficient and low-power NTT, which reduces the number of clock cycles to $n\log(n)$ cycles. The authors in [21] proposed a low-complexity NTT/INTT in the architecture of NewHope-NIST.

The proposed architecture combines the NTT, INTT, and point-wise multiplication architectures in an efficient way to utilize significantly fewer resources and improve the overall performance. To do so, using the Cooley-Turkey (CT) as NTT and the Gentleman-Sande (GS) as INTT [22], [23] is a well-known trick in the literature. Moreover, the resource sharing technique from [5], [24] is extended by using compact storage for pre-computed twiddle factors from [25] and doubled bandwidth scheme from [14], [21] to account for the high-performance architecture.

B. Our Contributions

To the best of our knowledge, there appear to be very few pure hardware implementations that focus only on the Kyber cryptosystem and make the best of all its features. This paper proposes an efficient hardware implementation of the module lattice-based post-quantum KEM CRYSTAL-Kyber on a Xilinx Artix-7 FPGA (as recommended by NIST) and the application specific integrated circuit (ASIC) platform. Our proposed architecture provides an efficient and high-performance set of components, including polynomial sampling, NTT, and point-wise multiplication, to accelerate lattice-based PQC exploiting fewer resources. The contributions of this paper are itemized in the following:

1) We propose a new approach for implementing a resource-efficient reconfigurable butterfly core on FPGA. We reduce the execution time for Kyber NTT computation from $\frac{N}{2}\log_2\frac{N}{2}+2N$ to $\frac{N}{2}\log_2\frac{N}{4}$ by doubling the transform throughput and merging the pre-processing into NTT algorithm. We also customize a memory addressing strategy to implement a high-speed polynomial multiplier on the target platform.
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) Our fast and scalable architecture provides a constant-time implementation over three different quantum security levels. To enhance our HW accelerator from a flexibility point of view, we design a set of customized high-level instruction codes to run the protocol. Hence, this set identifies the control flow of the proposed components and provides flexibility for integration with host processors.
4) We employ various optimization techniques to achieve an overall optimization in terms of efficiency, including parallelization, resource sharing, utilizing distributed RAM and ROM blocks, which significantly improve the area-time product. The proposed implementation is constant-time and is resistant to known timing attacks.

The rest of the paper is organized as follows. In Sec. II, we discuss the preliminaries. In Sec. III, our proposed algorithms

Authorized licensed use limited to: University of South Florida. Downloaded on March 02,2022 at 23:49:46 UTC from IEEE Xplore. Restrictions apply.

TABLE I
THE LIST OF SYMBOLS AND NOTATIONS USED IN THIS PAPER

Symbol	Definition
Regular font lower-case letter ($a/\hat{a}$)	Polynomial in normal/NTT domain
Bold lower-case letter ($\mathbf{a}/\hat{\mathbf{a}}$)	Polynomial vector in normal/NTT domain
Bold upper-case letter ($\mathbf{A}/\hat{\mathbf{A}}$)	Polynomial matrix in normal/NTT domain
$\mathbf{a}^T/\mathbf{A}^T$	Transpose of vector/matrix
$\mathbf{a} \circ \mathbf{b}$	Point-wise multiplication

TABLE II
PARAMETER SETS FOR KYBER IMPLEMENTATION [26]

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

and architectures are discussed. We discuss our results and compare them to the counterparts in Sec. IV. Finally, we conclude the paper in Sec. V.

II. PRELIMINARIES

A. Symbol Definition

To make the paper more readable, Table I provides the list of notations used in this paper. 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 $n = 2^{n'-1}$ is the dimension and q is the prime modulo.

B. Kyber Algorithms

Kyber [26] is an IND-CCA secure KEM based on hardness assumptions over module learning with errors (Module-LWE) [27]. NIST has recently announced the 3rd round PQC standardization candidates, and Kyber was among the chosen algorithms as a finalist [2]. Kyber provides three post-quantum security levels, and its parameter sets are reported in Table II.

Kyber cryptosystem uses a uniformly random ring element ρ. The Kyber KEM is defined as follows where sk stands for secret key, pk for public key, and ct for ciphertext:

- KeyGen(): This function returns (sk, pk) by choosing $\mathbf{s}$ and $\mathbf{e}$ from a binomial sampling, and $\hat{\mathbf{A}}$ from a uniform distribution. $pk = (\rho, \hat{\mathbf{t}})$ and $sk = \hat{\mathbf{s}}$ where $\hat{\mathbf{t}} = \hat{\mathbf{A}} \circ \hat{\mathbf{s}} + \hat{\mathbf{e}}$.
- Enc(pk, m, μ): Using seed of μ, a binomial sampling is employed to choose $\mathbf{r}$, $\mathbf{e}_1$, and e_2. Furthermore, $\hat{\mathbf{A}}^T$ is sampled from a uniform distribution. Computing of $\mathbf{u} = \text{INTT}(\hat{\mathbf{A}}^T \circ \hat{\mathbf{r}}) + \mathbf{e}_1$ and $v = \text{INTT}(\hat{\mathbf{t}}^T \circ \hat{\mathbf{r}}) + e_2 + m$ construct the ciphertexts such that $ct = (\text{Compress}(\mathbf{u}), \text{Compress}(v))$.
- Dec(sk, ct): Message m is computed such that $m = \text{Compress}(v - \text{INTT}(\hat{\mathbf{s}}^T \circ \hat{\mathbf{u}}))$, while $\mathbf{u}$ and v are extracted from ct.

1) Keccak: The most performance-critical part of the software implementation is the Keccak core based on the profiled cycle counts presented in [3], [7]. In fact, more than half of the reported clock cycles in SW and HW/SW benchmarking are used to compute Keccak. However, this core can be accelerated in a pure hardware architecture since Keccak is a hardware-friendly design of SHA.

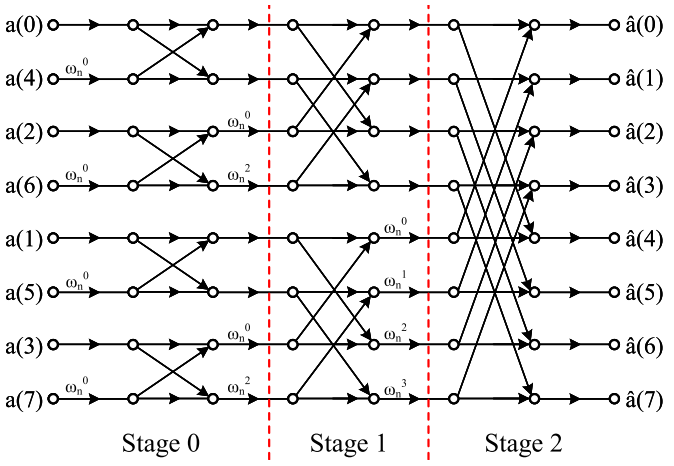

Fig. 1. 8-point NTT butterfly dataflow [28].

2) Sampling Units: The rejection sampling generates a matrix from the uniform distribution, while the accepted samples are smaller than q. The public matrix $\hat{\mathbf{A}}$ is sampled directly in the NTT domain. In the updated Kyber v3 specification the rejection probability calculated as $1 - q/2^{\lceil \log(q) \rceil}$ is increased from 3.48% to 18.7%.

Noise sampling is performed from a centered binomial distribution (CBD) based on the subtraction of the Hamming weights of the two η-bit chunks. Let β be the Keccak output, the coefficients are computed as follows:

$$e_i = \sum_{j=0}^{j=\eta-1} \beta_{2i\eta+j} - \sum_{j=0}^{j=\eta-1} \beta_{2i\eta+\eta+j} \tag{1}$$

which turns uniformly distributed samples into binomial distribution. According to Table II, in Kyber-512 architecture, two different samplers are implemented, i.e., $\eta = 2$ and $\eta = 3$, while binomial sampling units in Kyber-768 and Kyber-1024 work only with $\eta = 2$.

3) NTT and Multiplication: The centerpiece of KEM is NTT which is a fast Fourier transform (FFT) applied in a finite field. Fig. 1 illustrates the butterfly diagram for 8-point NTT. Let a be a polynomial as follows:

$$a(x) = (a_0, a_1, \ldots, a_{255}) \in \mathcal{R}_q \tag{2}$$

NTT(a) is defined as $\hat{a} = (\hat{a}_0 + \hat{a}_1 X, \hat{a}_2 + \hat{a}_3 X, \ldots, \hat{a}_{254} + \hat{a}_{255} X)$ such that $\hat{a}_{2i} = \sum_{j=0}^{127} a_{2j} \zeta^{(2\text{br}_7(i)+1)j}$ and $\hat{a}_{2i+1} = \sum_{j=0}^{127} a_{2j+1} \zeta^{(2\text{br}_7(i)+1)j}$, where $\zeta = 17$ is the first primitive 256-th root of unity modulo q, and br_7 is the bit reversal function. The pseudo-code of the iterative NTT is shown in Algorithm 1. The INTT is similar to NTT, while ω_n^{-1} is used instead of ω_n, and the resulting coefficients of $a(x)$ is divided by n.

However, the original computing of NTT and INTT needs the pre-processing and the post-processing, respectively. A point-wise multiplication includes 128 multiplications of polynomial of degree 2 modulo $X^2 - \zeta^{2\text{br}_7(i)+1}$.

Authorized licensed use limited to: University of South Florida. Downloaded on March 02,2022 at 23:49:46 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1 Iterative In-Place NTT Algorithm Based on Cooley-Tukey Butterfly [25]

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 form 1 to l **do**
  $m = 2^{l-i}$
  for j from 0 to $2^{i-1}-1$ **do**
    $W \leftarrow \omega_n^{1+j}$
    for k from 0 to $m-1$ **do**
      $T \leftarrow W \cdot \hat{a}[2 \cdot j \cdot m + k + m] \bmod q$
      $U \leftarrow \hat{a}[2 \cdot j \cdot m + k]$
      $\hat{a}[2 \cdot j \cdot m + k] = U + T \bmod q$
      $\hat{a}[2 \cdot j \cdot m + k + m] = U - T \bmod q$
    end for
  end for
end for
return $\hat{a}(x)$

Algorithm 2 Barrett Reduction Modulus $q = 3,329$ [29]

Input: $q = 3,329$, $m = \frac{2^{24}}{q} = 5,039$, $x \in [0, q^2)$
Output: $z = x \bmod q$

$u \leftarrow x \cdot m$
$u \leftarrow u \gg 24$
$u \leftarrow x - u \cdot q$
$v = u - q$
if $v \geq 0$ **then**
  $z = v$
else
  $z = u$
end if
return z

The matrix-vector multiplication $\hat{\mathbf{A}} \circ \hat{\mathbf{s}}$ in NTT domain for Kyber-512 is shown in (3) while a point-wise multiplication $\hat{\mathbf{A}}_{j,i} \circ \hat{\mathbf{s}}_i$ can be performed as shown in (4).

$$\hat{\mathbf{A}} \circ \hat{\mathbf{s}} = \begin{bmatrix} \hat{\mathbf{A}}_{00} & \hat{\mathbf{A}}_{01} \\ \hat{\mathbf{A}}_{10} & \hat{\mathbf{A}}_{11} \end{bmatrix} \circ \begin{bmatrix} \hat{\mathbf{s}}_0 \\ \hat{\mathbf{s}}_1 \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{A}}_{00} \circ \hat{\mathbf{s}}_0 + \hat{\mathbf{A}}_{01} \circ \hat{\mathbf{s}}_1 \\ \hat{\mathbf{A}}_{10} \circ \hat{\mathbf{s}}_0 + \hat{\mathbf{A}}_{11} \circ \hat{\mathbf{s}}_1 \end{bmatrix} \tag{3}$$

$$\begin{aligned}(\hat{a}_{j,2i} + \hat{a}_{j,2i+1}X) \cdot (\hat{s}_{2i} + \hat{s}_{2i+1}X) \\ = (\hat{a}_{j,2i}\hat{s}_{2i} + \hat{a}_{j,2i+1}\hat{s}_{2i+1}\zeta^{2\text{br}_7(i)+1}) \\ + (\hat{a}_{j,2i}\hat{s}_{2i+1} + \hat{a}_{j,2i+1}\hat{s}_{2i})X\end{aligned} \tag{4}$$

Operation in polynomial multiplication should be reduced with respect to the prime q. Although in the C reference implementation both Montgomery and Barrett reduction are employed, from a resource sharing optimization point of view, we focus on Barrett reduction as described in Algorithm 2 to avoid the cost of Montgomery domain conversion.

To conclude, we outlined the most time-consuming operations that are performed during KEM. These operations are composed of several basic computations, including hashing, polynomial generation, addition, subtraction, and multiplication. Dedicated architecture can be implemented to accelerate corresponding operations in hardware.

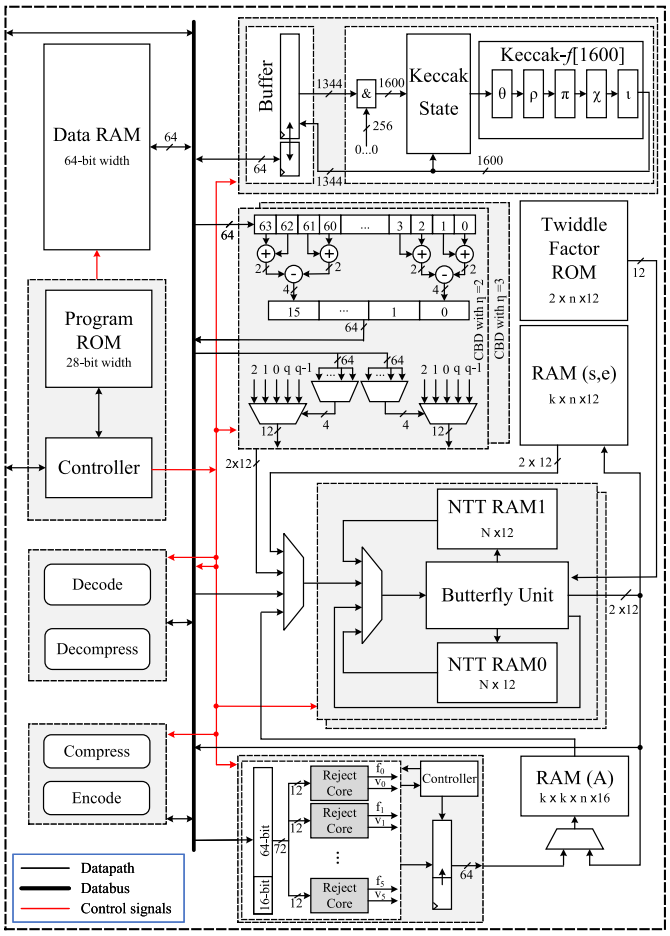

Fig. 2. Top-level architecture of Kyber KEM. The CBD core with $\eta = 3$ is implemented only in Kyber-512.

III. High-Speed Kyber Architecture

The top-level architecture of Kyber is designed and presented in Fig. 2.

A. High-Level Architecture

Full HW methodology enhances the performance of architecture over a HW/SW co-design scheme at the cost of a longer design cycle, killing the flexibility, and demands customized data paths for different protocol-level operations. However, using an instruction-set processor makes the design smaller, simpler, slower, and more controllable/programmable. A customized instruction-set can be a plausible option to achieve fine-tuned hardware acceleration with a low to moderate logic overhead. In order to implement a full HW architecture, cascading computation units in a customized data flow reduces the required latency significantly while the design becomes inflexible. In this paper, we implement all computation blocks in hardware; meanwhile, our implementation remains flexible to be extended, which is vital for a fast evolving field like PQC despite existing HW architecture.

Authorized licensed use limited to: University of South Florida. Downloaded on March 02,2022 at 23:49:46 UTC from IEEE Xplore. Restrictions apply.

TABLE III
PROPOSED INSTRUCTION FOR HASHING

Instruction	Description
`RST_Keccak`	Reset the Keccak buffer
`EN_Keccak`	Enable Keccak
`PD_Keccak A`	Padding Keccak hash for type A
`LDKeccak_CONST #<A>, B`	Load Keccak buffer with value A in B-width
`LDKeccak_MEM A, B`	Load Keccak buffer with address A in B-width

To enhance the proposed architecture from a flexibility point of view, we design 20 different customized high-level instruction codes to perform the protocol. In particular, each line of the program ROM is 25-bit wide: 5 bits for instruction code and two 10 bits for operand addresses. The instruction memory is located within the controller and stores instructions for all required operations, including arithmetic, Keccak, and various memory operations. For example, Table III summarizes our proposed hashing instructions for different hash types. As one can see, our instructions can be easily used for integration with classic cryptosystems, e.g., Ed448 digital signature scheme [30], in a hybrid architecture, which is beyond of this work. The data memory can share data with other modules through a databus handled by the controller. To perform KEM, the required parameters should be pre-loaded into the memory.

B. Keccak Core

Keccak unit is configured to perform four functions, including SHA3-256, SHA3-512, SHAKE-128, and SHAKE-256 during KEM. To design a high-performance core, we modify the high-speed core implementation of the Keccak provided by the Keccak team [31]. We develop a dedicated buffer for interfacing with the Keccak core. This dedicated buffer read/write data in 64-bit width from/to the memory unit. The buffer length is adjusted to the most extended required data, i.e., 1344-bit for SHAKE-128. Therefore, the buffer interfacing needs a maximum of 21 cycles, which can be handled during the Keccak sponge function computation, i.e., 24 cycles.

C. Rejection Sampling

Since the 64-bit data path can be matched with the Keccak core, the rejection data path is set to 64 bits. To design a high-performance rejection core, we implement six parallel cores in this module fed by Keccak results. Therefore, a buffer should be added to store the accepted samples. When the number of buffered samples is more than three, the 64 bits of the buffer, i.e., four accepted samples, are stored in the RAM.

As shown in Fig. 2, a 64-bit word is read from memory. Since 64-bit input is not a multiple of a 12-bit integer, the input buffer is extended to 80-bit to store some parts of input for the next cycles. In the first cycle, only four samples are generated in parallel, and 16 bits of the input are postponed to the next cycle. In the second cycle, all six cores work on 72 bits of the buffer, of which 16 bits are kept from the first iteration, and 56 bits are extracted from the second input. Hence, 8 bits of the input are postponed to concatenate with 64 bits of the third cycle processed with six rejection cores. A specific flag for each core shows whether the input is valid or not.

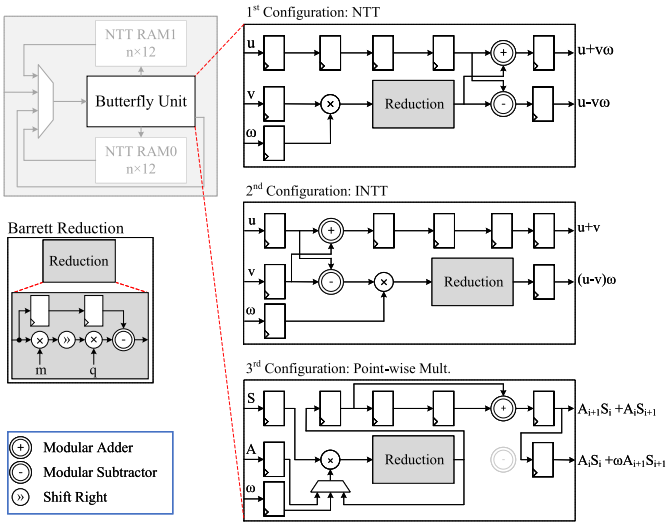

Fig. 3. Reconfigurable Butterfly Architecture.

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 Keccak core.

D. Binomial Sampling

Fig. 2 illustrates the datapath of the binomial sampler. Since this module is inherently lightweight, we implement 16 parallel combinational cores. Then, 16 consecutive samples are generated in parallel and stored in a buffer register. Although the resulting samples, which are in $[-\eta, \eta]$, can be presented in 3-bit, we use 4-bit representation to simplify the addressing. The main difference in implementing CBD core with $\eta = 3$ is an input buffer to keep data for concatenating with the input in the next cycles. In this mode, three consecutive 64-bit words are read to generate 32 samples in two words.

E. Butterfly Unit

The main configurations of our butterfly unit are detailed in Fig. 3. We employ hand-crafted resource sharing techniques to implement this core with optimized resources. There is only one modular multiplier in our butterfly architecture. In addition, we use only one reduction unit in the middle of the butterfly operation and employ a modular adder/subtractor in the proposed configurations. Hence, implementing Montgomery reduction requires more resources due to converting back from that domain and demands more clock cycles. Moreover, our proposed modular reduction is constant-time and takes two cycles, as illustrated in Fig. 3. As one can see, the architecture is pipelined to avoid any delay in butterfly operation.

1) Speeding up the NTT/INTT: An n-point NTT requires $n/2$ independent butterfly operations per stage. As a result, the naive implementation of polynomial multiplications requires 4,352 modular multiplications, of which $2 \times (7 \times 128 + 256) = 2,304$ modular multiplications for twice performing NTT, $5 \times 128 = 640$ modular multiplications for point-wise multiplication, and $7 \times 128 + 2 \times 256 = 1,408$

Authorized licensed use limited to: University of South Florida. Downloaded on March 02,2022 at 23:49:46 UTC from IEEE Xplore. Restrictions apply.

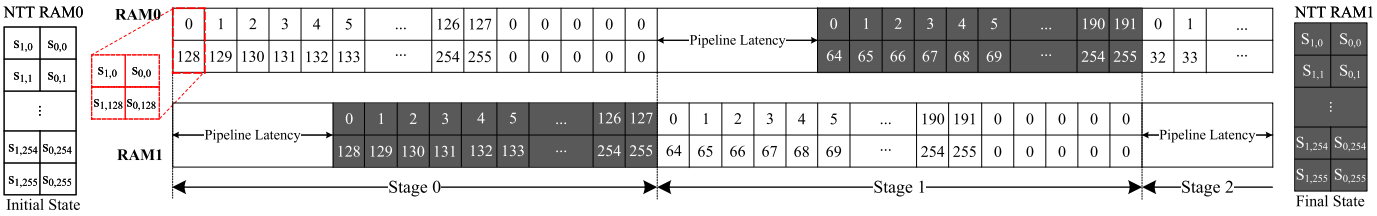

Fig. 4. The proposed address flow of our NTT memory architecture in the first two stages. (Butterfly inputs are in white and outputs are in black).

modular multiplications for INTT are required. To avoid the bit-reverse permutation in Algorithm 1, two different butterfly configurations, i.e., CT and GS, are required for NTT and INTT, respectively, as follows:

$$f.g = \text{INTT}^{\text{GS}}(\text{NTT}^{CT}(f) \circ \text{NTT}^{CT}(g)). \tag{5}$$

To be consistent with standard software implementation, the input polynomials in normal order are transformed to the NTT domain in bit-reverse order employing CT configuration, while twiddle factors are absorbed in bit-reversed order. The point-wise multiplication is performed in bit-reverse order and transformed back using GS configuration in normal order. However, the required twiddle factors are absorbed in the bit-reversed order.

We observe that an efficient implementation of point multiplication requires 3,584 modular multiplications reducing 18% complexity compared to the naive implementation. According to Fig. 3, for NTT operation, the butterfly is arranged based on CT configuration, while in INTT, it is reconfigured to match with the GS configuration. In NTT/INTT, when the pipeline is fulfilled, the butterfly unit can read and write two data inputs and outputs in each clock cycle.

The most crucial bottleneck in implementing NTT core is memory access because memory access patterns change during each operation stage [15], [32]. Therefore, designing efficient memory management is critical to avoid memory conflicts and achieve high throughput. On the other hand, memory bandwidth limits the efficiency of the butterfly operation. Hence, we use two memory units to provide double bandwidth during NTT operation to reduce latency. In the first round, the results are stored in *NTT RAM 0*. After completing the first round, the input coefficients are read from *NTT RAM 0,* and the butterfly outputs are stored in *NTT RAM 1*. This scenario is repeated for seven rounds until NTT is computed.

In this method, two coefficients are fetched from the first RAM block at a time and fed into a butterfly unit. Then, the butterfly output will be prepared and written into the second RAM block after pipelined stages, i.e., five cycles. Employing the ping-pong strategy, after 128 cycles, all coefficients are fed into the butterfly core, and the five additional cycles are required to complete a round of NTT/INTT computation. In the next round, the input coefficients are fetched from the second RAM block, and the outputs are stored in the first RAM block. This computation will be continued to complete all seven required rounds of NTT. To optimize the memory utilization in this method, different vectors are stored in the same RAM bock. For example, the $\mathbf{s}_0$ and $\mathbf{s}_1$ are located in the same memory, where in each address the lower column stores $\mathbf{s}_0$ and the higher column stores $\mathbf{s}_1$ coefficients. In each clock cycle, two addresses of memory (e.g., i and j) are read which contains four coefficients, i.e., $\mathbf{s}_{0,i}$ and $\mathbf{s}_{1,i}$ from address i, and $\mathbf{s}_{0,j}$ and $\mathbf{s}_{1,j}$ from address j. Then, $\mathbf{s}_{0,i}$ and $\mathbf{s}_{0,j}$ are fed into the first butterfly, while $\mathbf{s}_{1,i}$ and $\mathbf{s}_{1,j}$ are used by the second core. The results of these cores will be stored in the same fashion in the second RAM. Fig. 4 shows the address flow of our proposed NTT architecture using RAM0 and RAM1.

To implement a highly parallel architecture, we implement multiple butterfly units matched with the number of polynomial vectors in $\mathbf{s}$, i.e., two, three, and four units for Kyber-512, Kyber-768, and Kyber-1024, respectively.

Our first method reduces the NTT execution time from $\frac{N}{2}\log_2\frac{N}{2} + 2N$ to $\frac{N}{2}\log_2\frac{N}{2}$ compared with the naive implementation. In our second method, we take advantage of the NTT definition in the Kyber scheme to perform two independent NTT computations for odd and even coefficients. Hence, we employ two butterfly cores in parallel to computes NTT, which halves execution time to $\frac{N}{2}\log_2\frac{N}{4}$. In this method, each address of memory stores two consecutive coefficients, i.e., $\mathbf{s}_{i,2j}$ and $\mathbf{s}_{i,2j+1}$. Then, two addresses of memory are fed into two butterfly cores where contains 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. So, $\mathbf{s}_{i,2j}$ and $\mathbf{s}_{i,2k}$ are used for the first butterfly, which are independently processed form $\mathbf{s}_{i,2j+1}$ and $\mathbf{s}_{i,2k+1}$ in the second core. Similar to the previous method, the results should be stored similarly in the second RAM. Although this method does not improve the efficiency due to doubling the resources to halve the latency, it can accelerate the computations to target high-performance architectures.

2) Optimizing Point-Wise Multiplication: To implement an optimized high-throughput point-wise multiplication core, we use a specific memory pattern for matrix $\hat{\mathbf{A}}$ coefficients. In our proposed memory pattern for $\hat{\mathbf{A}}$, four consecutive coefficients are stored in pairs, i.e., ($\hat{\mathbf{A}}_{\mathbf{00}}(3)$, $\hat{\mathbf{A}}_{\mathbf{00}}(2)$, $\hat{\mathbf{A}}_{\mathbf{00}}(1)$, $\hat{\mathbf{A}}_{\mathbf{00}}(0)$), . . . ,($\hat{\mathbf{A}}_{\mathbf{11}}(255)$, $\hat{\mathbf{A}}_{\mathbf{11}}(254)$, $\hat{\mathbf{A}}_{\mathbf{11}}(253)$, $\hat{\mathbf{A}}_{\mathbf{11}}(252)$). Further, two parallel butterfly cores are employed to accelerate the polynomial multiplication. The number of the pipelined stages is set to five to design a high-throughput architecture for point-wise multiplication, i.e., 4-coefficient per 5-cycle. In other words, based on detailed scheduling and our proposed memory scheme, this design results in higher throughput while limits the maximum operating frequency. It is observed that the path from reduction output to the multiplier is the critical path. Nevertheless, increasing the pipeline latency improves the critical path

Authorized licensed use limited to: University of South Florida. Downloaded on March 02,2022 at 23:49:46 UTC from IEEE Xplore. Restrictions apply.

delay at the cost of decreasing the point-wise multiplication throughput.

Let $\hat{\mathbf{R}}_{\mathbf{00}} = \hat{\mathbf{A}}_{\mathbf{00}} \circ \hat{\mathbf{s}}_{\mathbf{0}}$; hence, based on (4), the $\hat{\mathbf{R}}_{\mathbf{00}}$ coefficients can be computed as follows:

$$\hat{\mathbf{R}}_{\mathbf{00}}(2i) = \zeta_i \hat{\mathbf{A}}_{\mathbf{00}}(2i+1)\hat{\mathbf{s}}_{\mathbf{0}}(2i+1) + \hat{\mathbf{A}}_{\mathbf{00}}(2i)\hat{\mathbf{s}}_{\mathbf{0}}(2i) \quad (6)$$

$$\hat{\mathbf{R}}_{\mathbf{00}}(2i+1) = \hat{\mathbf{A}}_{\mathbf{00}}(2i+1)\hat{\mathbf{s}}_{\mathbf{0}}(2i) + \hat{\mathbf{A}}_{\mathbf{00}}(2i)\hat{\mathbf{s}}_{\mathbf{0}}(2i+1) \quad (7)$$

Hence, we use the first core for the $\hat{\mathbf{R}}_{\mathbf{00}}(4i)$ and $\hat{\mathbf{R}}_{\mathbf{00}}(4i+1)$, and the second core works on $\hat{\mathbf{R}}_{\mathbf{00}}(4i+2)$ and $\hat{\mathbf{R}}_{\mathbf{00}}(4i+3)$. Operations in each step is described for a core as follows:

Step 1: $\hat{\mathbf{s}}_{\mathbf{0}}(2i+1)$ and ζ_i are read from NTT memory and twiddle factor ROM memory to perform modular multiplication, respectively.

Step 2: $\hat{\mathbf{s}}_{\mathbf{0}}(2i)$ is multiplied by $\hat{\mathbf{A}}_{\mathbf{00}}(2i)$. Furthermore, the previous multiplication result is passed into the modular reduction unit.

Step 3: $\hat{\mathbf{s}}_{\mathbf{0}}(2i)$ is multiplied by $\hat{\mathbf{A}}_{\mathbf{00}}(2i+1)$.

Step 4: The first step result after reduction is multiplied by $\hat{\mathbf{A}}_{\mathbf{00}}(2i+1)$.

Step 5: The second term of $\hat{\mathbf{R}}_{\mathbf{00}}(2i+1)$, i.e., $\hat{\mathbf{A}}_{\mathbf{00}}(2i)$ and $\hat{\mathbf{s}}_{\mathbf{0}}(2i+1)$ are multiplied. The reduced result of step 2, i.e., $\hat{\mathbf{A}}_{\mathbf{00}}(2i)\hat{\mathbf{s}}_{\mathbf{0}}(2i)$, is entered into the pipeline stages.

Steps 6-7: The reduction outputs, i.e., $\hat{\mathbf{A}}_{\mathbf{00}}(2i+1)\hat{\mathbf{s}}_{\mathbf{0}}(2i)$ and $\zeta_i\hat{\mathbf{A}}_{\mathbf{00}}(2i+1)\hat{\mathbf{s}}_{\mathbf{0}}(2i+1)$, are entered sequentially into the pipeline stages. Moreover, the next coefficients are read from the memories to start from Step 1.

Step 8: The modular addition computes $\zeta_i\hat{\mathbf{A}}_{\mathbf{00}}(2i+1)\hat{\mathbf{s}}_{\mathbf{0}}(2i+1) + \hat{\mathbf{A}}_{\mathbf{00}}(2i)\hat{\mathbf{s}}_{\mathbf{0}}(2i)$. Furthermore, $\hat{\mathbf{A}}_{\mathbf{00}}(2i)\hat{\mathbf{s}}_{\mathbf{0}}(2i+1)$ is passed from the reduction unit into the pipeline stages.

Step 9: The previous addition result, i.e., $\hat{\mathbf{R}}_{\mathbf{00}}(2i)$, is buffered in the next register, while the modular addition computes $\hat{\mathbf{A}}_{\mathbf{00}}(2i+1)\hat{\mathbf{s}}_{\mathbf{0}}(2i) + \hat{\mathbf{A}}_{\mathbf{00}}(2i)\hat{\mathbf{s}}_{\mathbf{0}}(2i+1)$.

Step 10: The $\hat{\mathbf{R}}_{\mathbf{00}}(2i)$ and $\hat{\mathbf{R}}_{\mathbf{00}}(2i+1)$, which are already buffered in the output registers, are stored in the memory.

Since the memory $\hat{\mathbf{A}}$ includes four coefficients per address, the addition between $\hat{\mathbf{A}}_{\mathbf{00}} \circ \hat{\mathbf{s}}_{\mathbf{0}}$ and $\hat{\mathbf{A}}_{\mathbf{01}} \circ \hat{\mathbf{s}}_{\mathbf{1}}$ can be performed by a 64-bit addition. In the described scenario, one port of the memory is always in read mode to feed the cores. The second port is used for accumulating the results.

F. Scalability

The proposed architecture for NTT computation employing two butterfly cores for Kyber-512 achieves high-performance results with reasonable resource utilization. However, different hardware resource utilization can be explored to achieve a desirable area-time trade-off from various optimization perspectives. For example, to reduce the required cycles, the number of butterfly cores can be increased to 4 cores. However, the resources can be saved if only one butterfly core is implemented at the cost of increasing the total latency. It should be noted that increasing the number of butterfly cores changes the memory access pattern, and some modifications should be considered to feed all cores. Hence, a high-performance design requires complex memory access management to reduce the access overhead.

Besides, a high-performance Keccak core occupies almost 25% of the total area. We can implement different architectures of this core and achieve scalability through area versus latency trade-offs.

This architecture can be easily scaled to match the upper or lower security level. To scale up the architecture, the same structure can be applied, while the number of butterfly cores should increase. Moreover, the depth of *Data RAM* and *RAM(A)* needs to be increased. The main difference between these architectures is using two separate CBD circuits for Kyber-512, which causes more resources to provide a dedicated sampler. Hence, a general core utilizing the most up security level resources with additional CBD core for $\eta = 3$ can be used to provide a scalable Kyber cryptosystem.

TABLE IV

FPGA IMPLEMENTATION RESULTS FOR OUR KECCAK, BINOMIAL, AND REJECTION CORES AND COMPARISON WITH STATE-OF-THE-ART

Work	Method	Platform	Area					Freq	Cycles
			#LUTs	#FFs	#Slices	#DSPs	#BRAM	[MHz]	[CCs]
Keccak-f[1600]									
[33]	SW	Cortex-M4	-	-	-	-	-	100	12,969
[34]	HW	Virtex-6	359	107	91	0	0	311	1,665
[6]	HW/SW	Artix-7	5,784	1,605	1,716	0	0	25	24
[35]	HW/SW	Zynq-7000	10,435	4,225	NA	0	0	-	12
This Work	**HW**	**Artix-7**	**4,405**	**1,629**	**1,825**	**0**	**0**	**115**	**24**
Binomial sampler ($n = 256$, $k = 4$)									
[36]	SW	Cortex-M4	-	-	-	-	-	100	52,603
[8]	HW/SW	Zynq-7000	124	0	NA	0	0	-	9,500†
[6]‡	HW/SW	Artix-7	784	47	334	0	0.5	25	505
This Work	**HW**	**Artix-7**	**88**	**96**	**45**	**0**	**0**	**115**	**68**
Rejection sampler ($n = 256$)									
[36]	SW	Cortex-M4	-	-	-	-	-	100	60,433
[8]	HW/SW	Zynq-7000	NA	NA	NA	NA	NA	-	22,414†
[6]‡	HW/SW	Artix-7	784	47	334	0	0.5	25	461
This Work	**HW**	**Artix-7**	**2,147**	**659**	**610**	**0**	**2**	**115**	**0 (432§)**

†[8] only reports latency of sampling one polynomial.
‡Total area for sampling units.
§Total CCs considering cycles absorbed by Keccak core.

TABLE V

ASIC IMPLEMENTATION RESULTS FOR OUR KECCAK, BINOMIAL, AND REJECTION CORES AND COMPARISON WITH STATE-OF-THE-ART

Component	Work	Parameters	Area #GEs	Freq [MHz]	Cycles [CCs]
Keccak-f[1600]	[9]	-	26K	300	24
	This Work	-	24K	**200**	**24**
Binomial sampler	[9]†	$k = 4$	325K	300	411
	This Work	$k = 4$	**1K**	**200**	**68**
Rejection sampler	[9]†	$q = 3,329$	325K	300	34
	This Work	$q = 3,329$	**13K**	**200**	**0 (432‡)**

†Total area for sampling units.
‡Total CCs considering cycles absorbed by Keccak core.

IV. EXPERIMENTAL RESULTS AND COMPARISON

In this section, we provide implementation results and compare them to the counterparts available in the open literature. Along with the fact that the implementations employ different platforms, a fair and meaningful discussion or comparison of different designs and implementations with previous work is not straightforward. Nevertheless, we like to put our results in the context with existing implementations to allow the reader a quick overview of other designs and architectures.

A. Results for Keccak and Polynomial Sampling

Tables IV and V report the required FPGA and ASIC resources and latency specifications for the Keccak, the CBD,

Authorized licensed use limited to: University of South Florida. Downloaded on March 02,2022 at 23:49:46 UTC from IEEE Xplore. Restrictions apply.

TABLE VI
FPGA IMPLEMENTATION RESULTS FOR OUR NTT CORE AND COMPARISON WITH STATE-OF-THE-ART ($n = 256$)

Work	Method	Platform	Parameter	Area					Freq	Cycles			NTT Area×Cycle Ratio
				#LUTs	#FFs	#Slices	#DSPs	#BRAMs	[MHz]	NTT [CCs]	INTT [CCs]	Point-wise Mult. [CCs]	
Botros *et al.* [3]	SW	Cortex-M4	$q = 3,329$	-	-	-	-	-	100	7,725	9,347	27,873	-
Karabulut *et al.* [19]	HW/SW	VIRTEX-7	$q = 3,329$	417	462	NA	0	0	-	43,756	NA	NA	53.9
Fritzmann *et al.* [8]	HW/SW	Zynq-7000	$q = 3,329$	2,908	170	NA	9	0	-	1,935	1,930	NA	16.6
Alkim *et al.* [7]	HW/SW	Artix-7	$q = 3,329$	NA	NA	NA	NA	NA	59	6,868	6,367	2,395	-
Banerjee *et al.* [6]	HW/SW	Artix-7	$q = 7,681$	2,983	0	957	0	11	25	1,289	NA	NA	11.4
Huang *et al.* [10]	HW	Artix-7	$q = 3,329$	NA	NA	NA	NA	NA	155	1,834	NA	NA	-
Fritzmann *et al.* [20]	HW	Zynq-7000	$q = 7,681$	980	395	NA	26	2	-	2,056	NA	NA	6.0
Chen *et al.* [14]	HW	Artix-7	$q = 7,681$	442	237	NA	1	5	130	2,055	NA	7,197	2.7
Chen *et al.* [37]	HW	Artix-7	$q = 7,681$	479	472	NA	1	2	246	4108	NA	NA	74.2
Zhang *et al.* [38]	HW	Artix-7	$q = 3,329$	609	640	NA	2	4	257	490	490	NA	0.85
Bisheh-Niasar *et al.* [13]	HW	Artix-7	$q = 3,329$	801	717	NA	4	2	222	324	324	NA	0.74
This work	**HW**	**Artix-7**	$q = 3,329$	**360**	**145**	**187**	**3**	**2**	**115**	**940**	**1203**	**1,289**	**1.0**
This work	**HW**	**Artix-7**	$q = 3,329$	**737**	**290**	**371**	**6**	**4**	**115**	**474**	**602**	**1,289**	**1.0**

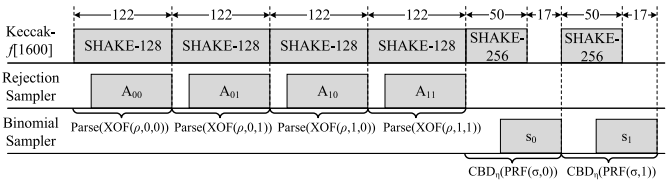

Fig. 5. Proposed Scheduling for Sampling Units in Kyber-512.

and rejection sampling cores in our design and other state-of-the-art implementations. As one can see, the software implementation of Keccak runs in thousands of clock cycles, which can be significantly accelerated while implemented in hardware. A lightweight Keccak core presented in [34] uses 359 LUTs to perform a round of Keccak-f[1600] in 1,665 cycles, while in [35], the authors proposed an architecture performing in 12 clock cycles at the cost of almost 10k LUTs. In our proposed design, a Keccak-f[1600] is performed in 24 cycles at the cost of 4.4k LUTs or 24k GEs. Additionally, decreasing the latency of the Keccak core does not considerably improve the performance due to interfacing cost, which requires 21 clock cycles for a 1,344-bit output.

The reported results show that the performance of our binomial and rejection sampler outperform sampling units of previous works [6], [8], [9]. Our proposed implementation takes advantage of parallel computations between our sampling units and Keccak core. Our binomial sampler requires 68 clock cycles for generating four polynomials of degree 256, i.e., 1,024 samples. The rejection sampler in our proposed scheme works simultaneously with the Keccak core. Therefore, its required latency for generating matrix $\hat{\mathbf{A}}$ with 1,024 samples, i.e., 432 clock cycles, is totally absorbed. This unit, with 16 parallel cores, occupies almost 2k LUTs in FPGA or 13k GEs on ASIC platform.

Fig. 5 shows the proposed scheduling for sampler units in Kyber-512. Rejection sampler works parallel by Keccak core, and therefore its latency, i.e., 108 cycles, is absorbed completely. The accepted samples will be stored in RAM(A), shown in Fig. 2. For a binomial sampling of a polynomial of degree 256 with $\eta = 3$, two rounds of Keccak are required. Each round of Keccak result is processed in 17 cycles by the binomial sampler. However, processing the second round result cannot be parallelized by the next CBD due to memory bandwidth limitation.

TABLE VII
ASIC RESULTS FOR NTT AND COMPARISON WITH STATE-OF-THE-ART

Work	Parameter	Area	Freq	Cycles		
				NTT	INTT	Point-wise
		[kGEs]	[MHz]	[CCs]	[CCs]	Mult. [CCs]
Banerjee *et al.*[6]	$q = 7,681$	NA	72	1,289	NA	NA
Xin *et al.* [9]	$q = 3,329$	512	300	41	NA	NA
Fritzmann *et al.* [20]	$q = 7,681$	14	25	2,056	NA	NA
This work	$q = 3,329$	1.9	200	**940**	**1203**	**1,289**
This work	$q = 3,329$	3.8	200	**474**	**602**	**1,289**

B. Results for the Butterfly Core

Tables VI and VII report the required FPGA and ASIC hardware resources and latency specifications for our proposed butterfly unit in different configurations, i.e., NTT, INTT, and point-wise multiplication, including other state-of-the-art implementations. We remark that a more technology-independent measurement is the required cycle. Thus, for efficiency comparison between different proposed NTT architectures, efficiency can be computed by the required clock cycles×area.

Our pipelined architecture employing our first method requires 133 cycles for performing one round of 256-point NTT; hence, a full NTT with seven rounds requires 940 cycles. Computing INTT requires 263 additional clock cycles for post-processing. Moreover, point-wise multiplication between two polynomials of degree 256 requires 1,289 clock cycles. Our proposed architecture is significantly smaller compared to previous best works occupying 360 LUTs, 145 FFs, 187 Slices, 3 DSPs, and 2 BRAMs.

Besides, our second proposed method requires 474 cycles for performing a full NTT employing two parallel butterfly cores. Hence, this method results in a significant speedup by halving the cycle count compared to other NTT implementations for Kyber. Although the efficiency of both methods is the same, a trade-off between area and time can be achieved.

The authors in [19] presented a flexible NTT architecture over RISC-V, which consumes significantly greater cycles. In [7], 3-layer merged NTT for NewHope was proposed. The work of [24] and [13] implemented 2-layer merged NTT using the KRED algorithm, while this reduction algorithm needs a special prime form. In [10], Montgomery reduction was employed. From a resource sharing perspective, we use a

Authorized licensed use limited to: University of South Florida. Downloaded on March 02,2022 at 23:49:46 UTC from IEEE Xplore. Restrictions apply.

TABLE VIII
FPGA IMPLEMENTATION RESULTS AND COMPARISON WITH STATE-OF-THE-ART

Work	Method	Platform	Area					Freq	Cycles			Total Time‡	Speedup	A×T§
			#LUTs	#FFs	#Slices	#DSPs	#BRAMs	[MHz]	KeyGen [kCCs]	Encaps [kCCs]	Decaps [kCCs]	[μs]	Ratio	Ratio
Kyber-512														
Botros *et al.* [3]	SW	Cortex-M4	-	-	-	-	-	100	499	634	597	12,400	83.9	-
Banerjee *et al.* [6]	HW/SW	Artix-7	15K	3K	4K	11	14	25	75	132	142	10,960	74.1	61.8
Fritzmann *et al.* [8]	HW/SW	Zynq-7000	24K	11K	NA	21	32	-	150	193	205	-	-	-
Alkim *et al.* [7]	HW/SW	Artix-7	2K	2K	NA	5	34	59	710	971	870	31,203	211.1	23.5
Basu *et al.* [4]†	HW	Virtex-7	1,978K	194K	NA	0	0	67	-	32	43	1,119	7.6	832.1
Huang *et al.* [10]†	HW	Artix-7	89K	NA	NA	354	202	155	-	49	69	761	5.1	25.5
Xing *et al.* [12]	HW	Artix-7	7K	5K	2K	2	3	161	4	5	7	72	0.48	0.19
Dang *et al. [11]*	HW	Artix-7	12K	10K	4K	8	15	210	-	3	4	35	0.23	0.15
Bisheh-Niasar *et al.* [13]	HW	Artix-7	11K	10K	4K	8	13	200	2	2	4	31	0,20	0.14
This work	**HW**	**Artix-7**	**18K**	**5K**	**5K**	**6**	**15**	**115**	**4**	**7**	**10**	**148**	**1.0**	**1.0**
Kyber-768														
Batros *et al.* [3]	SW	Cortex-M4	-	-	-	-	-	100	974	1,113	1,059	21,720	104.1	-
Banerjee *et al.* [6]	HW/SW	Artix-7	15K	3K	4K	11	14	25	112	178	191	14,760	70.7	66.3
Fritmann *et al.* [8]	HW/SW	Zynq-7000	24K	11K	NA	21	32	-	273	326	340	-	-	-
Huang *et al.* [10]†	HW	Artix-7	167K	NA	NA	292	202	155	-	77	102	1,155	5.5	57.8
Xing *et al.* [12]	HW	Artix-7	7K	5K	2K	2	3	161	6	8	10	111	0.53	0.23
Dang *et al. [11]*	HW	Artix-7	12K	10K	4K	8	15	210	-	4	6	46	0.22	0.25
Bisheh-Niasar *et al.* [13]	HW	Artix-7	12K	10K	4K	12	14	200	3	3	5	40	0.19	0.14
This work	**HW**	**Artix-7**	**16K**	**6K**	**4K**	**9**	**16**	**115**	**7**	**10**	**14**	**209**	**1.0**	**1.0**
Kyber-1024														
Batros *et al.* [3]	SW	Cortex-M4	-	-	-	-	-	100	1,525	1,732	1,653	33,850	165.0	-
Banerjee *et al.* [6]	HW/SW	Artix-7	15K	3K	4K	11	14	25	149	223	241	18,560	90.5	84.8
Fritzmann *et al.* [8]	HW/SW	Zynq-7000	24K	11K	NA	21	32	-	350	405	425	-	-	-
Alkim *et al.* [7]	HW/SW	Artix-7	2K	2K	NA	5	34	59	2,203	2,619	2,429	85,559	417.1	52.1
Huang *et al.* [10]†	HW	Virtex-7	133K	NA	NA	548	202	192	-	107	135	1,260	6.1	51.1
Xing *et al.* [12]	HW	Artix-7	7K	5K	2K	2	3	161	9	11	14	154	0.75	0.32
Dang *et al. [11]*	HW	Artix-7	12K	12K	5K	8	15	210	-	4	6	63	0.30	0.24
Bisheh-Niasar *et al.* [13]	HW	Artix-7	13K	12K	5K	16	16	185	3	4	6	56	0.27	0.22
This work	**HW**	**Artix-7**	**16K**	**6K**	**5K**	**12**	**17**	**112**	**10**	**14**	**18**	**286**	**1.4**	**1.4**
This work	**HW**	**Virtex-7**	**16K**	**6K**	**5K**	**12**	**17**	**156**	**10**	**14**	**18**	**205**	**1.0**	**1.0**

†Different architecture for Encaps and Decaps are used.
‡Time results include the total time of both parties computations (Encaps + Decaps), as the key generation can be done offline.
§$A \times T$ is the product of total time and LUT resources, which has a unit of LUT·s.

general reduction method that can be configured for different prime values in a hybrid cryptosystem. Moreover, in [37] and [21], the bandwidth doubling technique is used for feeding the processing units. Particularly, in this work, we propose a compact reconfigurable architecture to accelerate the polynomial multiplication, which is enhanced by borrowing the compact memory implementation [25], resource sharing technique [5], [24], and doubled bandwidth scheme [14], [21].

In comparison to the SW implementations, our first method achieves a speedup factor of 8.2×, 7.7×, and 21.6× for NTT, INTT, and point-wise multiplication, respectively. Our second proposed method can also accelerate 16.3×, 15.2×, and 21.6× NTT, INTT, and point-wise multiplication, respectively. However, our proposed architecture decrease 26% (15%) the performance compared to [13] ([38]) in the HW platform, while the NTT core designed in [13] employs 4 butterfly units. It should be noted that although the design presented in [9] is faster implementing a vectorized butterfly unit, it consumes 512k GE logic gates, which is several times bigger than our proposed design. Hence, our design outperforms state-of-the-art ASIC implementations with at least 11.6× better Area×Cycles.

Note that the NTT can also be parallelized by sampling unit to reduce the total latency; however, applying this parallelization in this work results in diminishing the flexibility and increasing the required memory units. To achieve both high speed and instruction-level flexibility, we do not follow this methodology such that the design remains flexible to add or modify new instructions.

TABLE IX
ASIC RESULTS AND COMPARISON WITH STATE-OF-THE-ART

Work	Tech.	Area		Freq	Cycles			Total Time†
		Logic Gates	SRAM		KeyGen	Encaps	Decaps	
	[nm]	[kGE]	[kB]	[MHz]	[kCCs]	[kCCs]	[kCCs]	[μs]
Kyber-512								
Banerjee *et al.* [6]	40	106	40.25	72	75	132	142	3,806
Fritzmann *et al.* [8]	65	170	465	45	150	193	205	8,844
Xin *et al.* [9]	28	979	12	300	19	46	80	420
Basu *et al.* [4]	65	1,341	-	200	-	-	43	-
This work	**65**	**95**	**80**	**200**	**4**	**7**	**10**	**85**
Kyber-768								
Banerjee *et al.* [6]	40	106	40.25	72	112	178	191	5,125
Fritzmann *et al.* [8]	65	170	465	45	273	326	340	14,800
This work	**65**	**93**	**175**	**200**	**7**	**10**	**14**	**116**
Kyber-1024								
Banerjee *et al.* [6]	40	106	40.25	72	149	223	241	6,444
Fritzmann *et al.* [8]	65	170	465	45	350	405	425	18,444
Xin *et al.* [9]	28	979	12	300	40	82	136	727
This work	**65**	**104**	**190**	**200**	**10**	**14**	**18**	**155**

†Time results include the total time of both parties computations (Encaps + Decaps), as the key generation can be done offline.

C. FPGA Implementations

Our proposed architecture for different NIST security levels is synthesized with Xilinx Vivado 2019.2 and implemented on a Xilinx Artix XC7A100T-3 FPGA. All given results are obtained after place-and-route (PAR). We report the area, timing, and area-time trade-off (number of LUT×time in μs) results of the design in Table VIII. In some previous works, each DSP is considered equivalent to 100 Slices [39]. However, no single element of FPGA can be accurately expressed in terms of other elements; hence, DSP and BRAM are not considered in A. To have a fair comparison, we evaluate the

Authorized licensed use limited to: University of South Florida. Downloaded on March 02,2022 at 23:49:46 UTC from IEEE Xplore. Restrictions apply.

TABLE X
COMPARISONS WITH EXISTING FPGA-BASED PQC IMPLEMENTATIONS OF CCA-SECURE KEM SCHEMES IN NIST SECURITY LEVEL 5

Work	Platform	Protocol	Area					Freq	Cycle	Time	A×T
			#LUTs	#FFs	#Slices	#DSPs	#BRAMs	[MHz]	[KCCs]	[us]	Ratio
Roy *et al.*[15]	UltraScale+	FireSaber-CCA-KEM	24K	10K	NA	0	2	150	23	153	1.1
Elkhatib *et al.*[40]	Virtex-7	SIKEp751-CCA-KEM	20K	39K	11K	452	42	233	5,930	25,451	155.2
Banerjee *et al.*[6]	Artix-7	FrodoKEM-1344-CCA-KEM	15K	3K	4K	11	14	25	144,028	5,761,122	26,346.6
This Work	**Virtex-7**	**Kyber-1024-CCA-KEM**	**16K**	**6K**	**5K**	**12**	**17**	**156**	**32**	**205**	1

performance of the proposed design on the state-of-the-art targeted platforms, which changes performance by a factor of 1.35×, 1.4×, and 0.68× on Zynq-7000, Virtex-7, and Virtex-6 compared to Artix-7.

We compare our architecture results to the best SW design on the ARM Cortex-M4 chip, as well as the HW implementations and the HW/SW co-design. The total latency is the summation of key encapsulation and key decapsulation (Encaps + Decaps), as the key generation can be done offline. As one can see, for NIST level 1 security, our proposed scheme occupies 18k LUTs, 5k FFs, 6 DSPs, and 15 BRAMs. It also runs at 115 MHz and performs the whole Kyber protocol in 148 μs. Our design achieves a speedup factor of 83.9× and 74.1× compared to the leading counterpart in SW and HW/SW designs. Furthermore, our architecture employing the various optimization techniques is highly efficient, with area-time trade-off being about 98% improved compared to [6]. It is to be noted that the HW/SW co-design [6]–[9] is a complete design for all Kyber security levels. The same improvement can be observed in the remaining security levels. Compared to HW architecture, our proposed design consumes 5× time than our previous work [13], resulting in a greater $A \times T$ by a factor of 7. Our design is also 2× slower and 2.5× larger compared to [12]. However, this overhead comes to keep the customized instruction-set design flexible compared to highly parallel [13] or highly compact architectures [12]. The hardware specially designed to cater a scheme may fail in flexibility; thereby, this work aims to achieve both high speed and flexibility for Kyber to support extension for building a hybrid cryptosystem.

Although our implementations are constant-time, investigating side-channel analysis attacks will part of our future work.

D. ASIC Results

The ASIC implementation results of our architectures based on the 65-nm TSMC cell library using Synopsys Design Compiler are presented in this section. All the designs are synthesized with a 5ns clock period. Table IX reports the maximum clock frequency and the amount of logic cells for our proposed designs and state-of-the-art implementations. As one can see, the placed-and-routed design of our proposed Kyber-1024 consists of 104 kGE for logic and 190 KB SRAM for memory, which shows a significant speedup compared to previous works.

E. Comparison With Other Implementations

In Table X, the comparison between our proposed architecture with some existing PQC hardware implementations targeting NIST security level 5 is reported. It should be noted that due to the varying techniques of different FPGA generations, a fair comparison is actually not accurately possible.

In [15], a fast architecture of Saber is proposed using the high-speed instruction-set coprocessor on a Xilinx ZCU102 board. In this work, a non-NTT-based approach is used, taking advantage of the module power of 2 in the Saber scheme, which results in 153 μs time execution. Employing multiply-and-accumulate units provides the required trade-off between area and time for different applications. However, this design needs more hardware resources compared to ours, which results in 1.1× area-time product.

We also compare our work with FrodoKEM-1344 based on standard learning with error problem. To the best of our knowledge, there is not a pure HW work for FrodoKEM targeting security level 5; hence, the results in [6] used a HW/SW approach are reported. As one can see, the FrodoKEM scheme requires a considerable cycle compared to other PQC schemes due to performing expensive matrix-vector multiplications. Our implementation of Kyber-1024 is almost 26,000 times faster, occupying almost the same resources compared to [6].

SIKE [40] as an isogeny-based PQC scheme requires significantly more DSP resources to design parallel Montgomery multiplier architecture over a large prime. Although this scheme outperforms FrodoKEM implementation, our Kyber-1024 design shows 155 times better area-time product compared to this scheme.

It should be noted that there is a large body of work on optimizing PQC schemes on a variety of platforms. For example, the work of [21] and [28] propose the NewHope on a Xilinx XC7Z020 and Zynq-7000, respectively. The architecture of NewHope is very similar to that of Kyber; however, this scheme has not been selected to continue into the third round of NIST. In [21], a low-complexity architecture of NewHope is introduced, having a competitive performance compared to our design. Hence, taking advantage of this architecture to improve the total performance of Kyber is kept for future works.

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 [30], [41], [42].

V. CONCLUSION

The threat from large-scale quantum computers is real, and we need to act now as the deployment, integration, and migration to quantum-safe security systems take several

Authorized licensed use limited to: University of South Florida. Downloaded on March 02,2022 at 23:49:46 UTC from IEEE Xplore. Restrictions apply.

years. In this paper, we have presented an instruction-set post-quantum cryptosystem for CRYSTALS-Kyber. Our proposed architecture is synthesized for a Xilinx Artix-7 FPGA (which is a NIST recommended tool for prototype) prototype and an ASIC. Implementing efficient components, including sampling cores, NTT, and point-wise multiplication architectures, increases the performance compared to the state-of-the-art SW and HW/SW implementations. More specifically, our proposed architecture performs Kyber-512, Kyber-768, and Kyber-1024 protocols in only 148, 209, and 286 μs on a Artix-7 FPGA, respectively. Our future work will focus on the side-channel resistance and the development of countermeasures against such attacks.

ACKNOWLEDGMENT

The authors would like to thank the reviewers for their comments.

REFERENCES

[1] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proc. 35th Annu. Symp. Found. Comput. Sci.*, Santa Fe, NM, USA, Nov. 1994, pp. 124–134.

[2] *Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process*, Nat. Inst. Standards Technol., Gaithersburg, MD, USA, 2020.

[3] L. Botros, M. J. Kannwischer, and P. Schwabe, "Memory-efficient high-speed implementation of Kyber on Cortex-M4," in *Proc. 11th Int. Conf. Cryptol.*, Rabat, Morocco, Jul. 2019, pp. 209–228, 2019.

[4] K. Basu, D. Soni, M. Nabeel, and R. Karri, "NIST post-quantum cryptography a hardware evaluation study," in *Proc. IACR*, 2019, p. 47.

[5] U. Banerjee, T. S. Ukyab, and A. P. Chandrakasan, "Sapphire: A configurable crypto-processor for post-quantum lattice-based protocols," in *Proc. IACR*, vol. 4, 2019, pp. 17–61.

[6] U. Banerjee, T. S. Ukyab, and A. P. Chandrakasan, "Sapphire: A configurable crypto-processor for post-quantum lattice-based protocols (extended version)," in *Proc. IACR*, 2019, p. 1140.

[7] E. Alkim, H. Evkan, N. Lahr, R. Niederhagen, and R. Petri, "ISA extensions for finite field arithmetic accelerating Kyber and NewHope on RISC-V," in *Proc. IACR*, vol. 3, 2020, pp. 219–242.

[8] T. Fritzmann, G. Sigl, and J. Sepúlveda, "RISQ-V: Tightly coupled RISC-V accelerators for post-quantum cryptography," in *Proc. IACR*, Aug. 2020, pp. 239–280.

[9] G. Xin *et al.*, "VPQC: A domain-specific vector processor for post-quantum cryptography based on RISC-V architecture," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 8, pp. 2672–2684, Aug. 2020.

[10] Y. Huang, M. Huang, Z. Lei, and J. Wu, "A pure hardware implementation of CRYSTALS-KYBER PQC algorithm through resource reuse," *IEICE Electron. Exp.*, vol. 17, no. 17, 2020, Art. no. 20200234.

[11] V. B. Dang, F. Farahmand, M. Andrzejczak, K. Mohajerani, D. T. Nguyen, and K. Gaj, "Implementation and benchmarking of round 2 candidates in the NIST post-quantum cryptography standardization process using hardware and software/hardware co-design approaches," in *Proc. IACR Cryptol. Arch.*, 2020, p. 795.

[12] Y. Xing and S. Li, "A compact hardware implementation of CCA-secure key exchange mechanism CRYSTALS-KYBER on FPGA," in *Proc. IACR*, Feb. 2021, pp. 328–356.

[13] M. Bisheh-Niasar, R. Azarderakhsh, and M. Mozaffari-Kermani, "High-speed NTT-based polynomial multiplication accelerator for CRYSTALS-Kyber post-quantum cryptography," *Proc. IACR*, 2021, p. 563.

[14] Z. Chen, Y. Ma, T. Chen, J. Lin, and J. Jing, "Towards efficient kyber on FPGAs: A processor for vector of polynomials," in *Proc. 25th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Beijing, China, Jan. 2020, pp. 247–252.

[15] S. Sinha Roy and A. Basso, "High-speed instruction-set coprocessor for lattice-based key encapsulation mechanism: Saber in hardware," in *Proc. IACR Trans. Cryptograph. Hardw. Embedded Syst.*, Aug. 2020, pp. 443–466.

[16] Y. Zhang, C. Wang, D. E. S. Kundi, A. Khalid, M. O'Neill, and W. Liu, "An efficient and parallel R-LWE cryptoprocessor," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 5, pp. 886–890, May 2020.

[17] A. C. Mert, E. Karabulut, E. Ozturk, E. Savas, M. Becchi, and A. Aysu, "A flexible and scalable NTT hardware: Applications from homomorphically encrypted deep learning to post-quantum cryptography," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Grenoble, France, Mar. 2020, pp. 346–351.

[18] A. C. Mert, E. Karabulut, E. Ozturk, E. Savas, and A. Aysu, "An extensive study of flexible design methods for the number theoretic transform," *IEEE Trans. Comput.*, early access, Aug. 19, 2020, doi: 10.1109/TC.2020.3017930.

[19] E. Karabulut and A. Aysu, "RANTT: A RISC-V architecture extension for the number theoretic transform," in *Proc. 30th Int. Conf. Field-Program. Log. Appl. (FPL)*, Aug. 2020, pp. 26–32.

[20] T. Fritzmann and J. Sepulveda, "Efficient and flexible low-power NTT for lattice-based cryptography," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, McLean, VA, USA, May 2019, pp. 141–150.

[21] N. Zhang, B. Yang, C. Chen, S. Yin, S. Wei, and L. Liu, "Highly efficient architecture of NewHope-NIST on FPGA using low-complexity NTT/INTT," in *Proc. IACR*, Mar. 2020, pp. 49–72.

[22] T. Pöppelmann, T. Oder, and T. Güneysu, "High-performance ideal lattice-based cryptography on 8-bit ATxmega microcontrollers," in *Proc. LATINCRYPT*, Guadalajara, Mexico, Aug. 2015, pp. 346–365.

[23] P. Longa and M. Naehrig, "Speeding up the number theoretic transform for faster ideal lattice-based cryptography," in *Proc. 15th Int. Conf.*, Milan, Italy, Nov. 2016, pp. 124–139.

[24] P.-C. Kuo *et al.*, "High performance post-quantum key exchange on FPGAs," in *Proc. IACR*, 2017, p. 690.

[25] C. Du and G. Bai, "Towards efficient polynomial multiplication for lattice-based cryptography," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Montréal, QC, Canada, May 2016, pp. 1178–1181.

[26] R. Avanzi *et al.*, "CRYSTALSKyber: Algorithm specification and supporting documentation (version 3.0). submission to the NIST post-quantum cryptography standardization project," NIST Post-Quantum Cryptogr. Standardization Project, Tech. Rep., 2020.

[27] J. Bos *et al.*, "CRYSTALS-Kyber: A CCA-secure module-lattice-based KEM," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, London, U.K., Apr. 2018, pp. 353–367.

[28] Y. Xing and S. Li, "An efficient implementation of the NewHope key exchange on FPGAs," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 3, pp. 866–878, Mar. 2020.

[29] P. Barrett, "Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor," in *Proc. CRYPTO*, Santa Barbara, CA, USA, 1986, pp. 311–323.

[30] M. Bisheh-Niasar, R. Azarderakhsh, and M. M. Kermani, "Area-time efficient hardware architecture for signature based on Ed448," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 68, no. 8, pp. 2942–2946, Aug. 2021.

[31] G. Bertoni, J. Daemen, S. Hoffert, M. Peeters, and G. V. Assche, "Keccak in VHDL," Keccak Team, Tech. Rep., 2020.

[32] S. S. Roy, F. Vercauteren, N. Mentens, D. D. Chen, and I. Verbauwhede, "Compact ring-LWE cryptoprocessor," in *Proc. Cryptograph. Hardw. Embedded Syst.*, Busan, South Korea, Sep. 2014, pp. 371–391.

[33] K. Stoffelen, "Efficient cryptography on the RISC-V architecture," in *Proc. LATINCRYPT*, Santiago de Chile, Chile, Oct. 2019, pp. 323–340.

[34] B. Jungk and M. Stottinger, "Hobbit—Smaller but faster than a dwarf: Revisiting lightweight SHA-3 FPGA implementations," in *Proc. Int. Conf. ReConFigurable Comput.*, Cancun, Mexico, Nov. 2016, pp. 1–7.

[35] T. Fritzmann, U. Sharif, D. Muller-Gritschneder, C. Reinbrecht, U. Schlichtmann, and J. Sepulveda, "Towards reliable and secure post-quantum co-processors based on RISC-V," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Florence, Italy, Mar. 2019, pp. 1148–1153.

[36] M. J. Kannwischer, J. Rijneveld, P. Schwabe, and K. Stoffelen, "PQM4: Post-quantum crypto library for the ARM Cortex-M4," PQM4, Tech. Rep., 2018.

[37] Z. Chen, Y. Ma, T. Chen, J. Lin, and J. Jing, "High-performance area-efficient polynomial ring processor for CRYSTALS-kyber on FPGAs," *Integration*, vol. 78, pp. 25–35, May 2021.

[38] C. Zhang *et al.*, "Towards efficient hardware implementation of NTT for kyber on FPGAs," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2021, pp. 1–5.

[39] M. Bisheh-Niasar, R. Azarderakhsh, and M. Mozaffari-Kermani, "Cryptographic accelerators for digital signature based on Ed25519," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 7, pp. 1297–1305, Jul. 2021.

Authorized licensed use limited to: University of South Florida. Downloaded on March 02,2022 at 23:49:46 UTC from IEEE Xplore. Restrictions apply.

[40] R. Elkhatib, R. Azarderakhsh, and M. Mozaffari-Kermani, "Highly optimized Montgomery multiplier for SIKE primes on FPGA," in *Proc. IEEE 27th Symp. Comput. Arithmetic (ARITH)*, Portland, OR, USA, Jun. 2020, pp. 64–71.
[41] M. B. Niasar, R. El Khatib, R. Azarderakhsh, and M. Mozaffari-Kermani, "Fast, small, and area-time efficient architectures for key-exchange on Curve25519," in *Proc. IEEE 27th Symp. Comput. Arithmetic (ARITH)*, Portland, OR, USA, Jun. 2020, pp. 72–79.
[42] M. B. Niasar, R. Azarderakhsh, and M. M. Kermani, "Efficient hardware implementations for elliptic curve cryptography over Curve448," in *Proc. 21st Int. Conf. Cryptol.*, Indocrypt, India, Dec. 2020, pp. 228–247.

Mojtaba Bisheh-Niasar (Student Member, IEEE) received the B.Sc. degree from Amirkabir University of Technology in 2011 and the M.Sc. degree in electrical engineering from Iran University of Science and Technology in 2015. He is currently pursuing the Ph.D. degree in computer engineering with Florida Atlantic University under the supervision of Dr. Azarderakhsh. He is also a Research Assistant with I-SENSE Lab. He is a Research Intern in azure hardware security architecture (AHSA) at Microsoft, Redmond, Washington. His research interests include applied cryptography, post-quantum cryptography, and efficient implementation of cryptographic algorithms.

Reza Azarderakhsh (Member, IEEE) received the Ph.D. degree in electrical and computer engineering from Western University in 2011. He has worked at the Center for Applied Cryptographic Research and the Department of Combinatorics and Optimization, University of Waterloo. He is currently an Associate Professor with the Department of Electrical and Computer Engineering, Florida Atlantic University. His current research interests include finite field and its application, elliptic curve cryptography, isogenies on elliptic curves, and lattice-based post-quantum cryptography. He was a recipient of the NSERC Post-Doctoral Research Fellowship. He is serving as an Associate Editor for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS.

Mehran Mozaffari-Kermani (Senior Member, IEEE) received the B.Sc. degree from the University of Tehran, Iran, and the M.E.Sc. and Ph.D. degrees from the University of Western Ontario, London, Canada, in 2007 and 2011, respectively. In 2012, he joined the Department of Electrical Engineering, Princeton University, NJ, USA, as an NSERC Post-Doctoral Research Fellow. From 2013 to 2017, he was an Assistant Professor with Rochester Institute of Technology and has joined the Department of Computer Science and Engineering, University of South Florida, in 2017, where he is currently an Associate Professor. He has been the TPC Member for a number of conferences, including HOST (publications chair), CCS (publications chair), DAC, DATE, RFIDSec, LightSec, WAIFI, FDTC, and DFT. He is serving as an Associate Editor for the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, the *Transactions on Embedded Computing Systems* (ACM), and the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS. He has been a Guest Editor of the IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, the IEEE/ACM TRANSACTIONS ON COMPUTATIONAL BIOLOGY AND BIOINFORMATICS, and the IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING for special issues on security.

Authorized licensed use limited to: University of South Florida. Downloaded on March 02,2022 at 23:49:46 UTC from IEEE Xplore. Restrictions apply.