An Efficient Hardware Design for Fast Implementation of HQC

Chen Li[1], Suwen Song[1], Jing Tian[1], Zhongfeng Wang[1], and Çetin Kaya Koç [2]
[1] School of Electronic Science and Engineering, Nanjing University, Nanjing, China
Email: chen_li@smail.nju.edu.cn, suwsong@sina.com, tianjing@nju.edu.cn, zfwang@nju.edu.cn
[2] University of California, Santa Barbara, Santa Barbara, USA
Email:cetinkoc@ucsb.edu

2023 IEEE 36th International System-on-Chip Conference (SOCC) | 979-8-3503-0011-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/SOCC58585.2023.10257054

***Abstract*—Hamming Quasi-Cyclic (HQC), as a code-based Key Encapsulation Mechanism (KEM) algorithm, has been selected as one of the three code-based candidates in the fourth round of standardizing post-quantum cryptographic primitives. Efforts are needed for the efficient hardware implementation for HQC. However, in existing hardware designs for HQC, they cost too many clock cycles in the polynomial multiplication module and do not pay enough attention to the decoders used in the decryption. Therefore, this paper presents an improved hardware design for HQC. Through applying a low-latency polynomial multiplication design, every stage of our design saves amounts of clock cycles. Moreover, an efficient Reed-Solomon (RS) decoder based on the enhanced Parallel Inversionless Berlekamp-Massey (ePIBM) algorithm and a Reed-Muller (RM) decoder based on the Fast Hadamard Transform (FHT) algorithm are introduced to reduce the overhead in the decryption. A complete architecture is finally implemented on the Xilinx Artix 7 FPGA (xc7a200t-3). Experimental results show that the proposed design for the HQC-128 requires 25% less area-delay product (ADP) than the latest one in decryption. Furthermore, the proposed design can perform key generation in 0.11ms, encapsulation in 0.22ms, and decapsulation in 0.38ms, which significantly outperforms the state-of-the-art design.**

***Index Terms*—Post-quantum cryptography, Hamming quasi-cyclic, Reed-Solomon decoder, Reed-Muller decoder, hardware implementation.**

I. Introduction

Powerful quantum computers have been proven that they can solve discrete logarithms and factor large numbers trivially, which extremely threaten the security of most public-key cryptography algorithms used for the last few decades, such as the Rivest Shamir Adleman (RSA), Diffie-Hellman, and Elliptic Curve Cryptography (ECC). To deal with this potential threat, the Post-Quantum Cryptography (PQC) is introduced. Since 2016, the National Institute of Standards and Technology (NIST) [1] has been conducting a standardization process to call for the public evaluation of PQC algorithms. This process has reached the fourth round and there survive four main families of PQC algorithms now: lattice-based, isogeny-based, hash-based, and code-based cryptography. To be more specific, CRYSTALS-Kyber [2] has been selected to be standardized in the KEM category as a lattice-based algorithm and there are still three code-based candidates to be considered: Bit Flipping Key Encapsulation (BIKE) [3], Classic McEliece [4], and HQC.

In this round, the algorithm performance and hardware implementation can be regarded as the most important factors for these candidates to be standardized. As one of the three KEM candidates, HQC mainly relies on the hardness of decoding random quasi-cyclic codes in Hamming metric. It not only has the optimal speed in key generation and decryption among them, but has also been proven to be Chosen Ciphertext Attack (IND-CCA) secure [5]. That means HQC can be a reliable choice definitely. However, the efficiency of HQC is still far lower than the traditional solutions, like the RSA and ECC. Besides, only a few papers explored its hardware implementations [6]-[8]. The first hand-optimized design of HQC is reported in [6]. It is also the state-of-the-art hardware work until now. The High-Level Synthesis (HLS) implementation of the HQC-128 version has been released in [7]. It allows faster experimentation of software code to a hardware platform but the efficiency is limited. The work in [8] shows an efficient architecture to realize the polynomial multiplication in HQC, which can complete polynomial multiplication operation in fewer resources with a small number of clock cycles by using the sparsity of the adopted polynomial multiplications. It is worth mentioning that the latest hardware design in HQC [6] costs too many clock cycles in the polynomial multiplication and pays not enough attention to the decoders used in decryption. To alleviate these problems, we propose an efficient architecture for HQC. It should be noted that this paper mainly implements the security of the HQC-

128 version as an example and the techniques can be directly used for other parameters. The main contributions of this work are summarized as follows:

• A low-latency polynomial multiplication architecture is presented to improve the performance, which can reduce the clock cycles by 42% compared with the polynomial multiplication architecture in [6].

• To further reduce the overhead in decryption, an efficient Reed-Solomon (RS) decoder based on the enhanced Parallel Inversionless Berlekamp-Massey (ePIBM) algorithm and a Reed-Muller (RM) decoder based on the Fast Hadamard Transform (FHT) algorithm are dedicatedly designed. The results implemented on Xilinx Artix 7 FPGA (xc7a200t-3) show that the proposed design can reduce the area-delay product (ADP) by about 25%.

The remainder of the paper is structured as follows. Section II describes the background of HQC. Section III gives the proposed hardware architecture. Section IV shows the implementation results of the proposed design and compares with related works. Section V concludes the paper.

II. Preliminaries

A. Notations

In this paper, F_2 denotes the binary field and $\mathcal{R} = F_2[X]/(X^n - 1)$ represents the quotient rings on F_2. The elements of $\mathcal{R}$ can be written as vector $\mathcal{V} = (v_0, v_1, ...v_{n-1})$ or polynomial $\mathcal{V} = \sum_{i=0}^{n-1} v_i X^i$. We define the polynomial multiplication for HQC as $W = SD\, mod\, (X^n - 1)$, where S and D are polynomials on F_2 and W is the resultant polynomial of multiplication. Note that S is a polynomial with only ω nonzero coefficients. $\mathcal{G}(\cdot), \mathcal{H}(\cdot)$ and $\mathcal{K}(\cdot)$ are hash functions used in HQC KEM. Since this paper is mainly for HQC-128 version, the length of the vector n is 17669. The weights of the sparse vector ω, ω_r are 66 and 75, respectively. As for RS code and RM code, $[N, K, d]$ denote the length, the dimension and the minimum distance of the code. They are [46,16,31] and [384,8,192], respectively. Specially, the codes used in HQC are shortened RS code and duplicated RM code. The multiplicity of duplicated RM code is 3.

B. Background of HQC

HQC Public Key Encryption (HQC-PKE) consists of three main primitives: Key Generation, Encryption and Decryption.

Key Generation: Firstly, vector h is sampled uniformly random and two vectors x, y are sampled with a specified weight ω. Then the product of vector h and the sparse vector y needs to add to the sparse vector x. The resultant vector $s = x + hy$ and the vector h compose the public key. The private key is the sparse vector (x, y). The above polynomial multiplication and addition take place in F_2.

Encryption: Just like key generation, three vectors r_1, r_2 and e are sampled with a specified weight ω_r. At the same time, the message m is encoded first by RS encoder and then RM encoder to get t. Finally, the ciphertext pair (u, v) can be calculated by $u = r_1 + hr_2$ and $v = t + sr_2 + e$.

Decryption: $v - uy'$ will be sent to the RM decoder first and then the RS decoder. The message m will be retrieved correctly if the number of the error in received ciphertext is less than the error-correction capability of the code. Indeed, the probability of decoding failures is very low. The entire process of HQC-PKE is shown in Algorithm 1.

HQC KEM is also composed of three primitives: Key Generation, Encapsulation and Decapsulation. The process of key generation in HQC-KEM is the same as that in HQC-PKE. Then the message m will be sampled to generate the shared secret using $pk = (h, s)$ while the seed $\theta = \mathcal{G}(m)$ ensures the randomness of the encapsulation. At the same time, the shared secrets $K = \mathcal{K}(m, c)$ and $d = \mathcal{H}(m)$ are used to calculate the ciphertext $[c|d]$. Concerning decapsulation, the c in ciphertext is used in the decryption to retrieve the message m' and it is necessary to check whether the received ciphertext is correct. Therefore, the encapsulation needs to be done again using the result m' to get the new ciphertext $[c'|d']$. Finally, it is easy to know if there is an error by verifying whether the received ciphertext and the new ciphertext are identical. Algorithm 2 shows the process of HQC-KEM.

Algorithm 1: HQC-PKE

Input : message $\mathbf{m}$, parameter $= (\omega, \omega_r, \theta)$
Output: $\mathbf{m}'$

Key Generation :
$\mathbf{h} \leftarrow \mathbf{R}, (\mathbf{x}, \mathbf{y}) \xleftarrow{\omega} \mathbf{R^2}$.
$\mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y}$.
return : $pk = (\mathbf{h}, \mathbf{s})$ $sk = (\mathbf{x}, \mathbf{y})$.
Encryption :
$\mathbf{r_1}, \mathbf{r_2}, \mathbf{e} \xleftarrow{\omega_r, \theta} \mathbf{R^3}$.
$\mathbf{u} = \mathbf{r_1} + \mathbf{h} \cdot \mathbf{r_2}$.
$\mathbf{t} = encode(\mathbf{m})$.
$\mathbf{v} = \mathbf{t} + \mathbf{s} \cdot \mathbf{r_2} + \mathbf{e}$.
return : $\mathbf{c} = (\mathbf{u}, \mathbf{v})$.
Decryption :
$\mathbf{m}' = decode(\mathbf{v} - \mathbf{u} \cdot \mathbf{y})$.
return : $\mathbf{m}'$.

Algorithm 2: HQC-KEM

Input : message $\mathbf{m}$, parameter $= (\omega, \omega_{\mathrm{r}})$
Output: K'

Key Generation :
$\mathbf{h} \leftarrow \mathbf{R}, (\mathbf{x}, \mathbf{y}) \xleftarrow{\omega} \mathbf{R^2}$.
$\mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y}$.
return : $pk = (\mathbf{h}, \mathbf{s})$ $sk = (\mathbf{x}, \mathbf{y})$.
Encapsulation :
$\theta = \mathcal{G}(\mathbf{m})$.
$\mathbf{c} = (\mathbf{u}, \mathbf{v}) = Encryption(pk, \mathbf{m}, \theta)$.
$K = \mathcal{K}(\mathbf{m}, \mathbf{c})$.
$\mathbf{d} = \mathcal{H}(\mathbf{m})$.
return : $(K, (\mathbf{c}, \mathbf{d}))$.
Decapsulation :
$\mathbf{m}' = Decryption(sk, \mathbf{c})$.
$\theta' = \mathcal{G}(\mathbf{m}')$.
$\mathbf{c}' = (\mathbf{u}', \mathbf{v}') = Encryption(pk, \mathbf{m}', \theta')$.
$\mathbf{d}' = \mathcal{H}(\mathbf{m}')$.
$K' = \mathcal{K}(\mathbf{m}', \mathbf{c})$.
if $\mathbf{c} \neq \mathbf{c}'$ *or* $\mathbf{d} \neq \mathbf{d}'$ **then**
  return : $(K', 0)$.
else
  return : $(K', 1)$.
endif

III. Proposed Hardware Architecture for HQC

In this section, we will mainly introduce the optimized modules in detail, including polynomial multiplication module, RM decoder module and RS decoder module. The polynomial multiplication is applied in every primitive of HQC. Therefore, we propose an efficient architecture with fewer clock cycles in this paper. Moreover, the work in [6] does not pay enough attention to the decoders used in decryption. To further reduce the overhead in this part, we dedicatedly implement an efficient RS decoder based on enhanced Parallel Inversionless Berlekamp-Massey (ePIBM) algorithm and a RM decoder based on Fast Hadamard Transform (FHT) algorithm. Meanwhile, SHAKE256 in HQC is used to realize several functions e.g., as a pseudorandom number generator (PRNG) to generate the specified weight vectors in Key Generation and Encryption, and as a PRNG for hashing in Encapsulation and Decapsulation.

A. Polynomial Multiplication

The whole polynomial multiplication module can be divided into two parts: accumulated addition and reduction. As is known, the length of the resultant polynomial after multiplying two n-bits polynomials can expand to $2n$ bits maximally. However, considering this polynomial multiplication takes place in F_2, the resultant polynomial needs to be reduced to back to n bit by reduction operation.

The polynomial multiplication in HQC is the sparse polynomial multiplier [9], [10], which means one of the operand vectors has only very few nonzero elements. If implementing the polynomial multiplication operation by the traditional methods, the complexity will be $\mathcal{O}(n^2)$. However, the nonzero coefficients of the sparse polynomial are all "1", which is different from the ordinary polynomial multiplication. So this operation can be achieved easily by shifting another polynomial based on the location of "1" in sparse polynomial and accumulating all of the shifted polynomials. In this case, the complexity of the algorithm can be reduced to $\mathcal{O}(n\omega)$.

The operands are 128 bits, and this paper will use the same parameters. Thus, the RAM_D, which is used to store the polynomial D. The depth and width of RAM_D are 139 and 128, respectively. Meanwhile, as the input of the multiplication, the indices loc of the nonzero elements in S will be stored in RAM_L. Its depth is ω and width is 15. As is shown in Fig. 1, when the process of the multiplication starts, the signal $raddr\_ini$ will be read from RAM_L, which is the highest 8 bits of the loc. It will be the initial reading address of the RAM_A, which are composed of four RAMs mainly used for storing the accumulated values generated in the process of multiplication. At the same time, the data $rdata\_d$ will be read from RAM_D. Then it needs to be shifted first to add the data $rdata\_a$ read from the RAM_A. The summation of them can be derived to the bit-wise XOR operations. The details of this procedure can be seen in Fig. 2.

Since the operands in every cycle are 128 bits, it should be considered that there are likely partial data of the $rdata\_d$ used in addition. Therefore, the $rdata\_d$ will undergo the left shifting operation controlled by the lowest 5 bits of the loc. Then the remains will be the $carry$ to be added in next cycle. Eventually, the summation will be divided into 4 parts and written to the different RAM_A simultaneously. Every time reading a loc from RAM_L, it needs 139 cycles for the process of addition until reading the next loc. It should be noticed that the reading and writing addresses of RAM_A and RAM_D need to add one per cycle in this process, which means each RAM_A needs at least 277 addresses to store the accumulated values and one more address to store the $carry$. After getting the final summations, the $2n$-bit polynomial needs to be reduced by slicing it into two parts and then performing an XOR operation.

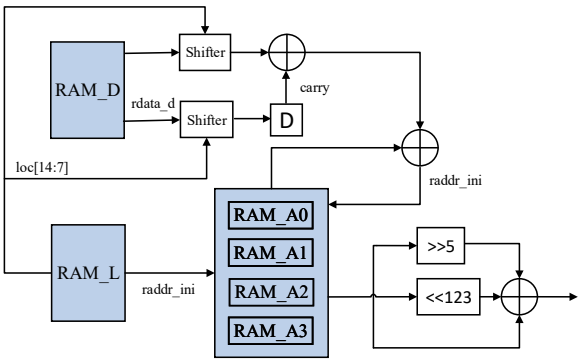

Fig. 1. The architecture of the polynomial multiplication.

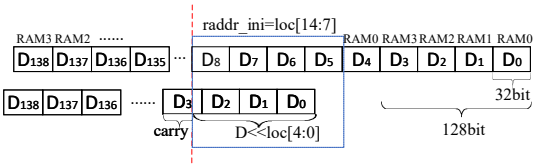

Fig. 2. The process of data shifting.

B. RM Decoder

The ciphertext needs to be decoded first by RM decoder and then RS decoder. Since the duplicated RM code used in HQC is a first-order RM code essentially, it can be decoded most efficiently using the FHT algorithm [11]. Thus, our proposed architecture will be based on FHT algorithm.

C. RS Decoder

In general, the process of decoding can be divided into three steps: Syndrome Computation(SC), Key Equation Solver(KES) and Chien Search and Error Evaluation(CSEE). Actually, the RS decoder occupies most of the area among the three modules in Decryption. Furthermore, the logic consumption of RS decoder is mainly in KES. Hence, the proposed architecture chooses an area-saving decoding algorithm: ePIBM algorithm. Since the ePIBM algorithm can significantly reduce the number of Processor Elements (PE) while ensure that other modules remain unchanged when compared to the other RS decoding algorithms. It can be regarded as the best RS decoding algorithm in time domain until now. The introduction of the ePIBM decoding algorithm can be found in [12], [13].

The decoded codewords from the RM decoder are first sent to calculate the syndrome S_j $(0 \leq j < 2t)$ where $2t = N - K$. The evaluation value of the received polynomial can be written as $y = y_{n-1}x^{n-1} + y_{n-2}x^{n-2} + \cdots + y_0$, where $x = \alpha^{j+1}$. If applying the Horner's rule, it can be rewritten as $y = (\cdots((y_{n-1}x + y_{n-2})x + y_{n-3})x + \cdots)x + y_0$. In fact, this formula can be achieved in hardware

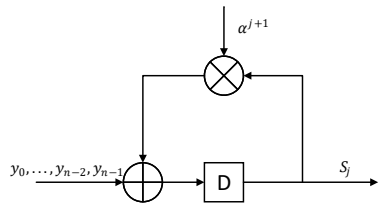

Fig. 3. Syndrome computation architecture.

serially or parallelly. In order to save the area, the proposed architecture chooses the former and the S_j is available in the register after N clock cycles. The simple feedback loop to compute the syndrome serially is shown in Fig. 3.

Let $S(x) = \sum_{j=0}^{2t-1} S_j x^j$ be the syndrome polynomial. In the original RS decoding algorithm, the aim of KES mainly solves the equation $\Lambda(x)S(x) = \Omega(x) \ mod \ x^{2t}$ to obtain the error locator polynomial $\Lambda(x) = \prod_{l=1}^{v}(1 - X_l x) = \Lambda_0 + \Lambda_1 x + \cdots + \Lambda_v x^v$ and the error evaluator polynomial $\Omega(x)$. Generally, the computation of this equation requires the inversion operation, which has very long data path and costs too many logic resources. Although the later algorithm eliminates this operation, it still needs $3t + 1$ PEs to solve that. But in the ePIBM algorithm, A scratch polynomial $B(x)$ is used to update the $\Lambda(x)$. And $\gamma^{(r)}$ is the r-th constant coefficient of the $\Lambda^r(x)$, which is used to calculate the discrepancy coefficient for iteration r. Moreover, $z = \alpha^{-(r-1)}$ is computed iteratively in synchronization with the Chien search on $B(x)$ and $\Lambda(x)$. As long as $\gamma^{(r)}$ is unchanged, the higher coefficients of $\Lambda(x)$ and $B(x)$ do not need to be calculated or stored, which means it only needs $2t + 1$ PE shown in Fig. 4(b) to finish this step. Moreover, the critical path of the PE has only one variable multiplier and one adder. After $2t$ cycles, the $\Lambda^{2t}(x)$ derived at the end is the error locator polynomial, whose inverse roots are the error location. The whole architecture for KES is shown in Fig. 4(a).

Eventually, the exhaustive search is used to find the roots of error polynomials over the entire finite field, and the Chien search can implement it efficiently. It will search for $\Lambda(x)$ in the order of α^{-i} for $i = 0, 1, \cdots, n-1$. If $\Lambda(\alpha^{-i}) = 0$, it means the i-th symbol of the received codewords is erroneous. Besides, by using the Horiguchi-Koetter formula, the error magnitudes e_{i_l} can be calculated as $e_{i_l} = (\gamma^{(2t)}\Lambda_0^{(2t)} z^{-i})/(B^{(2t)}\Lambda_{odd}(X_l^{-1}))$. After N cycles, the whole original message will be obtained just add e_{i_l} to the corresponding position of the received polynomial according to the error location found by Chien search. The architecture of the CSEE is shown in Fig. 5. Finally, the completion of RS decoding indicates the end of decryption.

TABLE I
IMPLEMENTATION RESULTS AND COMPARISON

Design	LUT	DSP	FF[1]	BR[2]	F[3]	Cycles	Times[4]	ADP
Polynomial Multiplication	2239	0	612	4	198	10921	55.16	123k
[6]	1834	0	573	4	228	18765	82.30	150k
Key Generation	2813	0	1318	10	172	16188	94.12	264k
[6]	2350	0	1106	9.5	164	23480	140	336k
Encryption	2716	0	1894	13	192	24016	125	339k
[6]	2245	0	1667	13	245	44982	183	411k
Decryption	6641	0	5466	13	187	14769	78.97	524k
[6]	5747	0	4801	12.5	204	24889	120	701k

[1] : FF=flip-flop.
[2] : BR=BRAM.
[3] : Unit for F (frequency) : Mhz.
[4] : Unit for Times : μs.

TABLE II
COMPARISON OF THE TIME AND LOGIC RESOURCES FOR OUR HQC ARCHITECTURE WITH THE RELATED WORKS

Scheme	LUT	DSP	FF	BR	F	Encap		Decap		KeyGen	
						Mcyc.	ms	Mcyc.	ms	Mcyc.	ms
HQC-128 (this work)	19794	0	11079	68	178	0.02	0.11	0.04	0.22	0.07	0.38
Classic McEliece [14]	40018	4	61881	178	113	0.97	8.60	0.03	0.30	0.10	0.90
BIKE [15]	52967	13	7035	49	96	0.26	2.60	0.01	0.10	0.19	1.90
HQC-128-RTL [6]	16956	0	9837	66	204	0.03	0.12	0.06	0.30	0.08	0.43
HQC-128-HLS [7]	20169	0	16374	25	148	0.04	0.27	0.09	0.59	0.19	1.27

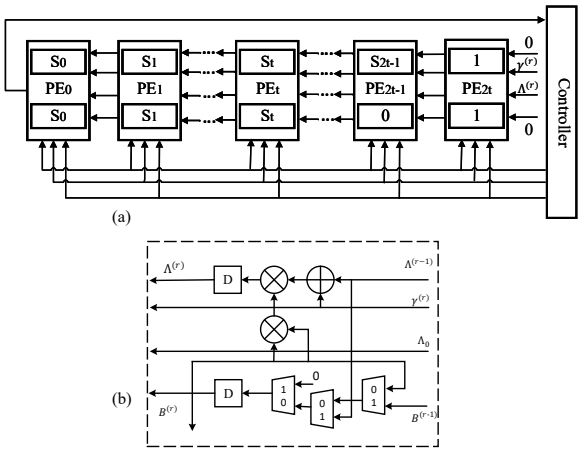

Fig. 4. ePIBM architecture for KES. (a) overall architecture; (b) architecture for PE

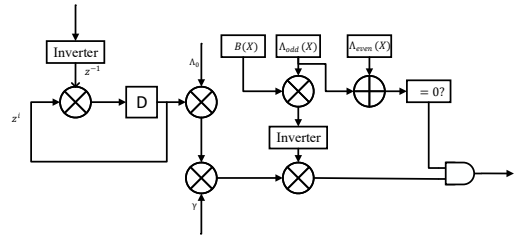

Fig. 5. CSEE architecture.

IV. IMPLEMENTATION RESULTS AND COMPARISON

In this seciton, the designed hardware architecture is coded with Verilog HDL and implemented on Artix 7 board with xc7a200t-3 FPGA chip. We compared our proposed architecture with [6] firstly. The results of the Polynomial multiplication, Key generation, Encryption and Decryption modules are exhibited in Table I. Since we refer to the SHAKE256 module in [6], it will not be repeated here.

Note that the SHAKE256 module is shared among all primitives, so it is not included in Table I. We consider the ADP as the criterion for comparison. Apparently, our proposed architecture has the lower area-time complexities than that in [6] based on the same memory block size and on the same FPGA device for the HQC-128 version. It can be seen in Table I that the proposed architecture has at least 18% less ADP than that in [6]. The degree of improvement increases to 21% in Key Generation and 17% in Encryption. As for Decryption, not only we save more clock cycles by the proposed Polynomial multiplication, but also choose the most efficient decoding algorithms for two decoders. The

ADP of our architecture is 25% less than the design in [6]. For a more comprehensive explanation, it can be pointed out that our proposed hardware design involves less delay time than [6] while only increase few area usage when considering the number of LUTs, FFs and BRAMs.

Moreover, we will give the latest results of three PKE and KEM candidates in NIST's standardization process: BIKE, Classic McEliece, and HQC. Their timing and area are listed in Table II. Since we use the same SHAKE256 module as [6], we also have the problems that the SHAKE256 module limits the maximum clock frequency of all primitives. Therefore, the dual clock design is applied in our proposed architecture, which can ensure that the SHAKE256 module is able to work at a lower frequency but the other modules at a higher frequency.

As Table II shows, our proposed architecture can perform key generation in $0.11ms$, encapsulation in $0.22ms$, and decapsulation in $0.38ms$, with slightly more logic resources like LUTs and FFs. The result of Encapsulation, Decapsulation, and Key Generation are all faster than the state-of-the-art works for HQC-128. In addition, our hardware design achieves the fastest Key Generation and Decapsulation among all candidates and the speed of Encapsulation is faster than all other candidates except BIKE design. In summary, our proposed architecture can achieve the high-speed process of encryption with as little area as possible, which can meet the requirements in the fourth round of standardizing cryptographic primitives exactly.

V. Conclusion

This paper presents an improved hardware design for HQC. The proposed design consumes small logic resources while achieving high speed with the aid of a low-latency polynomial multiplication module and two decoders based on efficient decoding algorithms. The implementation results demonstrate that our architecture outperforms prior works. In particular, it leads to a reduction of 25% on ADP in decryption compared to the latest one for the HQC-128. Additionally, the proposed optimization methods are also suitable for the HQC-192 and HQC-256 versions.

VI. Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant 62174097, in part by the Key Research of Jiangsu Province of China under Grant BE2022098, and in part by the National Key R&D Program of China under Grant 2022YFB4400604. (Corresponding authors: Suwen Song; Zhongfeng Wang.)

References

[1] G. Alagic, G. Alagic, J. Alperin-Sheriff, D. Apon, D. Cooper, Q. Dang, Y.-K. Liu, C. Miller, D. Moody, R. Peralta *et al.*, "Status report on the first round of the NIST post-quantum cryptography standardization process," 2019.

[2] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehle, "CRYSTALS - Kyber: A CCA-secure module-lattice-based KEM," in *2018 IEEE European Symposium on Security and Privacy*, 2018, pp. 353–367.

[3] N. Aragon, P. L. Barreto, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, S. Gueron, T. Güneysu, C. A. Melchor, R. Misoczki, E. Persichetti, N. Sendrier, J.-P. Tillich, and G. Zémor, "BIKE: Bit flipping key encapsulation," 2017.

[4] H. Singh, "Code based cryptography: Classic McEliece," *CoRR*, vol. abs/1907.12754, 2019. [Online]. Available: http://arxiv.org/abs/1907.12754

[5] L.-P. Wang, "Loong: a new IND-CCA-secure code-based KEM," in *2019 IEEE International Symposium on Information Theory (ISIT)*, 2019, pp. 2584–2588.

[6] S. Deshpande, M. Nawan, K. Nawaz, J. Szefer, and C. Xu, "Towards a fast and efficient hardware implementation of HQC," *IACR Cryptol. ePrint Arch.*, vol. 2022, p. 1183, 2022.

[7] C. Aguilar-Melchor, J.-C. Deneuville, A. Dion, J. Howe, R. Malmain, V. Migliore, M. Nawan, and K. Nawaz, "Towards automating cryptographic hardware implementations: a case study of HQC," in *Code-Based Cryptography: 10th International Workshop, CBCrypto 2022, Trondheim, Norway, May 2930, 2022, Revised Selected Papers*, p. 6276.

[8] Y. Tu, P. He, C. K. Koc, and J. Xie, "Leap: Lightweight and efficient accelerator for sparse polynomial multiplication of HQC," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pp. 1–5, 2023.

[9] S. Deshpande, S. M. d. Pozo, V. Mateu, M. Manzano, N. Aaraj, and J. Szefer, "Modular inverse for integers using fast constant time GCD algorithm and its applications," in *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*, 2021, pp. 122–129.

[10] E. Montagne and R. Surs, "Systolic sparse matrix vector multiply in the age of tpus and accelerators," in *2019 Spring Simulation Conference (SpringSim)*, 2019, pp. 1–10.

[11] M. Hashemipour-Nazari, K. Goossens, and A. Balatsoukas-Stimming, "Hardware implementation of iterative projection-aggregation decoding of Reed-Muller codes," in *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021, pp. 8293–8297.

[12] X. Zhang, "VLSI architectures for Reed Solomon codes: Classic, nested, coupled, and beyond," *IEEE Open Journal of Circuits and Systems*, vol. 1, pp. 157–169, 2020.

[13] Y. Wu, "New scalable decoder architectures for Reed Solomon codes," *IEEE Transactions on Communications*, vol. 63, no. 8, pp. 2741–2761, 2015.

[14] P.-J. Chen, T. Chou, S. Deshpande, N. Lahr, R. Niederhagen, J. Szefer, and W. Wang, "Complete and improved FPGA implementation of Classic McEliece," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, no. 3, pp. 71–113, 2022.

[15] J. Richter-Brockmann, J. Mono, and T. Gneysu, "Folding BIKE: Scalable hardware implementation for recongurable devices," *IEEE Transactions on Computers*, vol. PP, pp. 1–1, 05 2021.