CEDAR: A Compact and Efficient Decoder Architecture for RS-RM Code in HQC

Yazheng Tu*, Tianyou Bao*, and Jiafeng Xie
Department of Electrical and Computer Engineering, Villanova University, Villanova, PA, USA
Emails: {ytu1, tbao, jiafeng.xie}@villanova.edu; *: Equal Contribution

***Abstract*—The rapid development of quantum computing has driven a new wave of cryptographic innovation: Post-Quantum Cryptography (PQC), a class of algorithms that resist quantum attacks. In particular, the National Institute of Standards and Technology (NIST) has initiated the PQC standardization process, selecting five algorithms. Notably, HQC (the newest selection) is a code-based PQC scheme that has not been widely studied in the literature, especially with respect to its hardware acceleration. This paper follows the current trend to design CEDAR, a Compact and Efficient Decoder Architecture for Reed-Solomon Reed-Muller (RS-RM) code in HQC. We have proposed three layers of contributions in total: (i) an optimized RM decoder is designed; (ii) an efficient low-complexity RS decoder is also presented; (iii) a complete HQC decoder is implemented, along with a comprehensive evaluation (it is shown that CEDAR outperforms the existing approach). We hope this outcome will facilitate a more efficient hardware acceleration of HQC and impact the ongoing NIST PQC standardization process.**

***Index Terms*—Compact design, hardware accelerator, PQC, HQC decoder, Reed-Solomon, Reed-Muller.**

I. Introduction

The development of quantum computing brings threats to conventional cryptographic systems, particularly RSA and Elliptic Curve Cryptography, which are vulnerable to quantum attacks using Shor's algorithms [1]. In response, the National Institute of Standards and Technology (NIST) initiated the Post-Quantum Cryptography (PQC) standardization process in 2016 [2]. Several algorithms have been selected, including the most recent code-based Hamming Quasi-Cyclic (HQC) [3].

Brief background. HQC is a code-based Key Encapsulation Mechanism (KEM) method, whose hardness relies on the syndrome decoding problem for quasi-cyclic codes [4]. HQC performs polynomial reduction of computationally hard problems to ensure its security guarantees without requiring indistinguishability assumptions [5]. HQC contains two linear codes: a random quasi-cyclic code to ensure security, and an efficient and error-correction-capable fixed public concatenated code, which consists of Reed-Solomon (RS, external code) combined with Reed-Muller (RM, internal code). This dual-code structure reduces the ciphertext size while meeting the security requirements for NIST security levels 1, 3, and 5.

Existing works. Recent research in PQC has gradually switched to the hardware implementation side. For HQC, however, there are still not many related hardware designs: a sparse polynomial multiplier design is presented in [6]; a constant-timing fixed-weight sampler is reported in [7]; a high-level Synthesis designed HQC is released in [8]; hardware-implemented HQC is presented in [9]; a low-latency polynomial multiplication with RS and RM decoders is given in [10].

Challenges. Compared to operations like sparse polynomial multiplication and encoding, decoding in HQC involves more complex arithmetic procedures. Despite the existing hardware design works achieving various advances, there still lacks sufficient attention to a resource-optimized decoder for HQC: existing designs prioritize the optimization of sparse polynomial multiplication while the decoder part remains under-explored. Finally, recent reports like [9] has indicated HQC decoder is a critical component and its efficient implementation determines the overall hardware acceleration efficiency for HQC.

Contributions. Therefore, this paper presents CEDAR, a Compact and Efficient Decoder Architecture for RS-RM concatenated codes in HQC. The primary contributions include:

- A compact and pipelined RM decoder is proposed, with optimized Fast Hadamard Transform and a compare unit.
- An efficient RS decoder with low resource consumption is presented, which includes a novel low-complexity error location/evaluation polynomial calculator deploying Enhanced Parallel Inversionless Berlekamp-Massey Algorithm (ePIMBA) and a novel low-complexity iterative Chien Search and Error Evaluation (CSEE) component.
- A comprehensive evaluation is given to demonstrate superior performance compared to an existing implementation across all NIST security levels.

The remainder of this paper is organized as follows: Section II presents the preliminaries; Section III details the proposed decoder architecture; Section IV provides implementation results and comparision; and Section V draws a conclusion.

II. Preliminaries

A. HQC Arithmetic

Let $\mathbb{F}_2$ denote the binary finite field, and $R = \mathbb{F}_2[x]/\langle x^p - 1\rangle$ represent a polynomial ring, where $p > 3$ is a prime satisfying $\text{ord}_2(p) = p - 1$. Each polynomial $a \in R$ can be represented as $a = a_0 + a_1 x + \cdots + a_{p-1}x^{p-1}$, and is equivalently as a p-dimensional binary vector $\mathbf{a} = [a_0, a_1, \ldots, a_{p-1}] \in \mathbb{F}_2^p$.

This work was accepted in the 2026 IEEE International Symposium on Circuits and Systems (ISCAS'26).

The cyclic structure of R constains polynomial addition and multiplication. The polynomial addition is corresponded with coefficient-wise XOR: $a + b = [a_0 \oplus b_0, \ldots, a_{p-1} \oplus b_{p-1}]$, and the multiplication $c = a \cdot b$ produce coefficients $c_i = \bigoplus_{j+k \equiv i \bmod p}(a_j \wedge b_k)$ for $i, j, k \in \{0, 1, \ldots, p-1\}$.

HQC relies on random quasi-cyclic $[2p, p, d]$ code with public parity-check matrix $H = [I_p \mid \text{rot}(h)]$, where I_p is the $p \times p$ identity matrix and $\text{rot}(h)$ represents the circulant matrix generated from polynomial $h \in R$. The public error-correction code combines a shortened RS $[n_e, k_e, d_e]$ external code with a duplicated RM $[n_i, k_i, d_i]$ internal code, and produces a concatenated code with parameters $[n_e n_i, k_e k_i, d_e d_i]$, denoted by generator matrix G.

B. RS Code

A $[n, k, d]$ binary linear code $\mathcal{C}$ forms a subspace of $\mathbb{F}_2^n$ with dimension k and minimum distance d. Generator matrix $G \in \mathbb{F}_2^{k \times n}$ generates all 2^k codewords: $\mathcal{C} = \{mG \mid m \in \mathbb{F}_2^k\}$. Parity-check matrix $H \in \mathbb{F}_2^{(n-k) \times n}$ satisfies $H\mathbf{c}^T = \mathbf{0}$ for all $\mathbf{c} \in \mathcal{C}$, with $\mathbf{s} = H\mathbf{v}^T \in \mathbb{F}_2^{n-k}$ for arbitrary $\mathbf{v} \in \mathbb{F}_2^n$.

For each NIST security level, HQC have different parameters for each shortened RS code. For security levels 1, 3, and 5, the respective parameters are RS-S1$[n_e = 46, k_e = 16, d_e = 31]$, RS-S2$[n_e = 56, k_e = 24, d_e = 33]$, and RS-S3$[n_e = 90, k_e = 32, d_e = 59]$. Each code is capable of correcting up to $\delta = \lfloor (d_e - 1)/2 \rfloor$ symbol errors. Each code operates over $\mathbb{F}_{2^8}$ using irreducible polynomial $y^8 + y^4 + y^3 + y^2 + 1 \in \mathbb{F}_2[y]$. The message polynomial $m(x) \in \mathbb{F}_{2^8}[x]$ of degree $k_e - 1$ generates an n_e-symbol codeword through the encoding process.

The RS decoding process contains three stages. In the first stage, using the Horner's method to compute the syndromes $S_j = r(\alpha^j)$ for $j \in \{0, \ldots, 2t-1\}$ from the received polynomial $r(x)$. The second stage is to compute the error locator polynomial $\Lambda(x)$ and error evaluator polynomial $\Omega(x)$ based on the syndromes and this can be done by using the ePIMBA algorithm (Algorithm 1). The last stage, the Chien search identifies error locations by finding roots of $\Lambda(x)$, and Forney's formula calculates error magnitudes.

C. RM Code and Fast Walsh-Hadamard Transform

RM code uses parameters n_i for the code length, k_i for the dimension, and d_i for the minimum distance. HQC uses first-order binary RM codes extended with repetition. For each NIST security level, HQC uses first-order binary RM code $[n_i = 2^l, k_i = l + 1, d_i = 2^{l-1}]$. For security levels 1 ($l = 3$), 3 ($l = 5$), and 5 ($l = 7$), the RM code $[n_i, k_i, d_i]$ are $[384, 8, 192]$, $[640, 8, 320]$, and $[1024, 8, 512]$, respectively.

RM decoding employs Maximum Likelihood decoding via the Fast Walsh-Hadamard Transform, achieving $O(n_i \log n_i)$ complexity compared to $O(n_i^2)$ for naive approaches. Fast Walsh-Hadamard Transform is a computational algorithm that efficiently computes the Walsh-Hadamard transform of a vector. In HQC decoding, the Walsh-Hadamard Transform recursively decomposes the $2^m \times 2^m$ Hadamard matrix using the Walsh-Hadamard Transform definition:

$$H_m = \begin{pmatrix} H_{m-1} & H_{m-1} \\ H_{m-1} & -H_{m-1} \end{pmatrix} \quad (1)$$

Algorithm 1: Enhanced Parallel Inversionless Berlekamp-Massey Algorithm (ePIMBA) [11]

Input: $\mathbf{S} = [S_0, S_1, S_2, \ldots, S_{2t-1}]$

Initialization: $\hat{\Theta}^{(0)}(x) = [S_0, , S_1, \ldots, S_{2t-2}, 0, 1]$,
$\hat{\Omega}^{(0)}(x) = [S_0, S_1, \cdots S_{2t-2}, S_{2t-1}, 1]$
$\gamma^{(0)} = 1, L_\Lambda^{(0)} = L_B^{(0)} = 0, z = 1$

for $r = 0, \ldots, 2t-1$ **do**
  $\hat{\Omega}^{(r+1)}(x) = \gamma^{(r)} \cdot [L_1 \hat{\Omega}^{(r)}](x) - \hat{\Omega}_0^{(r)} \cdot \hat{\Theta}^{(r)}(x)$
  if $\hat{\Omega}_0^{(r)} \neq 0$ & $L_\Lambda^{(r)} \leq L_B^{(r)}$ **then**
    $\hat{\Theta}^{(r+1)}(x) \leftarrow [L_1 \hat{\Omega}^{(r)}](x)$
    $L_\Lambda^{(r+1)} \leftarrow L_B^{(r)} + 1$, $L_B^{(r+1)} \leftarrow L_\Lambda^{(r)}$
    $\gamma^{(r+1)} \leftarrow \hat{\Omega}_0^{(r)}$, $z \leftarrow \alpha^{-1} z$
  else
    if $L_B^{(r)} = t - 1$ **then**
      $\hat{\Theta}^{(r+1)}(x) \leftarrow [L_1 \hat{\Theta}^{(r)}](x)$
      $L_B^{(r+1)} \leftarrow L_B^{(r)}$
    else
      $\hat{\Theta}^{(r+1)}(x) \leftarrow \hat{\Theta}^{(r)}(x)$
      $z \leftarrow z \cdot \alpha^{-1}$, $L_B^{(r+1)} \leftarrow L_B^{(r)} + 1$
    end if
    $\gamma^{(r+1)} \leftarrow \gamma^{(r)}$, $L_\Lambda^{(r+1)} \leftarrow L_\Lambda^{(r)}$
  end if
  $\hat{\Theta}_{2t-r-2}^{(r+1)} = 0$
end for

Outputs: $\mathbf{\Lambda} = [\hat{\Omega}_0^{(2t)}, \hat{\Omega}_1^{(2t)}, \ldots, \hat{\Omega}_t^{(2t)}]$,
$\mathbf{B} = [\hat{\Theta}_0^{(2t)}, \hat{\Theta}_1^{(2t)}, \ldots, \hat{\Theta}_{t-1}^{(2t)}]$, $\gamma^{(2t)}$, $L_\Lambda^{(2t)}$, z

III. Proposed Decoder: CEDAR

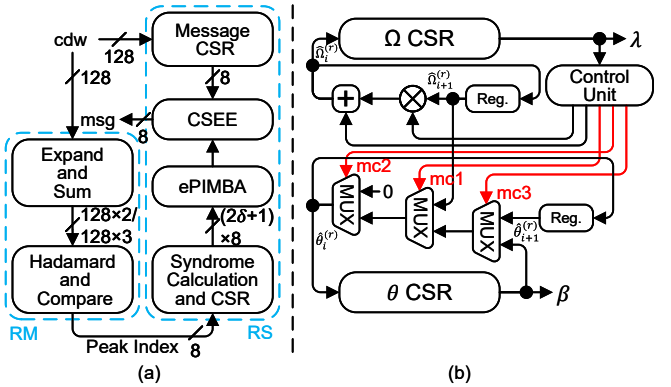

Fig. 1. (a) Overview of CEDAR; (b) Architecture of ePIMBA. CSR: circular shift-register.

A. Top Level Architectural Overview

As shown in Fig. 1(a), the HQC decoder implements a concatenated decoding that recovers the original message from the received codeword corrupted by errors, with the ability to recover at most δ-byte errors. The decoder operates on the 2-tage hierarchical structure where the duplicated RM code serves as the inner code and the shortened RS code serves as the outer code, forming an $[n_1 n_2, k, \delta]$ concatenated code.

The decoder accepts a 128-bit wide input stream representing the received codeword and processes it through

two sequential stages. In the first stage, the RM decoder performs soft-decision decoding on each of the duplicated RM codewords, producing codeword maximum likelihood indices (cdw) that indicate the most likely transmitted codeword for each segment. These decoded symbols are then fed to the RS decoder in the second stage, where algebraic decoding over the finite field $GF(2^8)$ corrects residual errors and recovers the k-byte message by utilizing the ePIMBA algorithm as shown in Algorithm 1. The decoder's parameterization supports all three security levels (128, 192, and 256 bits), with corresponding adjustments to the code parameters: $\delta \in \{15, 16, 29\}$, $n_1 \in \{46, 56, 90\}$, and $K \in \{16, 24, 32\}$. The decoder also takes in $start$ and outputs $done$ and reading address for the interaction with outer components e.g. BRAM, with the decoded message output on msg having width $8{\cdot}k$ bits. The modular architecture facilitates pipeline operation where RM decoding can keep working while the RS decoding of previous blocks proceeds.

B. RM Decoder

The RM decoder implements a Fast Hadamard Transform based decoding algorithm to identify the maximum-likelihood codeword and its corresponding index from the received codeword. The decoder consists of three main components: Expansion-and-Sum, Hadamard and Compare, and Control.

The Expansion-and-Sum component first processes the 128-bit input stream with multiplicity factor MULTIPLICITY $\in \{3, 5\}$ to accumulate replicated codewords. For HQC-1, three replicated codewords are summed element-wise; for HQC-2 and HQC-3, five replicas are summed. The sums are then sent to the Hadamard and Compare component. This component contains 128 3/5-bit full adders executing the additions and a $128 \times 3/128 \times 5$ register storing the accumulated results, depending on the selected security level.

With optimization to a similar design as [9], the Hadamard and Compare component implements a 7-layer butterfly pipeline computing the 128-point Fast Hadamard Transform through iterative pairwise additions and subtractions, along with the comparison of the calculated values to select the peak values and corresponding indices among all transformed coefficients. Each layer contains two adders to calculate the addition and subtraction between two selected coefficients and four simplified FIFOs to store intermediate results for the first and second halves of even and odd indexed coefficients. The FIFOs are optimized specially for this by saving abundant control and calculation read/write addresses and logic complexity. In the last layer, the newly calculated sum/difference will be compared with the existing peak value to determine the value with the maximum absolute value and its corresponding index (two counters are also involved to calculate the final index).

C. RS Decoder

The RS decoder performs error correction over the finite field $GF(2^8)$ using the primitive polynomial, implementing a shortened RS code with parameters $[n_1, k, \delta]$ where the error correction capability δ ranges from 15 to 30 depending on the security level. The decoder receives codeword symbols decoded concatenated RS-codeword from the RM decoder in 8 bits and processes the codeword through four primary components: the Syndrome Calculator and CSR (circular shift-register), the ePIMBA component, the CSEE component, and a Message CSR. The Syndrome Calculator first computes $2t$ syndrome values $S_i = \sum_{j=10}^{n_1-1} cdw_j \cdot \alpha^{ij}$ for $i = 0, 1, ..., 2t-1$, where cdw_j are the received symbols and α is the primitive element. These syndromes are stored and reused by the ePIMBA component to determine the error locator polynomial $\Lambda(x)$ and error evaluator polynomial $\Omega(x)$. The CSEE unit then performs root-finding on $\Lambda(x)$ to identify error positions and computes error values using Forney's algorithm. Finally, k-bytes corrected messages are calculated and output.

The Syndrome Calculator contains a $(2\delta + 1){\cdot}8$-bit shift register, a $GF(2^8)$ multiplier, and $(2t + 1)$ XOR gates. It implements Horner's method using the $GF(2^8)$ multiplier shared across 2δ 8-bit registers in the shift register, where each 8-bit register acts an accumulator with an 8-bit XOR gate. For computing syndrome S_i, the circuit iteratively evaluates $S_i = ((((0 \cdot \alpha^i \oplus cdw_0) \cdot \alpha^i \oplus cdw_1) \cdot \alpha^i \oplus \cdots) \oplus cdw_{n_1}$, where each received symbol cdw_j arrives sequentially from the RM decoder. The shared multiplier cycles through all 2δ syndrome accumulators to calculate the Galois multiplication between the received codeword and the corresponding power of α. The multiplier computes one product between the value from the accumulator and a power of α at each cycle to sequentially update each of the 2δ syndrome registers. After receiving all n_1 codewords and completing all $n_1 \cdot 2\delta$ calculations, the Syndrome CSR stores the computed syndrome values to provide sequential syndrome values to the ePIMBA component. Note that only 2δ registers are updated during the syndrome calculation, with one left to be 1 to provide an initial value required for the ePIMBA calculation (Algorithm 1).

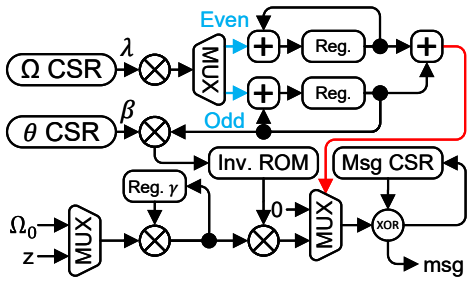

Fig. 2. Architecture of CSEE. CSR: circular shift-register.

The ePIMBA component (Fig. 1(b)) calculates the error locator/evaluation polynomial $\Lambda(x)$ through an iterative ePIMBA algorithm operating over $(2\delta + 1)^2$ iterations with two shift registers: the Ω register and the Θ register, where both are $(2\delta + 1)$ bytes wide. Initially, the Syndrome CSR is repurposed as the Ω CSR register, loaded with syndrome values $\Omega^{(0)} = [S_0, S_1, \ldots, S_{2t-1}, 1]$, while the Θ CSR is initialized to $\Theta^{(0)} = [S_0, S_1, \ldots, 0, 1]$. The ePIMBA component contains two $GF(2^8)$ multipliers executing the core update equations: $\Omega_{i+1}^{(r)} = \Delta \cdot \Theta_i^{(r)} \oplus \omega_i^{(r)} \cdot \gamma$ and $\Theta_{i+1}^{(r)}$ selected from $\{0, \Omega_i^{(r)}, \Theta_{i+1}^{(r)}\}$ based on control signals mc1, mc2, mc3 calculated by the control unit, where Δ represents the dis-

TABLE I
COMPARISON WITH THE EXISTING DESIGN BASED ON THE IMPLEMENTATION ON THE ARTIX-7 PLATFORM

Design	Device	Security	LUT	FF	BRAM	Slice	Cycle	Fmax (MHz)	Delay (ns)	eSlice[1]	eADP	eADPR
[9]	Artix-7	HQC-1	2,817	3,379	3 (2.5)*	952	4,611	205	22.49	2,232	50,209	25.11%
CEDAR	Artix-7	HQC-1	2,127	2,942	3.5	889	4,350	231	18.83	2,093	39,409	NA
[9]	Artix-7	HQC-3	3,257	4,727	3 (2.5)*	1,100	5,485	212	25.87	2,394	64,429	16.71%
CEDAR	Artix-7	HQC-3	2,685	3,615	3.5	1,013	5,284	232	22.78	2,356	53,666	NA
[9]	Artix-7	HQC-5	3,679	5,574	5.5 (2.5)*	1,243	9,199	206	44.66	3,219	143,734	2.78%
CEDAR	Artix-7	HQC-5	2,974	4,281	6	1,123	10,509	227	46.30	3,019	139,742	NA

*: The existing design didn't include the constant ROM used for the power of α, here we list the BRAM usage with and without the constant ROM. Note we use the BRAM number (includes constant ROM) to calculate the final eADP.
[1]: Here we follow the equation in [12] to calculate the equivalent Slice (eSlice), where 1 eSlice = slice + LUT/4 + BRAM×196.

crepancy value and γ is derived from the previous iteration's feedback. The coefficients in Ω and Θ are fed sequentially into the MUX at each cycle to perform the calculation, iterating through all $2\delta+1$ coefficient positions over $2\delta+1$ iterations to complete the calculation. After the calculation, the first $T+1 \in \{16, 16, 30\}$ coefficients in and $\Omega = [\Omega_0, \Omega_1, \cdots, \Omega_T]$ and first T coefficients in $\Theta = [\Theta_0, \Theta_1, \cdots, \Theta_{T-1}, 0]$ will be used as $\Lambda(x)$ and $\beta(x)$ in CSEE.

The CSEE unit (Fig. 2) performs Chien search and Forney error evaluation using the $(T+1)$-byte Ω and Θ CSR outputs, which function as shift registers representing the polynomial coefficients. The Θ CSR output provides the error locator polynomial $\lambda = [\lambda_0, \lambda_1, \ldots, \lambda_t]$, while the Ω register output provides the error evaluator polynomial $\beta = [\beta_0, \beta_1, \ldots, 0]$, with the $(t+1)$-th β coefficient β_{t+1} forced to zero. During Chien search, each coefficient λ_i is delivered out by the CSR serially and multiplied by α^i using dedicated $GF(2^8)$ multipliers, and the products are routed through a demultiplexer to accumulate in either odd-indexed or even-indexed accumulators. The sums of the two parallel accumulators are then sent to an XOR gate to compute $\Lambda(\alpha^j) = \sum_{i=0}^{t} \lambda_i \cdot (\alpha^j)^i$ to determine the roots. Whether $\Lambda(\alpha^j) = 0$ indicates if error at position j and will be used as a selection signal to determine whether the received message should be XORed with the calculated error value. Meanwhile, the β coefficients are directly multiplied by a power of α to evaluate $\Omega(\alpha^j) = \sum_{i=0}^{t} \beta_i \cdot (\alpha^j)^i$. The error value is calculated through Forney's algorithm using an inverse ROM for multiplicative inverse computation and two $GF(2^8)$ multipliers. Either the computed error values or 0's, depending on whether a root exists, are XORed with the corresponding received message from the Msg CSR to produce and output the corrected message.

IV. EVALUATION AND COMPARISON

Experimental Setup. The experimental setup is as follows: (i) the proposed CEDAR is coded in Verilog and implemented on the Artix-7 xc7a200t-3 FPGA through Vivado 2022.2 (after place & route, with functionality verified); (ii) we have obtained its implementation performance under three security level parameter sets of HQC; (iii) the obtained implementation results, including resource usage (LUTs, FFs, and BRAMs), maximum frequency (Fmax), delay, and equivalent Slice (eSlice) are listed in Table I along with those of [9].

Comparison of Implementation Results. Table I presents a comprehensive comparison of our decoder implementation against the state-of-the-art design [9]. Overall, our implementation demonstrates significant improvements in several critical metrics while maintaining competitive performance in others.

In terms of resource usage, our design achieves a notable reduction in eSlice count. For HQC-1, CEDAR achieves slice reduction by 6.6%, while for HQC-3 and HQC-5, the reductions are 7.9% and 9.7%, respectively. This improvement stems from our optimized control logic and efficient resource sharing in the ePIMBA processor. Although we use 0.5 more ROM to store the multiplicative inverses in CSEE, our eSlice still reaches a reduction of 6.2%, 1.6% and 6.6%, respectively.

Meanwhile, the maximum operating frequencies of the proposed design reach over 227MHz across all security levels, compared to 205-212MHz in [9], resulting in at least 9.4% frequency improvement. The enhanced frequency is attributed to our pipelined architecture and the streamlined datapath in the syndrome calculator, ePIMBA, and CSEE component.

The equivalent area-delay product (eADP) and its reduction rate (eADPR) further demonstrate the efficiency of the proposed design. Our HQC-1 design achieves an eADP of 39,409, achieving an eADPR of 25.11% compared [9]. For HQC-3, the eADPR is 16.71%, while HQC-5 shows a modest 2.78% improvement. The diminishing eADPR at higher security levels is primarily due to the increased error correction capability of the RS decoder. This increase causes latency to grow quadratically as $(2\delta+1)^2$, leading to in a drop on the eADPR at higher security parameters. However, the reduced area consumption ensures that the proposed design maintains lower eADP, which demonstrates an overall superior efficiency.

V. CONCLUSION

This paper presents CEDAR for the efficient implementation of the decoder within the newly selected NIST PQC algorithm HQC. We have presented the optimized pipelined RM decoder with Hadamard Transform and Compare architecture for efficient decoding. Then, the details of the efficient low-complexity RS decoder with iterative syndrome calculation, iterative ePIMBA calculation, and a streamlined CSEE unit with serial Chien search are provided. Finally, the implementation results are presented, and a comparison with the state-of-the-art design is provided, demonstrating eADPRs of 25.11%, 16.71%, and 2.78% for HQC-1, HQC-3, and HQC-5, respectively, confirming the superior efficiency of CEDAR.

ACKNOWLEDGMENT

It is supported by Villanova Last Year Ph.D. Fellowship.

REFERENCES

[1] P. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, (Santa Fe, NM, USA), pp. 124–134, IEEE, 1994-11, 1994.

[2] Information Technology Laboratory Computer Security Division, "Post-Quantum Cryptography." https://csrc.nist.gov/projects/post-quantum-cryptography, Jan. 2017.

[3] NIST PQC Standardization Process — HQC Announced as a 4th Round Selection. https://csrc.nist.gov/News/2025/hqc-announced-as-a-4th-round-selection.

[4] HQC. https://pqc-hqc.org/.

[5] P. Gaborit, C. Aguilar-Melchor, N. Aragon, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, E. Persichetti, G. Zémor, J. Bos, A. Dion, J. Lacan, J.-M. Robert, P. Véron, P. L. Barreto, S. Ghosh, S. Gueron, T. Güneysu, R. Misoczki, J. Richter-Brokmann, N. Sendrier, J.-P. Tillich, and V. Vasseur, "Hamming Quasi-Cyclic (HQC)," Aug. 2025.

[6] Y. Tu, P. He, Ç. K. Koç, and J. Xie, "LEAP: Lightweight and Efficient Accelerator for Sparse Polynomial Multiplication of HQC," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, pp. 892–896, June 2023.

[7] P. He, Y. Tu, and J. Xie, "LOCS: LOw-Latency and ConStant-Timing Implementation of Fixed-Weight Sampler for HQC," in *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, May 2023.

[8] 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* (J.-C. Deneuville, ed.), (Cham), pp. 62–76, Springer Nature Switzerland, 2023.

[9] S. Deshpande, C. Xu, M. Nawan, K. Nawaz, and J. Szefer, "Fast and Efficient Hardware Implementation of HQC," in *Selected Areas in Cryptography – SAC 2023: 30th International Conference, Fredericton, Canada, August 14–18, 2023, Revised Selected Papers*, (Berlin, Heidelberg), pp. 297–321, Springer-Verlag, Aug. 2023.

[10] C. Li, S. Song, J. Tian, Z. Wang, and Ç. K. Koç, "An Efficient Hardware Design for Fast Implementation of HQC," in *2023 IEEE 36th International System-on-Chip Conference (SOCC)*, pp. 1–6, Sept. 2023.

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

[12] S. Kundu, A. Ghosh, A. Karmakar, S. Sen, and I. Verbauwhede, "Rudraksh: A compact and lightweight post-quantum key-encapsulation mechanism," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2025, no. 2, pp. 647–680, 2025.