# Hardware Implementation of Q-decoding Algorithm in LEDAkem Encapsulation Mechanism

Bao Bui
*George Mason University*
Fairfax, Virginia 22030
bbui7@gmu.edu

***Abstract*—Error coding codes were originally used to for correct errors in transmitted messages, but they can also be used to encode data to provide confidentiality. The private key allows for efficient decoding to extract and recover the message. Decoding of a given code-based scheme is a hard problem and therefore, an attacker cannot easily decode without a private key or using only a given ciphertext and public key. A major drawback of code-based cryptosystems is the size of the public and private key pairs. This large size also implies many computations, storage and transmission requirements. In this paper, I will discuss the design of a hardware decoder that is aimed to speed up during the decapsulation of the NIST Round 1 Post-Quantum Cryptography (PQC) candidate LEDAKEM, which will be used to reduce the overall execution time. No previous hardware implementations for this algorithm exist. Based on a reference software implementation I have performed profiling to find the major time consuming operations for the decapsulation operations. These hardware components can be extended in future work to speed up the execution of LEDAKEM in a software/hardware co-design implementation. Theoretical results of the decapsulation will be presented and were obtained from the profiling, timing analysis, and Xilinx. Furthermore, this work will be part of a larger project that aims at obtaining the hardware results of many code-based schemes submitted to the NIST PQC standardization process.**

***Index Terms*—tbd**

## I. INTRODUCTION

### A. Background

With the advent of a scalable and reliable quantum computer, all current public-key cryptographic standards, such as Advanced Encryption Standard (AES), RSA, and the Diffie-Helman key-exhcange will be vulnerable to attack using Shor's quantum factoring algorithm. In order to circumvent this threat, NIST has lead a PQC standardization effort - with Round 1 starting on November 31, 2017 [1]. One of the several families of algorithms that are under review include code-based schemes, which have been well studied since the 1970s, both in terms of security and efficiency.

### B. Asymmetric and symmetric cryptorgraphy

Assuming Alice and Bob want to communicate confidentially, they can send data using either an asymmetric or symmetric key based algorithm. The focus for NIST is asymmetric key algorithms, which would be used to establish a shared key used in a symmetric key algorithm. The reason for this is symmetric schemes require keys to be exchanged, either physically or via a public data channel (i.e. the internet, sockets, etc). So typically, an asymmetric key exchange algorithm is used to establish the shared key, which only requires between hundreds to thousands of bits. In comparison, asymmetric schemes can have keys between thousands to millions of bits and require a similar increase in the number of computations required to produce a single encryption/decryption or signature generation/verification. After exchanging keys through a key-exchange mechanism (KEM), a symmetric key algorithm will reduce the overall computation required to send data in the established confidential channel.

In this paper, I will discuss LEDAKEM, which is a KEM based on the Niederreiter cyrptosystem using Quasi-Cyclic Low-Density Parity-Check (QC-LDPC) codes. In particlar, the paper will focus more on Q-decoder function in decapsulation. Fig.1 shows an overview of LEDAKEM, and that majority of the computation during encapsulation and decapsulation are spent in encoding and decoding, respectively. In this diagram, Bob receives a ciphertext, which is a syndrome generated by Alice using Bob's public key. It is decrypted using Bob's private key and used to derive a shared key, $k_s$. Any future communications can be made with $k_s$.

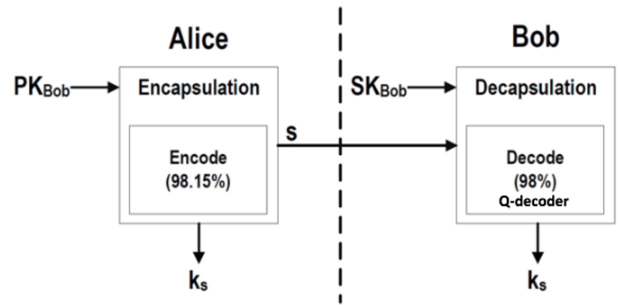


Fig. 1. Key Encapsulation Mechanism

### C. Parameter Sets

The parameter set chosen for this implementation is shown below in Table I. It was the lowest security level available, and so I thought it would be the easiest to implement. The value of p is 27,779, which sets the polynomial ring for operations as $F_2[x]/(x^p + 1)$. The parameter $n0$ denotes the number of blocks in the matrix $M$, which is used as the public key. Also, there are $n0 - 1$ multiplications during encoding.

TABLE I
PARAMETER SET CHOSEN FOR LEDAKEM

| *Cat*[1] | $\mathbf{n_0}$ | $\mathbf{p}$ | $\mathbf{d_v}$ | $\mathbf{m_0, \ldots, m_{n_0-1}}$ | $\mathbf{t}$ | *DFR* |
|---|---|---|---|---|---|---|
| 1 | 2 | 27,779 | 17 | [4,3] | 224 | $8.3\times10^{-9}$ |

[1]NIST security category for the PQC standardization effort.

### D. Motivation

The author's of LEDAKEM seek to use QC-LDPC codes to reduce the size of the public and private key pairs. The major advantage they claim over other code-based schemes is using only one row of the parity-check matrix as the public key. This is possible because each row is a cyclic shift of the previous one, and therefore, only the first row is required to perform all computations. Consequently, the transmission and memory requirements are reduced significantly compared to other code based schemes. $M$. Q-decoder is performed as a modified version of the original Bit-Flipping (BF) decoder [2] algorithm with the exploitation to the most correction of QC-LDPC.

## II. DESIGN OVERVIEW

### A. Design Methodology

In order to identify the most time consuming operation I performed the profiling on the reference implementation, which was submitted to NIST on November 30th, 2017. The platform used was the Zynq UltraScale board that has an ARM Cortex-A53 processor running at a 1.2GHz clock frequency. The results of profiling are shown in Fig. 1, where decoding takes 98% of decapsulation. The most time consuming operation in decoding is the Qdecode algorithm described in [3].

Based upon the software analysis, my target of a hardware component capable of executing the Qdecode algorithm, which will be described in detail below. These components can be used in a full sotware/hardware (SW/HW) codesign implementation, which is beyond the scope of this paper.

### B. Q-decoder Hardware

In LEDAkem, the Q-decoder algorithm promises high decoding speed based on QC-LDPC codes. In addition, compared to classic bit flipping, it exploits the utmost correction power in a small-sized integer matrix transposed H and a private integer matrix Q. Both matrix sizes are constructed based on variable nodes ( constant minimum weight in every column) and check nodes (constant minimum weight in every row) in the parity check matrix. The decoding is not necessary to check every element in the parity check matrix, only wherever its value is one. Therefore, the decoding execution time can be less due to less iterations. In addition, the procedure does not demand on computing matrix inverse and still works well even when the error distribution is not uniform. In this paper, I directly convert the pseudo-code given by the authors in [3] and convert it in to a hardware implementation.

### C. Software/Hardware Co-design

In order to reduce the execution time of software, I propose to use SW/HW codesign, where the reference software replaces the function calls bit flipping decoding with the appropriate calls to hardware component decoder hardware components. Through this process I will be able to gauge the overall speed up gained through using hardware to replace the most time consuming operations.

## III. HARDWARE IMPLEMENTATION DETAILS

### A. Q-decoder

In this section, a hardware implementation of the Q-decoder algorithm is presented based on [3]. The interface diagram is shown in Fig. 2. First, read_priv_synd and read_post signals must be set high to initialize internal signals. Following this the start signal must be set high to begin the bit-flipping decoding process. The output is the err_out vector after the process is complete. The bit flipping algorithm has the goal to "flip" the bit of both error vector (a share key) and a syndrome until either the syndrome equals zero or reaching the maximum iteration.

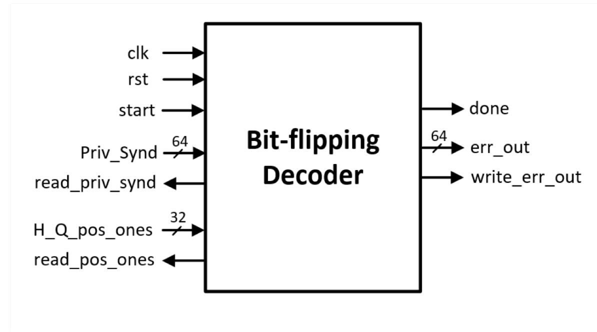


Fig. 2. Bit-flipping Decoder Interface

The decoder implementation in hardware consists of two ROMs, seven RAMs, five counters and four registers, which perform different tasks in three distinct parts. Fig 3 shows the block diagram of the circuit that finds unsatisfied parity checks. The input is the private syndrome, which is passed to the current syndrome in the first stage. Using counters, it will loop through to find unsatisfied one values of the current syndrome. The checking process will continue to check which column of H contributes to the most numbers of unsatisfied one values of the current syndrome. And, the unsatisfied parity check will be updated every iteration.

Next, the value of the threshold can be obtained as using a piecewise constant function of the current syndrome weight and the code parameters. The circuit for this operation is shown in Fig 4. For efficiency reasons, the function is precomputed and stored as a lookup table (LUT) containing pairs (weight, threshold). The Q-decoder computes the weight of the syndrome by computing the Hamming weight of s and determines the highest weight W_h among the ones in the lookup table. The threshold for the similarity is selected as the one paired to in the LUTs.

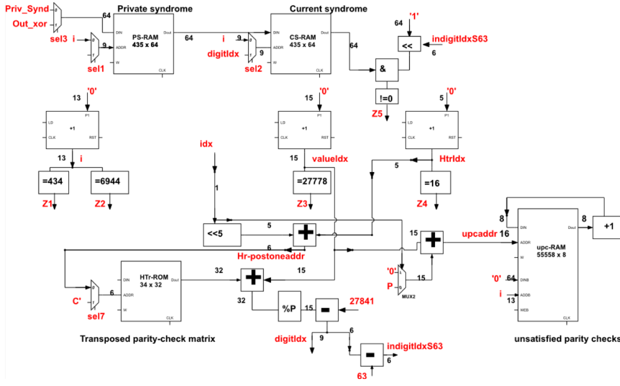


Fig. 3. Finding Unsatisfied Parity-Checks

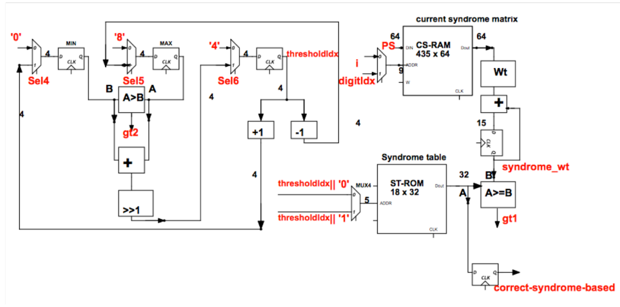


Fig. 4. Computing & Mapping Syndrome Weight

Lastly, the Q-decoder exploits the knowledge of the secret matrix QT to estimate if a bit flip should be performed. A circuit this operation is shown in Fig 5. This circuit depicts that for each bit of e being decoded a measure of correlation between the patterns of ones of a row of QT, blockwise cyclically shifted by the position of the bit of e itself, and the unsatisfied parity checks vector. If the correlation metric is above a given threshold, both the error vector e and the value of the syndrome s for the next iteration iteration are updated.

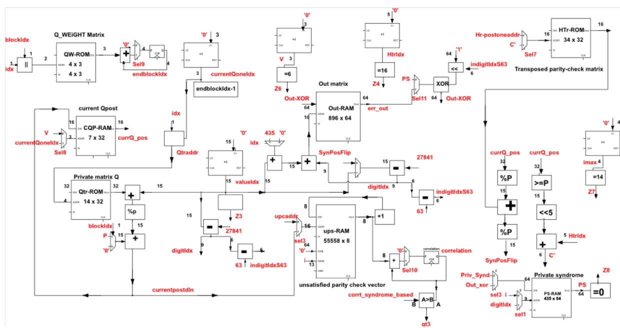


Fig. 5. Toggle error Based on Correlation Vs Threshold

## IV. Results

Implementation results for the Zynq Ultrascale board are shown in this section. The total time for execution is determined for the reference software using a clock frequency of 1.2 GHz, and the hardware running at 100 MHz.

### A. Q-decoder

Based on the block diagram, the synthesis prediction time can be formulated and calculated as shown below in the equation for "# of cycle".

$$\# \, of \, cycle = Iter_Max(T_{finding_{upc}} + T_{compute_{syn_{weight}}} + T_{corr,bf})$$

$$= Iter_max\left(\left(\frac{P}{64} + \frac{n_0 p}{8} + \frac{n_0 P d_v}{64}\right) + \left(\frac{P}{64} + \log_2(size(thre_{table}))\right) + \left(\frac{n_0 p}{64}(no(M_0 + M_1) + (1 + Md_v))\right)\right)$$

$$\# \, of \, cycle = \mathbf{522,703} \, cycles$$

The Q-decoder hardware results are shown in Table II. The speedup in hardware is about 16.5x the software. Also, I can see from the equation above, that the decoding time should scale only linearly with p.

TABLE II
SPEED-UP IN DECODING

| *Implementation* | *Area (LUTs)* | *Flip-Flops* | *Time (ms)* | *Speedup* |
|---|---|---|---|---|
| Ref. Software | - | - | 85.14[1] | 1 |
| Q-decoder in HW | tbd | tbd | 5.23[2] | 16.5 |

[1]Determined from percentage of clock cycles for only decoding.
[2]Total time determined using a clock frequency of 100 MHz.

## V. Conclusion and Future Work

I have demonstrated a speed up of 16.5x in decoding by using a hardware Q-decoder component, respectively. Additionally, it is clear that larger values of p, the polynomial ring size, will increase the area and time requirements decoding.

It appears that choosing a smaller polynomial ring would definitely help making a more efficient hardware implementation. Having elements in such a large ring restrict the amount of parallelism we can exploit, since any amount of parallelism would cost a lot of area.

Additionally, using vivado ver. 2017.2, a verification using test vectors generated using reference software implementation is left as future work, and will help to precisely measure the speed-up.

Note: Some results are still pending and are marked tbd above. They are taking an unusually large amount of time to generate, but will be updated asap, if possible.

## References


[1] L. Chen, D. Moody, and Y.-K. Liu. Post-quantum cryptography standardization - post-quantum cryptography | CSRC. [Online]. Available: https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization

[2] R. Gallager, "Low-Density Parity-Check Codes," *IRE Transactions on information theory*, vol. 8, no. 1, pp. 21–28, 1962.

[3] M. Baldi, A. Barenghi, F. Chiaraluce, G. Pelosi, and P. Santini, "LEDAcrypt/LEDAkem," original-date: 2018-01-11T12:34:36Z. [Online]. Available: https://github.com/LEDAcrypt/LEDAkem