Received August 10, 2020, accepted August 26, 2020, date of publication August 31, 2020, date of current version September 18, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3020262

Efficient and Scalable FPGA-Oriented Design of QC-LDPC Bit-Flipping Decoders for Post-Quantum Cryptography

DAVIDE ZONI, ANDREA GALIMBERTI, AND WILLIAM FORNACIARI, (Senior Member, IEEE)
Dipartimento di Elettronica Informazione e Bioingegneria (DEIB), Politecnico di Milano, 20133 Milan, Italy

Corresponding author: Davide Zoni (davide.zoni@polimi.it)

This work was supported by the EU H2020 "RECIPE" Project under Grant 801137.

ABSTRACT Considering code-based cryptography, quasi-cyclic low-density parity-check (QC-LDPC) codes are foreseen as one of the few solutions to design post-quantum cryptosystems. The bit-flipping algorithm is at the core of the decoding procedure of such codes when used to design cryptosystems. An effective design must account for the computational complexity of the decoding and the code size required to ensure the security margin against attacks led by quantum computers. To this end, it is of paramount importance to deliver efficient and flexible hardware implementations to support quantum-resistant public-key cryptosystems, since available software solutions cannot cope with the required performance. This manuscript proposes an efficient and scalable architecture for the implementation of the bit-flipping procedure targeting large QC-LDPC codes for post-quantum cryptography. To demonstrate the effectiveness of our solution, we employed the nine configurations of the LEDAcrypt cryptosystem as representative use cases for QC-LDPC codes suitable for post-quantum cryptography. For each configuration, our template architecture can deliver a performance-optimized decoder implementation for all the FPGAs of the Xilinx Artix-7 mid-range family. The experimental results demonstrate that our optimized architecture allows the implementation of large QC-LDPC codes even on the smallest FPGA of the Xilinx Artix-7 family. Considering the implementation of our decoder on the Xilinx Artix-7 200 FPGA, the experimental results show an average performance speedup of 5 times across all the LEDAcrypt configurations, compared to the official optimized software implementation of the decoder that employs the Intel AVX2 extension.

INDEX TERMS QC-LDPC codes, bit-flipping decoding, code-based cryptography, post-quantum cryptography, applied cryptography, FPGA, hardware design.

I. INTRODUCTION

The recent advances in quantum computing pose a serious threat to traditional public-key cryptography, whose security relies on the hardness of factoring large integers and of computing discrete logarithms in a cyclic group. In fact, Shor's algorithm [1] can compute the integer factorization and the discrete logarithm operations in polynomial time on a quantum computer, thus dramatically reducing the security margin of the current public-key cryptography primitives. To cope with this risk, the National Institute of Standards and Technology (NIST) is in the process of evaluating and standardizing novel quantum-resistant cryptosystems. The design of such cryptosystems must rely on computationally hard problems for which no polynomial-time solutions are possible even by leveraging quantum computers. In this scenario, code-based cryptography emerged as one of the few promising frameworks, together with lattice-based and hash-based cryptography [2], to design post-quantum cryptosystems. The security of code-based cryptography relies on the hardness of decoding a syndrome obtained with a random linear block code, i.e., the syndrome decoding problem [3], that has been proven to be a NP-complete problem. Moreover, the best solvers for this problem still offer exponential complexity even when implemented on quantum computers [4]. To this end, the syndrome decoding problem is widely assumed to have no polynomial-time solutions even on quantum computers. McEliece [5] proposed the first cryptosystem relying on the hardness of the syndrome decoding problem, although the huge state-space representation due to

The associate editor coordinating the review of this manuscript and approving it for publication was Yu-Chi Chen.

the use of Goppa codes prevented its wide adoption. However, the post-quantum resistance of the syndrome decoding problem motivates a huge amount of research to find out code families with a more efficient state-space representation that allows to reduce the size of the cryptographic key pairs. In this scenario, quasi-cyclic low-density parity-check (QC-LDPC) [6] and quasi-cyclic moderate-density parity-check (QC-MDPC) [7] codes, emerged as viable alternatives to ensure post-quantum resistance, while providing an efficient state-space representation. Such codes are at the core of two post-quantum candidate cryptosystems, i.e., LEDAcrypt [8] and BIKE [9], that have been recently admitted to the third round of the NIST' post-quantum competition. In this scenario, it is crucial to provide efficient hardware support for such quantum-resistant public-key cryptosystems, since their available software implementations [8], [9] reveal the impossibility to cope with the required performance. Considering the wide range of scenarios requiring the use of cryptographic primitives, a goal of the NIST competition is to ensure the possibility of implementing the selected post-quantum cryptosystems on the largest variety of computing platforms. To this end, for each proposal the software must be coded in the C language while a set of commercial FPGAs has been proposed as the reference hardware technology. The software realization allows to deploy the cryptosystem on a wide set of computing platforms ranging from embedded microcontrollers to servers and High Performance Computing (HPC) platforms. The FPGA implementation constitutes a common hardware technology to provide a fair comparison among different hardware solutions. In particular, the quality of the cryptosystems can be assessed independently from any specific optimization or technology, possibly made available by large silicon providers participating to the NIST post-quantum competition.

From the computational point of view, the binary polynomial multiplication and the syndrome decoding dominate the encryption and decryption primitives of QC-LDPC-based cryptosystems, respectively. However, the hardware primitives to support both the binary polynomial multiplication and the syndrome decoding problem in QC-LDPC codes are meant for small key-sizes, i.e., few thousands of bits at most, and mostly targeting telecommunication applications. Consequently, they cannot be readily employed to support QC-LDPC codes for post-quantum cryptography, for which the key-sizes are in the range of dozens of thousands of bits and the operativity is expected far beyond the range supposed for telecommunication applications. To the best of our knowledge, the work in [10] presents a flexible and scalable binary polynomial multiplier conceived for post-quantum QC-LDPC-based cryptosystems, while no equivalent solution is available to support the syndrome decoding of large codes.

To this end, the manuscript proposes an efficient and scalable bit-flipping architecture to support the decoding in QC-LDPC codes for post-quantum cryptography. We note that the literature contains several hardware implementations of QC-LDPC decoders. However, the majority of them are meant to support QC-LDPC codes in the field of telecommunications, thus preventing the use of such hardware solutions in post-quantum cryptography. More in detail, the performance of the available hardware decoders for current QC-LDPC codes is achieved by exploiting the channel information. This characteristic makes the use of such decoding algorithms unfeasible in the design of QC-LDPC decoders for post-quantum cryptosystems. In fact, such cryptographic primitives must be employed in scenarios where the channel information is not available, e.g., encryption of digitally stored data.

As a consequence, the bit-flipping algorithm represents the most widely adopted decoding scheme for two reasons. First, it avoids relying on the information from the physical medium, thus allowing the cryptosystem to operate in a wider range of scenarios. Second, it allows an efficient hardware implementation without employing the expensive real-value computation support required to exploit the information from the physical medium [11]. In particular, the real-value computation increases the complexity of the decoders thus limiting its scalability against large QC-LDPC codes [12]. It is important to note that both the LEDAcrypt [8] and the BIKE [9] cryptosystems, that are finalists of the NIST competition, make use of the bit-flipping decoding procedure.

In the following, the contributions of the manuscript are highlighted in Section I-A, while the background on QC-LDPC codes is detailed in Section I-B. The rest of the manuscript is organized in four parts. Section II details the state-of-the-art related to the bit-flipping decoding. The proposed decoder design is discussed in Section III and Section IV presents the experimental results. Finally, some conclusions are drawn in Section V.

A. CONTRIBUTIONS

The manuscript presents an FPGA-optimized hardware design methodology to implement efficient and scalable QC-LDPC bit-flipping decoders for post-quantum cryptography. We note that we are not proposing a novel post-quantum QC-LDPC cryptosystem. The crucial contribution of our research is the hardware implementation of a family of decoders that exploits the bit-flipping decoding algorithm and allows to accelerate any QC-LDPC-based cryptosystem. The assessment has been carried out against the Xilinx Artix-7 FPGA family, since it is the recommended target technology for any hardware implementation within the NIST post-quantum cryptography competition. Our solution allows the designer to trade the resource utilization with the obtained performance in terms of throughput, adding two relevant contributions with respect to the state-of-the-art:

- Efficient computing architecture - By leveraging the sparseness and quasi-cyclic properties of the considered QC-LDPC codes used to design post-quantum cryptosystems, the proposed architecture is optimized to efficiently compute the time-consuming vector-matrix multiplications that is at the core of the bit-flipping decoding algorithm. Moreover, the level of parallelism to perform each vector-matrix multiplications is a design-time parameter. According to the NIST guidelines, we demonstrated the effectiveness of our solution by implementing each configuration of the LEDAcrypt [8] cryptosystem on the entire family on the Xilinx Artix-7 FPGAs. The comparison of our solution implemented on the Xilinx Artix-7 200 against the performance-optimized software reference solution, which exploits the *Intel AVX2* extension, shows an average speedup of 5 times across the entire range of the LEDAcrypt configurations.

- Scalable architecture - The proposed decoder allows to optimally select the resource-performance trade-off regardless of the parameters of the underlying code. Indeed, the bandwidth of the decoder datapath is a design-time parameter. Moreover, the use of the FPGA block RAMs (BRAMs) instead of flip-flops to store the inputs, the intermediate values and the results allows to manage underlying codes with a codeword length ranging from few bits up to dozens of thousands bits, even on small FPGA targets. In particular, our exhaustive design space exploration demonstrates the possibility of implementing a performance-optimized decoder, for each configuration of the LEDAcrypt cryptosystem, over the entire Xilinx Artix-7 family of mid-range FPGAs.

B. BACKGROUND ON QC-LDPC CODES

Quasi-Cyclic Low-Density Parity Check (QC-LDPC) codes emerged as a viable solution to design post quantum cryptosystems due to their inner structure that allows to greatly reduce *i)* their software and hardware implementation complexity, and *ii)* the size of the used secret keys. The rest of this part reviews the basic elements of the QC-LDPC codes to ease the reading of the following parts of the manuscript.

1) LOW-DENSITY PARITY-CHECK CODES

Low-density parity-check (LDPC) codes are linear error correction codes introduced by Gallager [13], that allow to transmit messages over noisy channels. However, the recent advance in quatum-computing highlighted the possibility of using such codes to support the design of post-quantum cryptosystems.

Without loss of generality, we are focusing on binary LDPC codes since they are the most widely adopted in code-based cryptography. Starting from the definition of the Galois field of order 2, i.e., GF_2, we denote as GF_2^k the k-dimensional vector space defined over GF_2. To this end, a binary linear code denoted as $\mathbf{C}(n,k)$ is defined as a mapping which univocally associates each binary k-tuple, i.e., the information vector, to a binary n-tuple, i.e., the codeword (see Equation (1)).

$$\mathbf{C} : GF_2^k \rightarrow GF_2^n \tag{1}$$

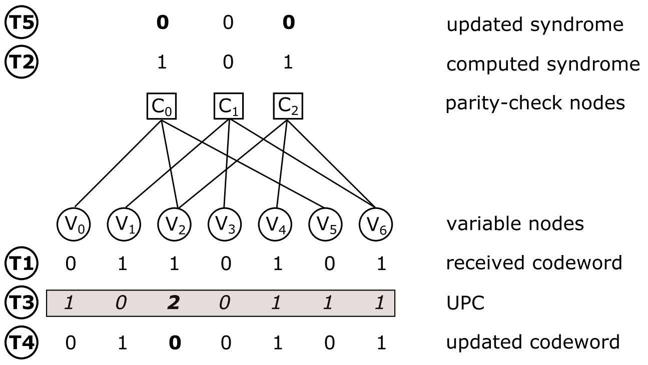


FIGURE 1. Tanner graph of an LDPC code with $n = 7$ and $r = 3$. The steps of the bit-flipping algorithm used to correct the bit of the codeword associated to the V_6 variable node are marked from T_0 to T_4.

In general, an LDPC code $\mathbf{C}(n,k)$ is defined by its parity-check matrix H that has r rows and n columns, where $r = n - k$ [11]. Such matrix can be graphically represented by the associated Tanner graph, that is a bipartite graph made of n variable nodes and r check nodes. A codeword bit is associated to each variable node, while each parity-check bit is associated to a check node. In particular, the set of all the parity-check bits defines the so-called syndrome vector s. Each $h_{i,j}$ element of the H matrix set to 1 indicates that the j-th bit in the codeword participates in the i-th parity check equation. The i-th syndrome bit is therefore computed as the bitwise XOR of all the codeword bits involved in the i-th parity-check equation. For example, the Tanner graph of a binary LDPC code with $n = 7$ and $r = 3$ is depicted in Figure 1, while Equation (2) defines the corresponding H matrix. For each parity-check node, the number of incoming edges is equal to the number of ones in the corresponding row of the H matrix, while the number of incoming edges to each variable node is equal to the ones in the corresponding column of the H matrix.

$$H = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \tag{2}$$

Once a codeword is received, the decoding procedure analyzes the parity-check equations by generating the syndrome s of the codeword c through H, according to Equation (3).

$$s = c \cdot H^T \tag{3}$$

The received codeword is considered to be error-free when the syndrome is a null vector. In case the received codeword contains errors, the error correction algorithm iteratively recovers such errors in the codeword until either all the parity check equations are satisfied, i.e., the syndrome is the null vector, or the codeword is declared unrecoverable and, thus, it has to be retransmitted by the sender. We note that regardless of the use of soft-, e.g., Logarithmic-Likelihood-Ratio Sum-Product Algorithm (LLR-SPA) [11], or hard-decision, e.g., bit-flipping (BF), error correction algorithms, all the available codeword decoding algorithms implement an iterative procedure. Soft-decision decoders represents the most employed decoding solutions in telecommunication applications due to their superior performance coming from the exploitation of the availalbe channel information [11]. In contrast, the bit-flipping algorithm represents the most employed decoding solution when no medium information is available, the floating point support is not available, and an efficient decoder design is required [11]. Considering its vast applicability and the possibility of delivering efficient decoders, the bit-flipping decoding algorithm is the sole solution adopted by the code-based cryptosystems participating to the NIST post-quantum competition.

Figure 1 depicts an example of the iterative bit-flipping decoding procedure, made of one iteration and five time-steps, i.e., T_1-T_5, to correct a received codeword by means of the bit-flipping algorithm. At time T_0 the sender transmits the codeword $c = 0100101_b$, that is received with an error by the receiver at time T_1 as $c = 0110101_b$. In this example, the received codeword contains an error in the bit associated to V_2, i.e., its value is 1 instead of 0. The bit-flipping decoding algorithm associates each bit of the received codeword to the corresponding variable node, and the syndrome is computed at time T_2 according to Equation (3). We note that the syndrome is made of three bits, i.e., one bit for each parity-check node. In particular, the parity-check equations corresponding to the parity-check nodes C_0 and C_2 are not satisfied and, thus, the error-recovery strategy of the bit-flipping algorithm takes place. For each iteration, the bit-flipping algorithm can flip one or more bits in the received codeword according to the information contained in the unsatisfied parity-checks (UPC) vector. For each variable node, the corresponding UPC value corresponds to the number of failed parity-check equations, i.e, the number of connected parity-check nodes whose associated syndrome bit has a value equal to 1. The UPC vector is defined by Equation (4) and it is computed at time T_3 (see Figure 1).

$$UPC = s \cdot H \tag{4}$$

Starting from the UPC vector, the bit-flipping algorithm flips each bit in the codeword for which the corresponding UPC value is above a certain threshold. We note that the threshold selection is a parameter of the bit-flipping algorithm and it strongly depends on the specific LDPC code. The threshold is selected to minimize the trade-off between the decoding failure rate (DFR), i.e., the number of times the algorithm fails decoding the received codeword, and the number of decoding iterations. At time $T4$, the codeword is updated by flipping the bits corresponding to UPC values greater or equal to the threshold (which, as an example, can be set to the maximum of the values assumed by the UPC vector). In our case, the codeword bit corresponding to the variable node V_2 is flipped from 1 to 0, since its UPC value is equal to 2 (the maximum value assumed by the UPC vector). Finally, at time $T5$, the syndrome bits associated to the flipped codeword bits are also flipped, which is a faster way to update the syndrome vector than recomputing the vector-matrix multiplication in Equation (3). In the example in Figure 1, the syndrome bits corresponding to parity-check nodes C_0 and C_2 are both flipped from 1 to 0. Being the syndrome after $T5$ equal to the null vector, the decoding procedure can be interrupted since the codeword has been certainly recovered correctly, i.e., all the transmission errors have been corrected. Otherwise, if the syndrome vector were not a null vector, the iterative procedure would have been continued by repeating the steps executed at the time-steps from $T3$ to $T5$.

2) CIRCULANT MATRICES

A circulant matrix is defined as a square matrix where each row is obtained by shift-rotating the preceding row to the right by one position. By construction, a circulant matrix is therefore regular, i.e., both columns and rows have constant weight. A $p \times p$ circulant matrix A, where each element is denoted as a_i with $i \in [0, \ldots, p-1]$, is shown in Equation (5).

$$A = \begin{bmatrix} a_0 & a_1 & a_2 & \ldots & a_{p-1} \\ a_{p-1} & a_0 & a_1 & \ldots & a_{p-2} \\ a_{p-2} & a_{p-1} & a_0 & \ldots & a_{p-3} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ a_1 & a_2 & a_3 & \ldots & a_0 \end{bmatrix} \tag{5}$$

We note that the arithmetic of circulant matrices of size p is isomorphic to the arithmetic of the polynomials modulo $x^p - 1$ over the same field as the coefficients of the circulant matrices. The circulant matrix A is therefore isomorphic to a polynomial $a(x)$ with coefficients given by the elements of the first row of the matrix, as shown in Equation (6).

$$a(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \ldots + a_{p-1} \cdot x^{p-1} \tag{6}$$

Considering the case of binary linear block codes, the arithmetic of $p \times p$ circulant matrices over $\mathbb{Z}_2$ can be substituted by the arithmetic of polynomials in $\mathbb{Z}_2[x]/(x^p + 1)$, which provides a reduction in the storage requirements and a faster execution of the arithmetic operations.

3) QC-LDPC CODES

Quasi-cyclic (QC) codes are linear block codes $\mathbf{C}(n, k)$ whose parity-check matrices H are composed of $r_0 \times n_0$ circulant blocks, each of size $p \times p$, where $n = n_0 \cdot p$, $k = k_0 \cdot p$ and $r_0 = n_0 - k_0$. Considering post-quantum code-based cryptosystems, we focus on the $r_0 = 1$ case, for which the corresponding family of QC codes has a rate of $(n_0 - 1)/n_0$. In this case, the parity check matrix is defined by Equation (7), where each block H_i with $i \in [0, \ldots, n_0 - 1]$ is a circulant matrix of size $p \times p$.

$$H = \begin{bmatrix} H_0 & H_1 & \ldots & H_{n_0-1} \end{bmatrix} \tag{7}$$

The structure of quasi-cyclic codes enables efficient encoding implementations by means of fast binary polynomial multipliers. However, the lack of an efficient decoding support, due to the inherent structure of the H matrix, prevented their widespread use for a long time. QC-LDPC codes have been explored as a particular class of quasi-cyclic codes that are characterized by parity-check matrices which are well-suited for LDPC decoding algorithms, i.e., the matrix is sparse and it avoids the presence of short length cycles in the associated Tanner graph [11]. In particular, QC-LDPC codes combine the efficient decoding and the low decoding failure rate (DFR) of LDPC codes, with the efficient encoding and the small memory footprint of QC codes.

II. RELATED WORKS

Traditionally, LDPC and QC-LDPC codes are used in wired, e.g., 10GBase-T Ethernet [14], and wireless, e.g. WiMax (IEEE 802.16e) and WiFi (802.11n) [15], telecommunication applications, due to their superior error-correction capabilities [11]. However, the recent advances in quantum computing highlighted the possibility of employing the class of QC-LDPC codes to design quantum-resistant cryptosystems [2]. In particular, two of the 17 cryptosystems admitted to the final round of the post-quantum U.S. NIST contest employ QC-LDPC codes [8], [9].

From the decoding point of view, the state of the art contains several proposals addressing the optimized design of decoders to support QC-LDPC codes. In the following we classified them in two main groups: i) soft-decision decoders, e.g., Belief Propagation (BP), Sum-Product Algorithm (SPA) and their variations, that employ a message passing structure, and ii) hard-decision decoders, i.e., bit-flipping algorithms, designed to offer a simple decoder implementation. Traditionally, soft-decision decoders offer superior decoding performance than hard-decision ones, i.e., bit-flipping solutions, thanks to the use of the channel information. In contrast, bit-flipping decoders have a favorable less complex design. The rest of this section discusses the state-of-the-art proposals on decoding, targeting QC-LDPC codes. We note that the review aims to highlight the main limitations and constraints that prevent the use of current state-of-the-art solutions in the design of QC-LDPC decoders for post-quantum cryptography.

Among the soft-decision decoders, [16] proposed a FPGA-based QC-LDPC decoder for the Chinese Digital Television Terrestrial Broadcasting (DTTB) standard, which is based on the soft-decision min-sum algorithm. Reference [17] describes a parallel GPU implementation of the soft-decision min-sum decoder for QC-LDPC codes, targeting both the WiMax and WiFi standards. Despite the interesting performance of the proposed parallel GPU decoder, the underlying QC-LDPC $\mathbf{C}(n, k)$ codes for WiFi and WiMax have the (n, k) pair of parameters equal to (1944, 972) and (2304, 1152) for WiFi and WiMax, respectively, thus tens of times smaller than the ones employed in post-quantum QC-LDPC cryptosystems. An additional hardware implementation of a soft-decision decoder for the 802.11n WiFi standard, thus targeting small codes, is proposed in [18]. In contrast, [19] presents a 90nm CMOS implementation of a soft-decision decoder for QC-LDPC codes with n values up to 96000 bits. Despite the code size is aligned with the one employed in current QC-LDPC-based cryptosystems, the decoder in [19] is tailored to a specific code structure that is intended for telecommunications. To this end, the underlying code cannot offer the security margin required by post-quantum code-based cryptosystems.

Considering telecommunication applications, the use of soft-decision decoders represents the optimal choice due to the possibility of implementing a system approaching the channel capacity limit [11]. However, such superior performance is achieved by leveraging the channel information in the decoding procedure. To this end, QC-LDPC codes meant for post-quantum cryptosystems can not employ soft-decision algorithms, since the cryptosystem is expected to operate even when the channel information is not available, e.g., encryption and decryption of digitally stored data. Moreover, the complexity of soft-decision decoders limits their scalability in supporting large QC-LDPC codes [12]. The state-of-the-art contains several families of proposals, i.e., Weighted Bit Flipping (WBF) [20], Modified WBF (MWBF) [21], and Gradient Descent Bit Flipping (GDBF) [22], aiming at optimizing the performance of the baseline bit-flipping algorithm, i.e., hard-decision decoders. However, for each of them, the performance improvement is obtained by leveraging some sort of channel information, thus preventing their use in the design of QC-LDPC decoders for post-quantum cryptosystems [12]. In summary, the baseline bit-flipping algorithm represents the most important candidate to deliver hardware accelerated decoders for quantum-resistant QC-LDPC cryptosystems. We note that such design choice is also supported by the fact that all the QC-LDPC-based cryptosystems that entered the final round of the post-quantum NIST competition make use of the vanilla bit-flipping decoding procedure.

Among the hard-decision decoders, [23] presents a hardware implementation of the Key Encapsulation Mechanism (KEM) for the LEDAcrypt cryptosystem submitted to the first round of the NIST competition. Such version of the cryptosystem proposes a variant of the bit-flipping decoder, i.e., Q-decoder, that has been dismissed due to a set of security vulnerabilities in the theoretical decoding scheme [24]. In fact, the current LEDAcrypt submission to the third round of the NIST competition employs a baseline bit-flipping decoder, thus making the work in [23] obsolete. Reference [25] proposes a lightweight implementation of a bit-flipping decoder for QC-LDPC codes. Despite the fact that the solution in [25] does not offer a configurable area-performance trade-off, it suffers two other limitations. First, the decoding performance is in the order of tens of milliseconds, while our solution offers a performance that is 100 times better, on average. Moreover, the design is limited to small QC-LDPC codes that offer an 80-bit security level, while our solution targets larger QC-LDPC codes to achieve a security level between 128 and 256 bits. The BIKE round 3 specification document [9] discusses the decoder implementation of the BIKE cryptosystem, that leverages a light variant of the bit-flipping algorithm. In particular, the baseline bit-flipping algorithm has been slightly modified in its first iteration to conditionally perform an additional error correction task. We also note that a software implementation of the BIKE bit-flipping decoder employing the *Intel AVX512* extension is discussed in [26]. In a similar manner, both the reference *C11* and the optimized *Intel AVX2* software implementations of the bit-flipping decoder employed in the current version of the LEDAcrypt cryptosystem, are discussed in its third round specification document [8].

We note that despite the huge effort in designing efficient bit-flipping decoders for post-quantum QC-LDPC cryptosystems, all the available solutions are software-implemented, while the available hardware solutions target telecommunications QC-LDPC codes. To this end, it is of paramount importance to provide efficient and scalable hardware decoders to support the emerging QC-LDPC cryptosystems, since the available software solutions reveal the impossibility to cope with the required performance also considering the stiff increase of the key-size expected in the near future.

III. METHODOLOGY

This section describes the architecture of an efficient and scalable bit-flipping decoder for large QC-LDPC codes employed in the design of post-quantum cryptosystems. The efficiency is achieved by means of an optimized architecture to perform the time-consuming vector-matrix multiplications within the decoding procedure. In addition, the proposed decoder is meant to scale across a wide range of FPGAs rather than being hard-coded to a specific target. In particular, the configurable architecture allows to implement the decoder for large QC-LDPC codes even on resource-constrained FPGA targets. The rest of this section discusses the proposed bit-flipping architecture. Section III-A presents the architectural details of the dual-memory component at the core of our solution, while Section III-B is devoted to the complexity analysis.

The architecture of the proposed bit-flipping decoder is built upon the bit-flipping procedure described in Algorithm 1. The main bit-flipping decoder function, i.e., `BFDecoding`, executes a predefined number of iterations (see lines $3-9$ in Algorithm 1) to produce the error vector (`e`) and the decoding failure boolean flag (`fail`) as outputs.

Each iteration consists of a sequence of six operations: *i)* threshold computation (line 4), *ii)* unsatisfied-parity-check (UPC) computation (line 5), *iii)* error bit-flip computation (line 6), *iv)* error update (line 7), *v)* syndrome bit-flip computation (line 8), and *vi)* syndrome update (line 9). From the computational complexity viewpoint, the UPC computation (see line 5 in Algorithm 1) and the syndrome bit-flip computation (see line 8 in Algorithm 1) represent the two most critical operations. In fact, both the UPC and the syndrome bit-flip computations impose a vector-matrix multiplication, i.e., the former between the syndrome and the H matrix and the latter between the error bit-flip vector and the H matrix. In particular, the UPC computation is performed in the integer domain, while the syndrome bit-flip computation is instead performed in the binary domain. Concerning the remaining operations, the threshold computation procedure is usually customized to minimize both the decoding failure rate (DFR) of the underlying QC-LDPC code and the number of iterations required to decode a codeword, and it is negligible from the computational viewpoint. Finally, the other three operations, namely the error bit-flip computation and the update of both the error and the syndrome vectors, are also negligible from the computational point of view, since they consist in vector operations with a linear complexity.

Algorithm 1 Bit-Flipping Decoding Procedure for LDPC Codes. H Is a $p \times n$ Parity Check Matrix, Where $n = p \cdot n_0$. s Is a p-Bit Syndrome Vector. e Is an n-Bit Error Vector. $fail$ Is a 1-Bit Flag That Is Set in Case of Decoding Failure

```
1:  function [$e, fail$] BFDecoding($H, s$)
2:      $e = 0; fail = 0$;
3:      for $i \in 1 : iter_{max}$ do
4:          $thr = Threshold(s)$;
5:          $upc = s \cdot H$;
6:          $e_{bf} = (upc >= thr)\ ?\ 1 : 0$;
7:          $e = e \oplus e_{bf}$;
8:          $s_{bf} = e_{bf} \odot H^T$;
9:          $s = s \oplus s_{bf}$;
10:     $fail = (s == 0)\ ?\ 0 : 1$;
11:     return $e, fail$;
```

Starting from the bit-flipping procedure, Figure 2 shows the architectural top level view of the proposed decoder (`BF-decoder`). The `BF-decoder` takes the syndrome (s) and the parity-check matrix (H) in input, and it outputs the error vector (e) and the boolean flag ($fail$) to signal any failure in the decoding procedure.

From the computational viewpoint, the BF-decoder is made of two stages to calculate the UPCs (`calcUpc`) and the syndrome bit-flips (`calcBf`). We note that the decoding architecture is optimized by leveraging the sparseness and quasi-cyclic properties of QC-LDPC codes. Indeed, only the positions of the dv ones of the first row of each H_i block are stored, since each H_i block is both a sparse and a circulant matrix.

The `calcUpc` stage takes the syndrome and the blocks of the H matrix in input and it outputs the UPCs and the weight of the syndrome. At the beginning of a new decoding, i.e., when the `isNewDec` signal of `MUX1` is equal to 1, the initial syndrome is received from the primary inputs. In contrast, for each subsequent iteration of the decoding algorithm, the syndrome is updated with the syndrome bit-flip vector generated by the `calcBf` stage. At the start of each iteration, the weight of the syndrome is computed by the `sWeight` module, and its value is used by the `upc2bf` module. The `calcUpc` module sequentially computes the UPCs for each H_i block of the H matrix. Once the UPCs from the H_i block, i.e., UPC_i, have been computed, they are passed to the `upc2bf` module with a bandwidth equal to BW

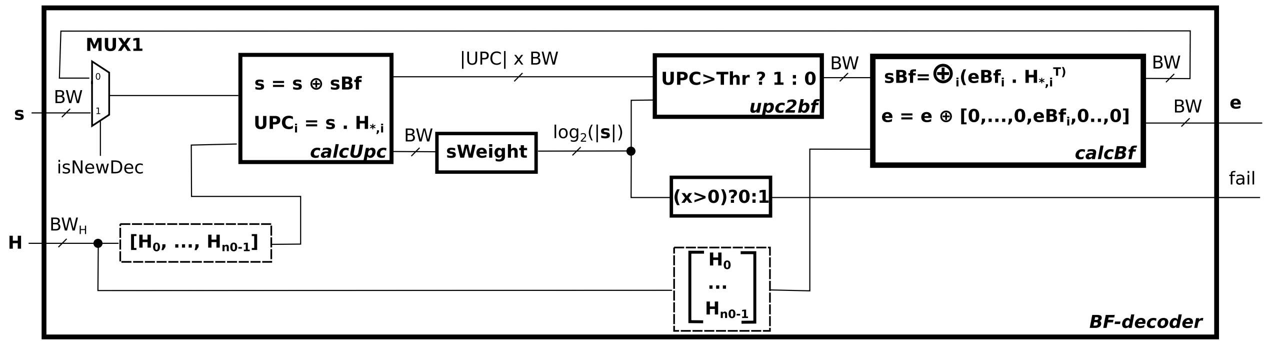


FIGURE 2. Top level view of the proposed bit-flipping decoding architecture.

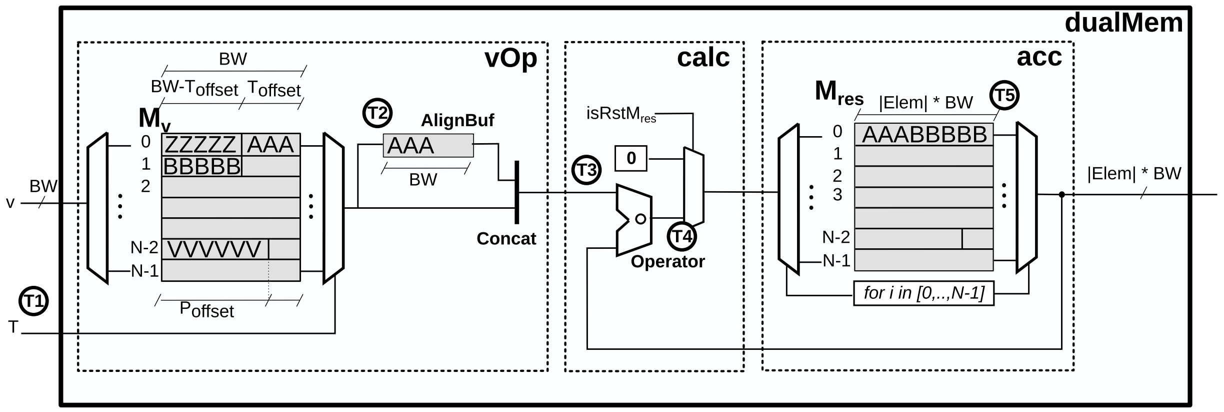


FIGURE 3. Detailed view of the proposed dual-memory architecture.

times the size in bits of the maximum UPC value (which is v). The `upc2bf` module filters the incoming UPCs by comparing them with the UPC threshold to produce, as a result, the error bit-flip vector (eBf_i). We note that the eBf_i vector is fed to the `calcBf` stage to *i)* compute the syndrome bit-flips, and to *ii)* update the error vector.

Within each iteration of the bit-flipping algorithm, the `calcBf` stage updates both the syndrome bit-flips and the error vector starting from any incoming error bit-flip vector (eBf_i). In particular, the chain made of the `calcUpc` and the `upc2bf` modules produces a set of n_0 eBf_i vectors, where each of them corresponds to a specific H_i block of the parity-check matrix. To this end, the `calcBf` stage receives n_0 eBf_i vectors, i.e., one for each H_i block in the H matrix, to compute the fully updated syndrome bit-flip vector (sBf), as well as the update of the error vector.

We note that the error vector e is made by n_0 blocks of size $1 \times p$, thus each error bit-flip vector (eBf_i) updates a portion of the error vector (see the $e = e \oplus [\mathbf{0}, \ldots, \mathbf{0}, eBf_i, \mathbf{0}, \ldots, \mathbf{0}]$ update equation in Figure 2). In contrast, sBf is a $1 \times p$ row-vector obtained by performing the bitwise XOR of all the received eBf_i row-vectors (see $sBf = \oplus_i(eBf_i \cdot H*, i^T)$ equation in Figure 2). At the end of each iteration of the decoding procedure, i.e., lines $4 - 8$ in Algorithm 1, the bitwise XOR between the current syndrome and the syndrome bit-flip vector is performed in the next iteration (see line 9 in Algorithm 1). From the architectural viewpoint, the syndrome update is performed by the `calcBf` module (see the $s = s \oplus sBf$ equation in Figure 2).

A. THE DUAL-MEMORY COMPUTING ARCHITECTURE

Apart from the configurable bandwidth (BW) that is used to trade the performance with the resource utilization, the proposed decoder implements a dual-memory architecture to perform the efficient and scalable computation of the two most time-consuming operations in the bit-flipping decoding procedure, i.e., the UPC computation ($UPC_i = s \cdot H_i$) and the generation of the syndrome bit-flip vector ($sBf_i = eBf_i \cdot H_i^T$).

This section discusses the proposed dual-memory architecture that is meant to perform the efficient vector-matrix multiplication between a vector v and a sparse circulant matrix A. We demonstrate that the use of such an architecture allows to adopt an efficient divide-and-conquer approach in the computation, thus delivering an additional knob to trade the performance with the resource utilization.

Figure 3 shows the `dualMemory` architecture as made of three stages, i.e., `vOp`, `calc`, and `acc`. The operand (vOp) and accumulator(acc) stages implement two memory elements to store the vector v and the partial result vector, respectively. In contrast, the compute stage ($calc$) performs the actual vector-matrix computation starting from the inputs from both the accumulator and the operand stages. The dual-memory architecture receives two inputs, i.e., the vector v and the position of the ones in the A matrix, and outputs the vector resulting from the vector-matrix multiplication. The v vector is actually a primary input of the dual-memory module and it is stored in the memory of the operand stage (M_v). In contrast, the circulant matrix A is never stored nor received as input in its dense representation.

We note that the computational efficiency of the proposed dual-memory architecture sits on the possibility of substituting the time-consuming vector-matrix multiplication with a set of fast shift-rotate additions due to the fact that the A matrix, i.e., the H_i blocks of the H matrix in QC-LDPC codes, is both circulant and sparse. In particular, it is sufficient to store the positions of the ones in the first column of A.

To this end, each T value in input to the dual-memory module represents the position of a one in the first column of the A matrix. For each T value, the dual-memory module performs a shift-rotate of the v vector by T positions before adding the result to the accumulator by means of the compute stage (*calc*). We note that the generic $\circ$ operation performed by the compute stage *calc*, can be specialized depending on the actually required operation.

For example, Equation 8 shows the vector-matrix multiplication between a 4-bit row-vector (b) and a 4×4, binary, circulant matrix (C).

$$\begin{aligned} r &= b \cdot C \\ &= \begin{bmatrix} b_0 & b_1 & b_2 & b_3 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix} \\ &= \begin{bmatrix} b_2 + b_3 & b_0 + b_3 & b_0 + b_1 & b_1 + b_2 \end{bmatrix} \end{aligned} \tag{8}$$

In particular, the sparse representation of the C matrix, i.e., C^{sp}, that is made of the positions of the 1s in its leftmost column, is defined in Equation 9.

$$C^{sp} = \begin{bmatrix} 2 & 3 \end{bmatrix} \tag{9}$$

To this end, the vector-matrix multiplication between b and C^{sp} can be computed as the sum of the dense vector shift-rotated to the left by amounts equal to the elements of C^{sp}. This is shown in Equation 10, where b is the dense vector and the sparse-represented positions of the 1s in C are identified as C_i^{sp}. The $x <<< y$ notation specifies a left shift-rotate of vector x by y positions.

$$\begin{aligned} r &= b \cdot C \\ &= \sum_{i=0}^{v-1} (b <<< C_i^{sp}) \\ &= (b <<< 2) + (b <<< 3) \\ &= \begin{bmatrix} b_2 & b_3 & b_0 & b_1 \end{bmatrix} + \begin{bmatrix} b_3 & b_0 & b_1 & b_2 \end{bmatrix} \\ &= \begin{bmatrix} b_2 + b_3 & b_3 + b_0 & b_0 + b_1 & b_1 + b_2 \end{bmatrix} \end{aligned} \tag{10}$$

From the computational viewpoint, the dual-memory module updates the accumulator's memory with a sequence of five steps for each T value in input. At time $T1$ a new T position is received by the `vOp` module that performs the readout from the M_v memory. We note that the T position can be misaligned with respect to the BW-bit size of each line in the M_v memory, thus the `AlignBuf` in the `vOp` module is used to store the trail of the first readout line at time $T2$, e.g., *AAA* in Figure 3. After the first clock cycle used to align the reads from the M_v memory, the `vOp` module produces BW bits of data for the successive sets of clock cycles required to completely readout the v vector. Each line produced by the `vOp` module is obtained by concatenating the content of the `AlignBuf` with the initial part of the next readout line from M_v. In particular, each readout line from M_v has the first part used to compose the BW-bit output, while the remaining part is stored in the `AlignBuf` buffer to be concatenated in the next clock cycle. Considering the scenario depicted in Figure 3, the `vOp` module outputs the first BW bits, i.e., *AAABBBBB*, at time $T3$. At the same time, the M_{res} memory is completely read starting from line 0. In particular, the content of each line q of M_{res} is combined with the output from the `vOp` module at time $T4$, before being stored at the same q-th line of M_res at time $T5$.

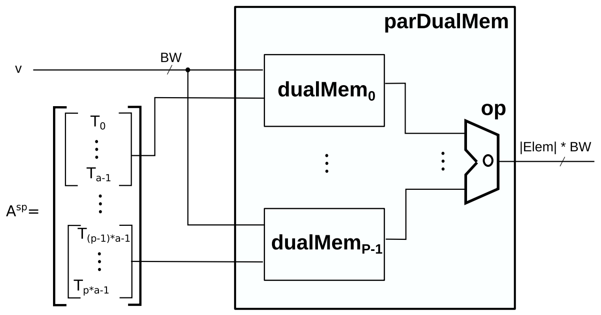


FIGURE 4. Detailed view of the proposed parallel dual-memory architecture.

1) DIVIDE-AND-CONQUER APPROACH

Figure 4 shows the parallel architecture (*parDualMem*) made of a set of dual-memory modules to perform the efficient vector-matrix multiplication. The module takes the vector v and the sparse representation of the binary circulant matrix A (A^{sp}) in input and produces the vector-matrix product in output. Depending on the actual operator implemented in place of the generic $\circ$ one, the size of each element of the output can vary. To this end, the bandwidth of the `parDualMem` module allows to output BW elements of the result at once, while $|Elem|$ identifies the size of the generic element of the result vector. Starting from the sparse representation of the binary circulant matrix A, each dual-memory module receives a subset of positions and, for each of them, it accumulates the shift-rotate of the v vector. The final result is obtained by combining the outputs from all the implemented dual-memory modules operated by the *op* computing block (see Figure 4). We note that the *parDualMem* architecture allows a design-time configurable parallelism ranging from 1 to the number of ones in the first column of the A matrix.

2) THE parDualMem MODULE IN THE DECODER

Within the proposed decoder, the `parDualMem` architecture is employed to efficiently perform the UPC computation in the `calcUpc` module (see $UPC_i = s \cdot H_i$ in Figure 2) and the generation of the syndrome bit-flip vectors in the `calcBf` module (see $sBf_i = eBf_i \cdot H_i^T$ in Figure 2). In particular, the UPC computation, i.e., $upc_i = s \cdot H_i$, corresponds to a vector-matrix multiplication in the integer domain. Thus, the configurable *calc* stage in the *dualMemory* module is customized to perform the integer addition. Differently, the syndrome bit-flip computation, i.e., $s_i^{bf} = e_i^{bf} \odot H_i^T$, corresponds to a vector-matrix multiplication computed in the binary domain, or equivalently, since circulant matrices are isomorphic to polynomials modulo $x^p - 1$, to a binary polynomial (or carry-less) multiplication. To this end, the configurable *calc* stage in the *dualMemory* module is customized to implement the bitwise XOR operator.

We note that each instance of the `parDualMem` module allows to independently configure the level of parallelism between 1 and v, i.e., the number of ones in each column of the parity-check matrix.

B. TIME AND SPACE COMPLEXITY ANALYSIS

This section discusses the complexity analysis of the proposed bit-flipping decoding architecture in terms of both time and space. The goal is to highlight the architectural optimizations that allow to implement a family of decoders for large QC-LDPC codes across a wide range of resource-performance trade-offs.

1) TIME COMPLEXITY

Equation (11) is a 6-parameter equation that defines the time required to perform a complete decoding procedure (T_{dec}), expressed in terms of number of clock cycles. The parameter $iter_{max}$ represents the maximum number of decoding iterations, p is the number of syndrome bits, n_0 is the number of circulant blocks that compose the parity-check matrix H, v is the weight of each column of the H matrix, bw is the bandwidth of the decoder datapath in bits, and par is the parallelism in the UPC and syndrome bit-flips computation. Note that we do not have control over the $iter_{max}$, n_0, p and v parameters, since they are parameters of the QC-LDPC code. In contrast, the scope of our investigation is to provide an efficient and scalable hardware decoder to support the implementation of any already parametrized QC-LDPC cryptosystem.

$$T_{dec} = iter_{max} \times (n_0 + 1) \times \left\lceil \frac{p}{bw} \right\rceil \times \left\lceil \frac{v}{par} \right\rceil \quad (11)$$

More in detail, $iter_{max}$ is a parameter of the decoding algorithm, p, n_0, and v are parameters of the considered QC-LDPC code, while bw and par are configurable parameters of the proposed decoding architecture that can be tuned to explore different resource-performance trade-offs.

Equation (11) is the product of four terms. Once the parameters of the code, i.e., p, n_o, and v, are set, the first term, i.e., $iter_{max}$, defines the maximum number of iterations in the bit-flipping decoding procedure to achieve the required Decoding Failure Rate (DFR). The second term, i.e., $(n_0 + 1)$, accounts for the `calcUpc` and `calcBf` operations across the entire set of blocks in the H matrix. In particular, the decoding architecture is optimized to perform such processing in a pipelined fashion by leveraging two computational aspects. First, the computation on each block of H is independent from all the others. Second, for each block of H, the computations within the `calcUpc` and `calcBf` can be performed independently. Table 1 shows the pipelined execution of the decoder to perform a single iteration when the underlying code features a parity-check matrix H made of three circulant blocks, i.e., the n_0 code parameter is equal to 3. Time is expressed in time epochs, i.e., 1, 2, 3 and 4, where the duration of each epoch depends on the computational time required by the slowest of the `calcUpc` and `calcBf` stages. The pipelined execution allows to reduce the computational time from $2 \times n_0$, if the `calcUpc` and `calcBf` stages were completely serialized, to $(n_0 + 1)$.

TABLE 1. Temporal evolution of the pipelined execution of one iteration of the decoding procedure, when the parity-check matrix H is composed of three blocks ($n_0 = 3$).

Time epoch	1	2	3	4
$\mathbf{H_0}$	calcUpc	calcBf		
$\mathbf{H_1}$		calcUpc	calcBf	
$\mathbf{H_2}$			calcUpc	calcBf

To optimize the performance of the pipelined architecture, the par and bw parameters are set to the same values for both stages. The stages are thus balanced, i.e, they have the same execution time. The third term, i.e., $\left\lceil \frac{p}{bw} \right\rceil$, represents the number of memory lines to be read and written for each 1 position in the H matrix. As shown by Equation (11), the computational time decreases when the bandwidth bw increases. Last, the fourth term, i.e., $\left\lceil \frac{v}{par} \right\rceil$, accounts for the parallel computation of the ones of the H matrix. Indeed, for each block in the H matrix, our decoding architecture allows to configure how many 1 positions of H are processed in parallel in the `calcUpc` and `calcBf` stages.

2) SPACE COMPLEXITY

Equation (12) defines the memory requirement (M_{dec}) of the proposed bit-flipping decoding architecture, expressed as the cumulative memory required by the `calcUpc`, i.e., $M_{calcUpc}$ and the `calcBf`, i.e., M_{calcBf} stages. The memory requirement is provided in terms of number of BRAM memories, that are the de-facto storage memory in the FPGA. We note that the flip-flops, that represent the other type of memory resource in FPGAs, are not accounted for in the rest of the analysis for two reasons. First, their storage capacity is only a tiny fraction of the capacity offered by BRAM memories. Second, flip-flops are usually employed to store partial, i.e., temporary, results within a computational stage, thus minimally affecting the memory space requirements.

$$\begin{aligned} M_{dec} &= M_{calcUpc} + M_{calcBf} \\ &= \Big(M_H + par \times (M_s + M_{upc})\Big) \\ &\quad + \Big(M_H + par \times (M_e^{bf} + M_s^{bf}) + M_e\Big) \end{aligned} \quad (12)$$

According to Equation (12), the `calcUpc` stage requires to store the H matrix (M_H) the syndrome (M_s), and the computed UPCs (M_{upc}). We note that the term (M_s+M_{upc}) defines the memory requirement of a single dual-memory component within the `calcUpc` stage. In particular, the architectural parameter *par* is the multiplier to the cumulative memory requirement, that accounts for the possibility of implementing a parallel set of dual-memory blocks to compute the UPCs in the `calcUpc` stage.

In a similar manner, the `calcBf` stage requires to store the H matrix (M_H), the error bit-flips (M_e^{bf}), the syndrome bit-flips (M_s^{bf}) and the error vector (M_e). In particular, the term (M_e^{bf} + M_s^{bf}) defines the memory requirement of a single dual-memory component within the `calcBf` stage. As for the `calcUpc` stage, the architectural parameter *par* represents the multiplier to the cumulative memory requirement, that accounts for the possibility of implementing a parallel set of dual-memory blocks to compute the bit-flips in the `calcBf` stage.

Given the bandwidth ($BRAM_{bw}$) and the size in bits ($BRAM_{size}$) of a single FPGA BRAM memory, Equations (13)- (16) define the detailed memory requirements to store each of these matrices and vectors. In particular, Equation (13) defines the number of BRAM memories required to store the syndrome (M_s), error bit-flips (M_e^{bf}) and syndrome bit-flips (M_s^{bf}) vectors. We note that all of them share the same size of p bits.

$$M_s = M_e^{bf} = M_s^{bf} = \left\lceil \frac{p}{BRAM_{size}} \right\rceil \times \left\lceil \frac{bw}{BRAM_{bw}} \right\rceil \quad (13)$$

In a similar manner, Equation (14) defines the number of BRAM memories necessary to store the error vector (M_e).

$$M_e = n_0 \times \left\lceil \frac{p}{BRAM_{size}} \right\rceil \times \left\lceil \frac{bw}{BRAM_{bw}} \right\rceil \quad (14)$$

Last, the number of BRAMs required to store the UPCs (M_{upc}) and the positions of the ones in the H matrix (M_H) are defined by Equation (15) and Equation (16), respectively.

$$M_{upc} = \left\lceil \frac{p \times log(v)}{BRAM_{size}} \right\rceil \times \left\lceil \frac{bw}{BRAM_{bw}} \right\rceil \quad (15)$$

$$M_H = n_0 \times \left\lceil \frac{v \times log(p)}{BRAM_{size}} \right\rceil \times \left\lceil \frac{bw}{BRAM_{bw}} \right\rceil \quad (16)$$

We note that the term $\left\lceil \frac{bw}{BRAM_{bw}} \right\rceil$ is common to Equations (13)-(16), and it defines the integer number of BRAM memories as a function of the bandwidth parameter (bw). In particular, a bw value exceeding the available BRAM bandwidth imposes an integer increase in the number of BRAMs, regardless of the actual occupation in bits of the stored element. Given the code parameters, the space complexity highlights that the actual memory requirement to implement the decoder is a function of the two configurable architectural parameters, i.e., *par* and *bw*, that allow to regulate the resource-performance trade-off.

Considering Equation (13) and Equation (14), the term $\left\lceil \frac{p}{BRAM_{size}} \right\rceil$ defines the number of BRAM elements as a function of the size of p with respect to the storage capacity of a single BRAM, i.e., $BRAM_{size}$. The additional n_0 multiplier in Equation (14) highlights that the size of the error vector is n_0 times bigger than the syndrome. Considering Equation (15), the term $p \times log(v)$ accounts for the need to store p UPCs, each of which is the sum of v syndrome bits. In a similar manner, the term $v \times log(p)$ in Equation (16) accounts for the need to store the v positions of the ones for a block of the H matrix, where each of the v positions requires $log(p)$ bits.

IV. EXPERIMENTAL EVALUATION

This section discusses the characteristics of the proposed hardware decoder in terms of area and performance (execution time) with the final goal of highlighting the efficiency, flexibility and scalability of our solution. We note that we are not proposing a novel post-quantum QC-LDPC cryptosystem. In contrast, the proposed design methodology is meant to deliver an efficient and flexible hardware implementation to support any QC-LDPC cryptosystem that employs a bit-flipping decoding procedure. To the end of demonstrating the value of the proposed QC-LDPC bit-flipping (BF) decoding architecture, we employed the LEDAcrypt IND-CPA key encapsulation module (KEM) as our representative use case. In particular, the proposed decoder has been implemented on all the FPGAs of the mid-range Xilinx Artix-7 family. However, for the sake of readability and without loss of generality, results are reported only for the smallest and the largest FPGAs in the family, i.e., Artix-7 12 and Artix-7 200. We note that the Xilinx Artix-7 family offers the best price/performance and it has been suggested by NIST as the reference FPGA family for its post-quantum cryptography competition. Performance results are compared with two state-of-the-art software implementations running on an Intel i7 processor.

The rest of this section is organized in three parts. Section IV-A overviews the LEDAcrypt cryptosystem, with emphasis on the characteristics of the underlying code. Section IV-B details the experimental settings, encompassing both hardware and software. Finally, the experimental results, in terms of both resource utilization and performance, are discussed in Section IV-C.

A. LEDAcrypt CRYPTOSYSTEM

We consider the IND-CPA Key Encapsulation Module (KEM-CPA) from the LEDAcrypt post-quantum cryptography suite [27] as a representative use case for our QC-LDPC

BF decoder. The LEDAcrypt KEM-CPA is a code-based cryptosystem that relies on the Niederreiter cryptoscheme and employs a QC-LDPC code. LEDAcrypt is one of the finalists in the National Institute of Standards and Technology (NIST) initiative for the standardization of quantum-resistant public-key cryptosystems. This part overviews the LEDAcrypt configurations with the goal of demonstrating the wide applicability of our decoding architecture. The interested reader can find all the information related to the characteristics of the code in the publicly available documentation of the cryptosystem [8].

The code underlying the KEM-CPA has a code word length that is $p \cdot n_0$ and an information word length of $p \cdot (n_0 - 1)$, where $n_0 \in \{2, 3, 4\}$ and p is a large prime number. The LEDAcrypt KEM-CPA provides three security levels (equivalent respectively to AES-128, AES-192 and AES-256) and for each security level three different code rates, i.e., three different values of n_0. The block size p, the weight v of each H_i block and the number of decoding iterations $iter_{max}$, are reported in Table 2 for each LEDAcrypt-KEM-CPA configuration. Considering the decoding stage, the LEDAcrypt KEM-CPA cryptosystem employs the standard bit-flipping decoder, for which the pseudocode has been reported in Algorithm 1. We also note that the BIKE cryptosystem, i.e., another QC-LDPC-based cryptosystem that accessed the final stage of the NIST contest, employs the same decoding scheme with a marginal change that solely affects the first decoding iteration [9]. The experimental results detailed in this section have been obtained for code parameters corresponding to each of the nine configurations (see Table 2) of the LEDAcrypt IND-CPA KEM [8].

TABLE 2. Code parameters of the LEDAcrypt-KEM-CPA configurations [8].

Configuration	SL	n_0	p	v	$iter_{max}$
C1	AES-128	2	10883	71	6
C2		3	8237	79	5
C3		4	7187	83	4
C4	AES-192	2	21011	103	6
C5		3	15373	117	5
C6		4	13109	123	4
C7	AES-256	2	35339	137	4
C8		3	25603	155	4
C9		4	21611	163	4

B. EXPERIMENTAL SETUP

1) HARDWARE SETUP

The architecture for QC-LDPC bit-flipping decoding discussed in Section III has been described in SystemVerilog and it has been implemented using the Xilinx Vivado 2018.2 hardware design suite. The experimental evaluation has been carried out on two FPGAs from the mid-range Xilinx Artix-7 family. In particular, the Artix-7 12 (*xc7a12tcsg325-1*) and the Artix-7 200 (*xc7a200tsbg484-1*) FPGAs are respectively the lowest and the highest end of the Xilinx Artix-7 family (see the details of their available resources in Table 3).

TABLE 3. Available resources on the Artix-7 12 and Artix-7 200 FPGAs.

	Artix-7 12	Artix-7 200
LUT	8000	134600
FF	16000	269200
BRAM	20	365

Each design has been implemented considering a 100 MHz operating frequency, i.e., a 10 ns clock period. It is worth noticing that for each considered FPGA we only reported the best decoder configuration, i.e., the feasible one providing the best performance in terms of the time to compute the entire decoding. Such configurations have been identified after an extensive design space exploration considering all the combinations of values for the configurable parameters of our design, across a wide range of variability. In particular, we explored four bandwidths (`BW`), i.e., 32, 64, 128 and 256 bits, while the parallelism in the UPC (`Stage1`) and syndrome update computation (`Stage2`) has been explored considering a large set of values comprised between 1 and 32, i.e., 1, 2, 4, 8, 16, 24 and 32 bits.

2) SOFTWARE SETUP

We used two reference implementations for QC-LDPC BF decoding, extracted from the official implementation of the LEDAcrypt KEM-CPA [8]. First, the `C11` implementation is used as the reference design for performance evaluation. Second, the optimized software implementation employing the Intel `AVX2` extension has been considered as the top-notch software implementation from the performance point of view.

To ensure a fair performance assessment, the `C11` and the `Intel AVX2` software implementations of the QC-LDPC BF decoder have been adapted to execute $iter_{max}$ iterations. In the same way, early-termination is not available on our decoding architecture, i.e., the number of decoding iterations is fixed to $iter_{max}$.

The experimental evaluation of the software-implemented decoding has been carried out on an Intel Core i7-6700HQ processor, forcing a fixed operating frequency of 3.5 GHz to avoid performance variability due to the power management controller. For each LEDAcrypt-KEM-CPA configuration (see Ci with $i \in [1, ..9]$ in Table 2), the execution time of the decoding procedure for the two considered software implementations, i.e., `C11` and `AVX2`, was obtained by averaging the results of 30 executions.

3) FUNCTIONAL VALIDATION

The functional validation is meant to check the correctness of the decoding results obtained from the hardware implementation of our decoder template architecture. As the golden reference for our functional validation, we employed the QC-LDPC BF decoding procedure extracted from the publicly available LEDAcrypt-KEM-CPA, i.e., the official `C11` software implementation of the entire cryptosystem. In particular, for each configuration defined in the LEDAcrypt-KEM-CPA cryptosystem, i.e. the 9 configurations reported in Table 2, we collected the results of the software execution of 10000 different decoding procedures. The same decodings have been computed through both the post-implementation (timing) simulation and the board prototype execution of different implementations of our decoder template architecture. We note that the hardware implementation takes in input the H matrix and the initial syndrome computed by the software while, at the end of the decoding procedure, the error vector and the value of the boolean flag used to signal the possible decoding failure are compared to the values computed by the software implementation. For the post-implementation (timing) simulation, we implemented our decoder targeting the Xilinx Artix-7 12 (*xc7a12tcsg325-1*) and Artix-7 200 (*xc7a200tsbg484-1*) FPGAs. For the board prototype execution, we implemented our decoder targeting the Digilent Nexys 4 DDR FPGA board, which features a Xilinx Artix-7 100 (*xc7a100tcsg324-1*) FPGA. In particular, we implemented a performance-optimized instance of our decoder for each combination of the nine LEDAcrypt-KEM-CPA configurations and the three considered FPGAs, i.e., Xilinx Artix-7 12, Artix-7 100 and Artix-7 200. Each one of the 27 implemented decoders executed the 10000 decodings and the output results were compared with the output of the corresponding software-executed decoding.

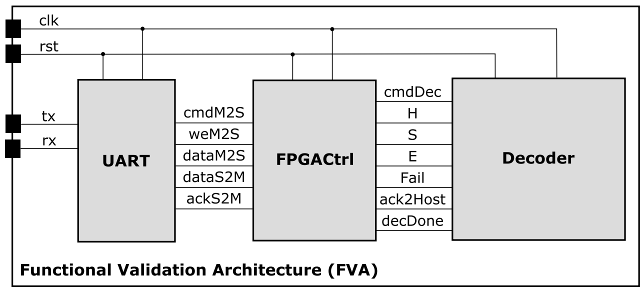


FIGURE 5. Hardware setup for the functional assessment of the proposed QC-LDPC BF decoder. The hardware setup is made of three parts. The UART module allows the communication between the host computer and the d `Functional Validation Architecture`. The FPGA controller (`FPGACtrl`) coordinates the communication with the host computer and the hardware execution of the decoding procedure. The decoder (`Decoder`) implements the version of the proposed QC-LDPC BF decoder architecture optimized for the underlying FPGA and LEDAcrypt-KEM-CPA configuration.

A functional validation architecture (`FVA`) is employed to provide a unified validation infrastructure for both the post-implementation (timing) simulations and the board prototype executions (see Figure 5). The `FVA` is made of three parts. The FPGA controller (`FPGACtrl`) communicates with the host computer to collect the input H matrix (`H`) and syndrome (`S`) and returns the error vector (`e`) and the failure flag (`Fail`). The UART module (`UART`) creates a simple yet effective communication channel between the FPGA controller and the host computer. The `Decoder` block represents the performance-optimized implementation of our QC-LDPC BF decoder architecture, that is specifically tailored for each combination of FPGA and LEDAcrypt-KEM-CPA configuration.

To perform a decoding, the `FPGACtrl` module drives the `cmdM2S` and the `weM2S` signals to collect the H matrix and the initial syndrome from the `UART` interface. We implemented a blocking communication protocol between the FPGA controller and the UART, thus the FPGA controller waits until the UART has sent the required data before closing the communication. Once the inputs have been collected, the `cmdDec` signal is used to load the operands into the decoder and to start the decoding. For each clock cycle, BW bits of the H matrix and the syndrome are passed to the decoder through the `H` and `S` signals. The decoder reports the end of the decoding through the `decDone` signal while, for each clock cycle, BW bits of the decoding result, i.e., the error vector, are loaded into the FPGA controller through the `e` signal. The `cmdDec` and the `ack2Host` signals are used to implement the acknowledged protocol to feed the inputs and to retrieve the output from the decoder. Last, the FPGA controller sends the decoding result to the host computer through the UART by means of the `dataM2S` data signal. We note that the `cmdM2S` and the `ack2Host` signals are used to implement the acknowledged protocol to exchange the decoding operands and the result between the FPGA controller and the UART module.

C. AREA AND PERFORMANCE RESULTS

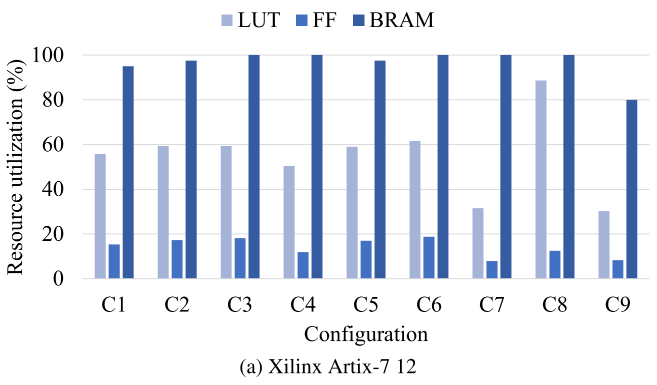
This section discusses the area and the performance of the proposed decoder, to demonstrate its efficiency and scalability across the entire family of mid-range Xilinx Artix-7 FPGAs.

1) AREA RESULTS

The proposed decoder makes use of the BRAMs of the FPGA as the primary means of storage for the inputs, the intermediate values and the result, allowing the decoder to fit on tiny FPGAs even for codes with a large block size p. In such a way, the maximum allowed dimension of the dense vectors that store the syndrome, the error and the UPCs is not a function of the available amount of flip-flops, that easily become the scarcest resources on small FPGAs, but it is instead a function of the available BRAM storage capacity. We note that a single BRAM can store up to 36kb and the smallest considered FPGA features 20 BRAMs.

For each configuration of the LEDAcrypt cryptosystem, considering the Xilinx Artix-7 12 and Artix-7 200 FPGAs, Figure 6 reports the normalized resource utilization of the LUT, flip-flop and BRAM elements, as a percentage on the total available.

As expected, the use of BRAM resources dominates each design on both the Xilinx Artix-7 12 and Artix-7 200 thus minimizing the use of flip-flops, which are therefore never the scarcest resource or a major showstopper. We note that even if the flip-flop utilization is low, the unused flip-flop resources can not be exploited to further improve the design. For example, on average the FF utilization on the Xilinx Artix-7 12 is



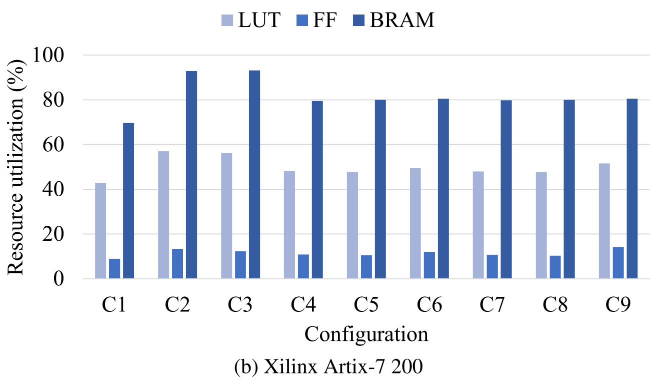


FIGURE 6. **Resource utilization of the proposed QC-LDPC BF decoder implemented on the Xilinx Artix-7 12 and Artix-7 200 FPGAs. Look-Up Table (LUT), Flip-Flop (FF) and Block-RAM (BRAM) resource types are considered. The utilization for each resource type is expressed as a percentage of the available resources on the target FPGA.**

below 15%, while the BRAM utilization is above 95% (see Figure 6a). However, the entire Xilinx Artix-7 12 features 16,000 FFs, thus their contribution is lower that the storage capacity of a single BRAM. In a similar manner, the FF utilization on the Xilinx Artix-7 200 is lower than 15% for each LEDAcrypt configuration. Even in such scenario, it is impossible to improve the design by leveraging on the FF resources. Instead, the average BRAM resource utilization is 82%, thus the storage capacity is never the bottleneck of the implemented designs. In contrast, the limiting factor to a better resource utilization is the timing, which becomes the bottleneck on the decoder implementations on the Xilinx Artix-7 200, due to the massive parallelism that is achieved thanks to the vast amount of available resources. However, a complete performance discussion is left to the performance evaluation part in the following of this section.

Considering the Look-Up Tables (LUTs), we note an average utilization of 55% and 50% on the Xilinx Artix-7 12 and Xilinx Artix-7 200, respectively. Despite such resource type never becomes the scarcest one across the entire set of considered decoder implementations, its utilization varies depending on the actual level of parallelism of each implemented decoder, since LUTs are used to implement the combinational logic of the decoder. Table 4 reports the level of parallelism for the two design-time knobs of our decoding architecture for each combination of FPGA and LEDAcrypt configuration. The 32-bit bandwidth (`BW`) is found to be the optimal value to implement each LEDAcrypt configuration on the Xilinx Artix-7 12 FPGA, while the optimal level of parallelism (`PAR`) to maximize the computation of the two vector-matrix multiplications in the bit-flipping procedure, i.e., $UPC = H \cdot s$ and $s = H \cdot e$, ranges between 1 and 4, thus determining a variability in the used LUTs depending on the implemented LEDAcrypt configurations. For example, LUT utilization is around 60% for configurations in the range $C1 - C6$, while it drops down to 30% for $C7$ and $C9$ and it peaks to almost 90% for $C8$ (see Figure 6a). We note that `PAR` is equal to 4 for $C1 - C6$ configurations, thus determining a

TABLE 4. **Configuration parameters for the hardware instances on the Artix-7 12 and 200 FPGAs.**

Configuration	Artix-7 12		Artix-7 200	
	BW	PAR	BW	PAR
C1	32	4	128	24
C2	32	4	128	32
C3	32	4	128	32
C4	32	2	128	24
C5	32	4	128	24
C6	32	4	128	24
C7	32	1	128	24
C8	32	2	128	24
C9	32	1	128	24

higher use of LUTs with respect to $C7$ and $C9$ configurations, since even if the latter targets larger QC-LDPC codes, they are implemented with $PAR = 1$. Similarly, the large use of LUTs for $C8$ is motivated by both the larger QC-LDPC code compared to the one of $C1 - C6$ and the possibility of using $PAR = 2$ without exceeding the available hardware resources of the FPGA. Considering the implemented decoders on the Xilinx Artix-7 200, the average LUT utilization is 50% with small variations between different configurations (see Figure 6b). The 128-bit bandwidth has been found optimal for the entire set of LEDAcrypt configurations, while $PAR = 24$ is employed for all the configurations but $C2$ and $C3$, for which 32 is used instead (see Table 4). Such increase in the parallelism for $C2$ and $C3$ impacts the used LUTs, for which a value slightly below 60% is reported (see LUT for $C2$ and $C3$ in Figure 6b). We note one more time that the impossibility of resorting to either a more aggressive level of parallelism or a larger bandwidth for the decoders implemented on the Artix-7 200 FPGA, is due to the imposed timing constraints equal to 10ns and not to the available resources on the board. However, as it is explained in the following, such decoder implementations allow to overcome the performance of optimized software-implemented decoders employing the Intel AVX2 extension by 5 times, on average.

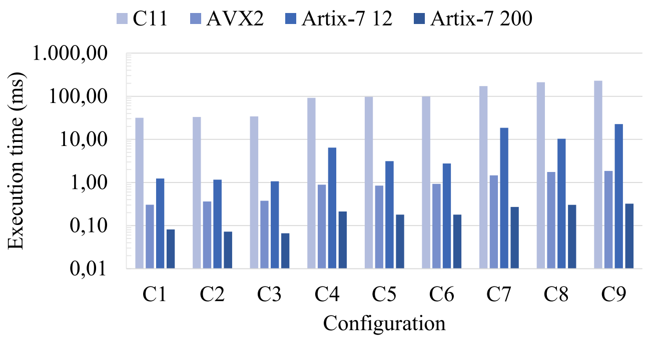


FIGURE 7. **Execution time (in milliseconds) of QC-LDPC BF decoding. Results are shown for software decoding on the Intel i7 processor and for hardware decoding on the Artix-7 12 and Artix-7 200 FPGAs.**

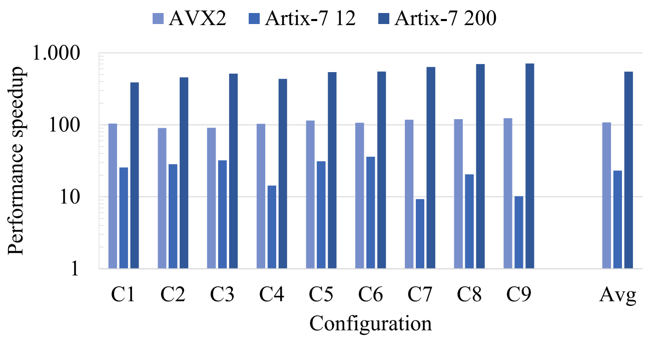


FIGURE 8. **Performance improvement with respect to C11 software decoding executed on the Intel i7 processor. Results are shown for AVX2 software decoding on the Intel i7 processor and for hardware decoding on the Artix-7 12 and Artix-7 200 FPGAs.**

2) PERFORMANCE RESULTS

Figure 7 reports the performance results expressed as the execution time to complete the bit-flipping decoding procedure for all the LEDAcrypt configurations. The results are reported for each configuration considering the two software implementations, i.e., `C11` and `AVX2`, and the two hardware implementations, which targets the Xilinx Artix-7 12 and Artix-7 200 FPGAs, respectively. As expected, the execution time increases with the size and the weight of the QC-LDPC code for all the implementations. For example, `C11` takes 32 ms and 229 ms to complete the decoding of $C1$ and $C9$, respectively (see Figure 7). In order to highlight the actual performance speedup across the different implementations of the decoding procedure, Figure 8 reports the performance speedup of the AVX2 software and of the two hardware implementations, normalized with respect to the *C11* software version. The decoders targeting the low-end Xilinx Artix-7 12 FPGA show an execution time comprised between 1 and 25 milliseconds, with a corresponding performance improvement between 9 and 36 times (23 times on average) with respect to the `C11` software implementation. We note that the optimized software implementation employing the Intel AVX2 extension (`AVX2`) shows an average performance speedup of 108 times compared to the *C11* reference software version. However, our decoders targeting the Xilinx Artix-7 200 FPGA show an average 5 times speedup against the performance-optimized software employing the Intel AVX2 extension (see Figure 8). Such results demonstrate the superior performance and scalability of our decoding architecture against optimized software solutions exploiting custom and hardware-accelerated instructions offered by recent high-end Intel processors.

V. CONCLUSION

This work presented an efficient and scalable architecture of QC-LDPC bit-flipping decoders for post-quantum cryptography. The design efficiency is achieved through an optimized architecture computing the time consuming vector-matrix multiplications that are at the core of the bit-flipping decoding procedure. The design scalability is achieved by exploiting both a configurable bandwidth in the decoder datapath and via the massive use of the available FPGA block RAMs (BRAMs) to store the inputs as well as the partial and the final results, without affecting the flip-flops that are typically the scarcest hardware resource of any design.

To demonstrate the effectiveness of our solution, we considered the nine configurations of the LEDAcrypt cryptosystem as representative use cases for QC-LDPC codes suitable for post-quantum cryptography. For each configuration, our template architecture can deliver a performance-optimized decoder implementation for each FPGA of the Xilinx Artix-7 mid-range family. The experimental results demonstrate that our optimized architecture allows the implementation of large QC-LDPC codes even on the smallest FPGA of the Xilinx Artix-7 family. Considering the implementation of our decoder on the Xilinx Artix-7 200 FPGA, the experimental results show an average performance speedup of 5 times across the full range of the LEDAcrypt configurations, compared to the official optimized software implementation of the decoder employing the Intel AVX2 extension.

REFERENCES

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

[2] N. Sendrier, "Code-based cryptography: State of the art and perspectives," *IEEE Secur. Privacy*, vol. 15, no. 4, pp. 44–50, 2017.

[3] E. Berlekamp, R. McEliece, and H. van Tilborg, "On the inherent intractability of certain coding problems (Corresp.)," *IEEE Trans. Inf. Theory*, vol. 24, no. 3, pp. 384–386, May 1978.

[4] D. J. Bernstein, "Grover vs. mceliece," in *Proc. Int. Workshop Post-Quantum Cryptogr.* Berlin, Germany: Springer, 2010, pp. 73–80.

[5] R. J. McEliece, "A public-key cryptosystem based on algebraic coding theory," Commun. Syst. Res. Sect., NASA, Washington, DC, USA, DSN Prog. Rep. DSN PR 42-44, Jan. 1978, pp. 114–116.

[6] M. Baldi, M. Bodrato, and F. Chiaraluce, "A new analysis of the McEliece cryptosystem based on QC-LDPC codes," in *Proc. Int. Conf. Secur. Cryptogr. Netw.*, Amalfi, Italy, Sep. 2008, pp. 246–262.

[7] R. Misoczki, J.-P. Tillich, N. Sendrier, and P. S. L. M. Barreto, "MDPC-McEliece: New McEliece variants from moderate density parity-check codes," in *Proc. IEEE Int. Symp. Inf. Theory*, Istanbul, Turkey, Jul. 2013, pp. 2069–2073.

[8] M. Baldi, A. Barenghi, F. Chiaraluce, G. Pelosi, and P. Santini. *LEDAcrypt Website*. Accessed: Sep. 3, 2020. [Online]. Available: https://www.ledacrypt.org/

[9] N. Aragon, P. S. L. M. 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, V. Vasseur, and G. Zémor. *BIKE Website*. Accessed: Sep. 3, 2020. [Online]. Available: https://www.bikesuite.org/

[10] D. Zoni, A. Galimberti, and W. Fornaciari, "Flexible and scalable FPGA-oriented design of multipliers for large binary polynomials," *IEEE Access*, vol. 8, pp. 75809–75821, 2020.

[11] M. Baldi, *QC-LDPC Code-Based Cryptography*. Cham, Switzerland: Springer, 2014.

[12] M. Ismail, I. Ahmed, J. Coon, S. Armour, T. Kocak, and J. McGeehan, "Low latency low power bit flipping algorithms for LDPC decoding," in *Proc. 21st Annu. IEEE Int. Symp. Pers., Indoor Mobile Radio Commun.*, Sep. 2010, pp. 278–282.

[13] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inf. Theory*, vol. 8, no. 1, pp. 21–28, Jan. 1962.

[14] Z. Zhang, V. Anantharam, M. J. Wainwright, and B. Nikolic, "An efficient 10GBASE-T Ethernet LDPC decoder design with low error floors," *IEEE J. Solid-State Circuits*, vol. 45, no. 4, pp. 843–855, Apr. 2010.

[15] S. H. Gupta and B. Virmani, "LDPC for Wi-Fi and WiMAX technologies," in *Proc. Int. Conf. Emerg. Trends Electron. Photonic Devices Syst.*, Dec. 2009, pp. 262–265.

[16] N. Jiang, K. Peng, J. Song, C. Pan, and Z. Yang, "High-throughput QC-LDPC decoders," *IEEE Trans. Broadcast.*, vol. 55, no. 2, pp. 251–259, Jun. 2009.

[17] G. Wang, M. Wu, Y. Sun, and J. R. Cavallaro, "A massively parallel implementation of QC-LDPC decoder on GPU," in *Proc. IEEE 9th Symp. Appl. Specific Processors (SASP)*, Jun. 2011, pp. 82–85.

[18] I. Tsatsaragkos and V. Paliouras, "A reconfigurable LDPC decoder optimized for 802.11n/AC applications," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 1, pp. 182–195, Jan. 2018.

[19] M. Zhao, X. Zhang, L. Zhao, and C. Lee, "Design of a high-throughput QC-LDPC decoder with TDMP scheduling," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 62, no. 1, pp. 56–60, Jan. 2015.

[20] Y. Kou, S. Lin, and M. P. C. Fossorier, "Low-density parity-check codes based on finite geometries: A rediscovery and new results," *IEEE Trans. Inf. Theory*, vol. 47, no. 7, pp. 2711–2736, 2001.

[21] J. Zhang and M. P. C. Fossorier, "A modified weighted bit-flipping decoding of low-density parity-check codes," *IEEE Commun. Lett.*, vol. 8, no. 3, pp. 165–167, Mar. 2004.

[22] T. Wadayama, K. Nakamura, M. Yagita, Y. Funahashi, S. Usami, and I. Takumi, "Gradient descent bit flipping algorithms for decoding LDPC codes," in *Proc. Int. Symp. Inf. Theory Its Appl.*, Dec. 2008, pp. 1–6.

[23] J. Hu, M. Baldi, P. Santini, N. Zeng, S. Ling, and H. Wang, "Lightweight key encapsulation using LDPC codes on FPGAs," *IEEE Trans. Comput.*, vol. 69, no. 3, pp. 327–341, Mar. 2020.

[24] D. Apon, R. Perlner, A. Robinson, and P. Santini, "Cryptanalysis of ledacrypt," Cryptol. ePrint Arch., Springer, Cham, Switzerland, Tech. Rep. 2020/455, 2020. [Online]. Available: https://eprint.iacr.org/2020/455

[25] I. V. Maurich and T. Guneysu, "Lightweight code-based cryptography: QC-MDPC McEliece encryption on reconfigurable devices," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2014, pp. 1–6.

[26] N. Drucker, S. Gueron, and D. Kostic, "Qc-mdpc decoders with several shades of gray," in *Post-Quantum Cryptography*, J. Ding and J.-P. Tillich, Eds. Cham, Switzerland: Springer, 2020, pp. 35–50.

[27] M. Baldi, A. Barenghi, F. Chiaraluce, G. Pelosi, and P. Santini, "LEDAcrypt: QC-LDPC code-based cryptosystems with bounded decryption failure rate," in *Code-Based Cryptography*, M. Baldi, E. Persichetti, and P. Santini, Eds. Cham, Switzerland: Springer, 2019, pp. 11–43.

DAVIDE ZONI received the M.Sc. and Ph.D. degrees from the Politecnico di Milano, Italy. He is currently a Postdoctoral Researcher with the Politecnico di Milano. He has published more than 40 papers in journals and conference proceedings. He filed two patents on cybersecurity. He is also the Principal Investigator of the LAMP proof-of-concept project that aims at delivering a side-channel resistant RISC-based system-on-chip. His research interests include RTL design and optimization for multi-cores with particular emphasis on low power methodologies and hardware-level countermeasures to side-channel attacks. He received two HiPEAC collaboration grants in 2013 and 2014, two HiPEAC industrial grant in 2015 and 2017, and the Switch2Product Competition in 2019.

ANDREA GALIMBERTI received the M.Sc. degree in computer science and engineering from the Politecnico di Milano, Italy, in 2019, where he is currently pursuing the Ph.D. degree. His research interests include computer architectures, hardware-level countermeasures to side-channel attacks, and design of hardware accelerators.

WILLIAM FORNACIARI (Senior Member, IEEE) received the M.Sc. and Ph.D. degrees. He is currently an Associate Professor with the Politecnico di Milano, Italy. He has published six books and around 300 papers in international journals and conferences, collecting six best paper awards, and one certification of appreciation from IEEE. He holds three international patents on low-power design. He has been a coordinator of both FP7 and H2020 EU-projects. In 2016, he received the HiPEAC Technology Transfer Award. His research interests include embedded and cyber-physical systems, energy-aware design of SW and HW, run-time management of resources, high-performance computing, design optimization, and thermal management of multi-many cores.