Efficient Architectures for Multigigabit CCSDS LDPC Encoders

Dimitris Theodoropoulos, *Student Member, IEEE*, Nektarios Kranitis, *Member, IEEE*, Antonis Tsigkanos, *Member, IEEE*, and Antonios Paschalis, *Member, IEEE*

***Abstract*— Quasi-cyclic low-density parity-check (QC-LDPC) codes have been adopted by the Consultative Committee for Space Data Systems (CCSDS) as the recommended standard for onboard channel coding in Near-Earth and Deep-Space communications. Encoder architectures proposed so far are not efficient for high-throughput hardware implementations targeting the specific CCSDS codes. In this article, we introduce a novel architecture for the multiplication of a dense quasi-cyclic (QC) matrix with a bit vector, which is the fundamental operation of QC-LDPC encoding. The architecture leverages the inherent parallelism of the QC structure by concurrently processing multiple bits, according to an optimized scheduling. Based on this architecture, we propose efficient encoders for CCSDS codes, according to all the applicable low-density parity-check (LDPC) code encoding methods. Moreover, in the special case of the code for Near-Earth communications, we also introduce a preprocessing algorithm to efficiently handle the challenges arising from the generator's matrix circulant size (511 bits). The proposed architectures have been implemented in various field-programmable gate array (FPGA) technologies and validated in Zynq UltraScale+ multiprocessor system-on-chip (MPSoC), achieving a significant speedup compared with previous approaches, while at the same time keeping resource utilization low.**

***Index Terms*— Consultative Committee for Space Data Systems (CCSDS), channel coding, field programmable gate arrays (FPGAs), low-density parity-check (LDPC) codes, multiprocessor system-on-chip (MPSoC), parity check codes.**

I. Introduction

LOW-DENSITY parity-check (LDPC) codes are linear block codes, characterized by large block lengths and sparse parity-check matrices. The initial Gallager codes [1] were random, and although they exhibited excellent error-correcting capabilities, hardware implementation was challenging. In order to reduce implementation complexity and enhance encoding/decoding speed, an additional structure has been designed into the parity check matrices of all practical LDPC codes in modern applications, so that they consist of an array of juxtaposed cyclic submatrices, named the circulants, which can be efficiently implemented. These structured codes are collectively referred to as quasi-cyclic (QC) LDPC codes. QC-low-density parity-check (QC-LDPC) codes have been adopted by many modern communication standards, such as IEEE 802.11, 802.16, and DVB-S2.

A special class of structured LDPC codes, the protograph-based QC codes have recently received considerable research interest [2] in many modern standards. Their ability to mitigate noise and intersymbol interference (ISI) degradation effects in magnetic recording (MR) channel is described in [3]. The protograph codes proposed by Fang *et al.* [4] exhibit an outstanding performance over the partial response (PR) channel, used to model MR systems. The work by Chen *et al.* [5] shows the application of LDPC codes to physical layer network coding (PNC) and describes the research on LDPC codes for PNC as an emerging research trend. A class of protograph-based LDPC codes for use on PNC is also proposed in [5]. Finally, a novel family of root-protograph QC-LDPC codes has recently been proposed in [6] for modern point-to-point and multirelay wireless communication applications, modeled according to the block (or slow) fading channel. The Consultative Committee for Space Data Systems (CCSDS) has standardized in [7] a number of protograph-based QC-LDPC code families for space communication protocols, as alternatives to concatenated convolutional and Reed–Solomon codes. The recommended LDPC codes outperform their predecessors in every aspect, including power, spectral efficiency, and bit error rate (BER) performance, especially in the error-floor region. In this article, we focus on these specific codes.

A multitude of encoder architectures for QC-LDPC codes has been proposed in the literature. Most of these architectures, however, focus on a specific standard or class of standards, leveraging the specific properties of the particular code. The result is that although they exhibit outstanding performance characteristics for the specific code family, they are either altogether nonapplicable to CCSDS or their adoption to CCSDS codes comes with a significant performance penalty. Most of these encoder architectures require a specific structure in the parity check matrix of the code, not satisfied by CCSDS codes. Examples of such cases are given in [8]–[11].

Previous works in [12]–[15] target specifically CCSDS codes. Our previous work [12] introduced a parallel architecture for the execution of the vector-matrix multiplication involved in LDPC encoding, by leveraging the inherent paral-

Manuscript received September 30, 2019; revised January 10, 2020; accepted February 13, 2020. Date of publication March 2, 2020; date of current version April 24, 2020. This work was supported in part by the Hellenic Foundation for Research and Innovation (HFRI) and in part by the General Secretariat for Research and Technology (GSRT) under the first call for HFRI Research Projects for the support of Postdoctoral Researchers under Grant 990. *(Corresponding author: Dimitris Theodoropoulos.)*

Dimitris Theodoropoulos, Antonis Tsigkanos, and Antonios Paschalis are with the Digital Systems and Computer Architecture Laboratory (DSCAL), Department of Informatics and Telecommunications, National and Kapodistrian University of Athens, 157 84 Athens, Greece (e-mail: theodimitris@di.uoa.gr; antts@di.uoa.gr; paschalis@di.uoa.gr).

Nektarios Kranitis is with the Department of Digital Systems, University of the Peloponnese, 231 00 Sparta, Greece (e-mail: nkran@di.uoa.gr).

Digital Object Identifier 10.1109/TVLSI.2020.2975050

lelism of the generator matrix. This article advances to more efficient encoding methods, resulting in significant performance speedups. The approach proposed in [13] involves wide XOR operations over a significant number of bits (2048 in the provided example) and requires significant register resources for the shift registers. The architecture proposed in [14] is bit-serial. A parallel implementation is described in Section IV for comparisons with the encoder architectures proposed in this article. Miles *et al.* [15] proposed a packing–unpacking preprocessing scheme for a specific class of CCSDS codes, followed by two parity generators, which in turn is based on shift registers with multiple shift values. It is shown in Section IV that their architecture requires significant combinatorial resources.

Other architectures have been proposed, which target LDPC code standards with compatible characteristics with CCSDS codes, or they are generic enough to be applicable to a broader range of LDPC code standards, including CCSDS. The work shown in [16] proposes various types of encoding circuits, based on shift registers, which achieve encoding complexity linearly proportional to the number of parity bits of the code, or the total bits of the code in the case of the parallel approach. The shift-register-adder-accumulator (SRAA) serial encoding scheme described is one of the approaches provided in the CCSDS standard [7], based on a shift register for the circulants and a register for the calculation of parity bits. Another parallel SRAA approach is also proposed in [16], which achieves encoding in *cb* cycles (following the authors' notation), all input bits participate in the calculation of each parity bit in one clock cycle. The efficiency of these architectures for CCSDS codes is limited in terms of achievable throughput, resource requirements, and critical path of a hardware implementation. Other implementations of the SRAA architecture are found in [17] and [18]. Note that [17] is optimized for sparse circulants. Finally, the work in [19] describes an implementation of the architectures introduced in [14].

Another class of the so-far-proposed encoder architectures designed for other standards, but are also applicable to CCSDS codes, is based on the Richardson–Urbanke (R–U) LDPC encoding method [20]. Although they are suitable for these standards, they are not expected to scale efficiently for CCSDS codes, which are characterized by large dense matrices. Examples of this case are [21] and [22]. Similarly, in the architecture proposed in [23] and [24], the authors leverage the SRAA modules introduced in [16] for the dense matrix operations involved in the R–U algorithm. The proposed method is affected by the same scalability issues concerning the SRAA architecture in [16].

Other authors propose a different encoding scheme, based on the triangular decomposition of the rightmost part of the parity check matrix of the code. Examples of such architectures can be found in [25]–[28]. The advantages of these algorithms are limited to random Gallager codes. However, the adoption of this encoding method for CCSDS inflicts a major performance penalty due to the loss of QC structure in the decomposed triangular matrices. The work in [29] extends the procedure introduced in [26] and is efficient for the selected codes (multilevel QC-LDPC codes). However, for CCSDS codes, the decomposed matrices are dense and random and the proposed architectures are not applicable. Another fully parallel method is also proposed in [29], targeting high throughput. This method involves the parallel multiplication of large dense vectors and cannot scale for higher block lengths or other codes: its application on CCSDS codes requires a large number of resources and introduces a large critical path. As part of this article, we created and simulated a hardware description of this fully parallel method for the CCSDS rate $1/2$ code with a block length of 2048 bits. Targeting Virtex 5 field programmable gate array (FPGA) technology, the synthesized encoder design required more than 72-K lookup tables (LUTs), fitting only high-end FPGA devices, with significant routing delays.

To sum up, the so-far-proposed architectures targeting different QC-LDPC codes of other communication standards are not efficient for the specific CCSDS codes and codes with similar characteristics. Moreover, even those focusing on CCSDS codes are not suitable for high-throughput hardware implementations, since they cannot handle efficiently the multiplication of a binary vector with dense QC matrices. This multiplication is the critical operation involved in any LDPC encoding method.

The contribution of this article is summarized in the following points.

1) A novel parallel algorithm is introduced for the efficient multiplication of a binary vector with a dense QC matrix.
2) New encoder architectures are proposed, leveraging the introduced algorithm, according to all applicable encoding methods. This is the first published implementation for CCSDS or similar codes to the best of our knowledge. These architectures address the limitations of previous works, pertaining to CCSDS codes.
3) The performance and efficiency of the different encoding methods are analytically estimated and assessed, specifically for CCSDS codes. This is the first time such a comparison is made.
4) A new processing algorithm is introduced, for the special case of rate 7/8 codes. This method handles the inconveniencies of the code structure efficiently.

The combined result of these contributions is that we were able to implement and verify on-chip, hardware encoders for CCSDS codes, with significantly improved performance compared to existing implementations. The remainder of this article is organized as follows: Section II provides the necessary background on CCSDS codes. Section III describes the introduced algorithm, which implements the dense QC matrix multiplications. In Section IV, encoder architectures are proposed, which leverage the above algorithm for CCSDS LDPC encoding with all the different encoding method variants. Hardware implementations are presented in Section V and compared with previous work. Section VI concludes this article.

II. CCSDS Codes

This article focuses on the LDPC codes implemented in the data link layer of CCSDS protocols, and more specifically the

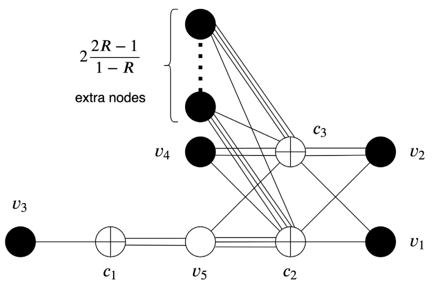


Fig. 1. AR4JA protograph. R is the rate of the code.

synchronization and channel-coding sublayer of the telemetry Space Data-Link Protocol (TM-SDLP), as they are defined in the CCSDS standard [7]. Two classes of LDPC codes were adopted for use in TM-SDLP: one optimized for Deep-Space and another for Near-Earth communications.

For Deep-Space communications, nine codes are defined, which belong to *Accumulate*, *Repeat by* 4, and *Jagged Accumulate (AR4JA)* class of LDPC codes. They are the result of the combination of three block length sizes (1-, 4-, and 16-K bits) over three code rates: 1/2, 2/3, and 4/5. Their parity check matrices consist of an array of circulant sparse $M \times M$ submatrices, where M is a parameter dependent on the block length and code rate. All these codes are constructed by uplifting the protograph depicted in Fig. 1. Nodes labeled u_j in the graph (variable nodes) denote input information block bits, or equivalently columns of the parity-check matrix, whereas those labeled c_i (check nodes) correspond to parity-check equations or rows of the parity-check matrix. If H is the parity-check matrix of the code, a line connects variable node j to check node i when $H(i, j) = 1$. Node v_5 corresponds to punctured bits in the codeword, which are not transmitted, so as to increase the rate of the code.

The parity-check matrix is built from the protograph according to the following procedure: first, the protograph is expanded by a factor of 4, and the parallel edges of the protograph are randomly spread between the expanded nodes. For the matrix H' which corresponds to the expanded graph, each element $H'(i, j)$ representing a single edge at the initial protograph is assigned the value $I^{(0)}$, whereas elements of the matrix corresponding to multiple parallel edges are assigned a random value $I^{(x)}$, where $x \in [0, m-1]$ and $m = M/4$. The intermediate matrix H' is subsequently expanded by a factor of m, by substituting each nonzero element of the intermediate matrix I_x by the $m \times m$ identity matrix I_m, rotated by x positions to the clockwise direction.

For Near-Earth communications, a basic LDPC code with dimensions (8176, 7154) is defined, constructed on the 3-D Euclidean geometry EG(3, 2^3) over the field GF(2^3). We refer to this code as C2 code. The QC parity check matrix consists of a 2×14 array of 511×511 sparse circulants. On the contrary, the generator matrix in a systematic circulant form is a composition of a 7154×7154 identity matrix and a 14×2 array of 511×511 dense circulants. The encoding process prepends 18 zero bits to the 7136 bits of the incoming frame to be encoded. These bits participate in the encoding process but they are not transmitted as part of the systematic output of the encoder. Two tailing bits are also added to the final codeword to ensure that the output codeword length is divisible by 8 and 16. With the addition of these tailing bits, the dimensions of the recommended code are finally (8160, 7136).

The standard [7] provides the option for randomization of the output codeword to ensure sufficient transition density on the transmitted vector. The encoder's output consists of the optionally randomized codeword, prepended by a 64-bit (AR4JA) or 32-bit (C2) synchronization sequence.

III. Proposed Algorithm

In the general case, consider the multiplication $p = sW$ of the bit vector s with the dense QC matrix W, which is an $r \times c$ array of $m \times m$ circulants. Vectors s and p are also partitioned into a number of r and c subvectors correspondingly, each consisting of m bits, as shown in (1) and (2). Bit j of subvector p_i is calculated by (3), where $W_{l,i}(j)$ is the jth column of the circulant $W_{l,i}$

$$s = \begin{bmatrix} s_1 & s_2 & \ldots & s_r \end{bmatrix} \tag{1}$$
$$p = \begin{bmatrix} p_1 & p_2 & \ldots & p_c \end{bmatrix} \tag{2}$$
$$p_i(j) = \sum_{l=1}^{r} s_l W_{l,i}(j) \quad (1 \le i \le c, 1 \le j \le m). \tag{3}$$

In a parallel implementation, a number of L input bits of vector s can be concurrently processed at each clock cycle. If parameter L is such that m is an integral multiple of L, (3) can be completed in mr/L cycles according to the following method.

We define the subvectors of s_l being processed in the current clock cycle as in (4). We also split the column vector $W_{l,i}(j)$ of (3) into m/L subvectors of L bits. Subvector $W_{l,i}^{(e)L}(j)$ is defined in (5), where $0 \le e \le \frac{m}{L} - 1$. In practice, since L bits of input vector p are processed, index e refers to one of the m/L execution cycles in which circulant $W_{l,i}$ is involved

$$s_l^{(e)L} = \begin{bmatrix} s_l(eL+1) \; s_l(eL+2) \; \ldots \; s_l(eL+L) \end{bmatrix} \tag{4}$$
$$W_{l,i}^{(e)L}(j) = [W_{l,i}(eL+1, j) \; W_{l,i}(eL+2, j) \; \ldots \; W_{l,i}(eL+L, j)]^T. \tag{5}$$

Because submatrices $W_{i,j}$ are cyclic, (6) holds. To simplify notation, we define $W_{l,i}^{L}(j) = W_{l,i}^{(0)L}(j)$. Symbol $\overset{m}{\oplus}$ signifies modulo-m addition

$$W_{l,i}^{(e+\upsilon)L}(j) = W_{l,i}^{(e)L}\left(j \overset{m}{\oplus} \upsilon L\right) \quad 0 \le \upsilon \le \frac{m}{L} - 1. \tag{6}$$

At each clock cycle, for each bit position j of subvector p_i, we calculate a partial result $\varrho_i^{(e)}(j, l)$ in parallel, according to (7). Note that $W_{l,i}^{L}(j)$ is independent of the execution cycle. This means that each $\varrho_i^{(e)}(j, l)$ depends only on the input bits and the current circulant index l

$$\varrho_i^{(e)}(j, l) = s_l^{(e)L} W_{l,i}^{L}(j). \tag{7}$$

Each $p_i(j)$ can be calculated by accumulating the corresponding values of $\varrho_i^{(e)}(j, l)$, using shift registers. Equation (3) can be rewritten as in (8), if we take into account (6) and (7). The partial results which are calculated at each clock cycle e are accumulated into $p_i(j)$, as indicated by the internal summation in (8). The external summation updates the values of $W_{l,i}^{L}$ in (7) and the accumulation continues for all l

$$p_i(j) = \sum_{l=1}^{r} \left(\sum_{e=0}^{\frac{m}{L}-1} \varrho_i^{(e)}\left(j \overset{m}{\oplus} eL, l\right) \right) \quad \left\lfloor \frac{m}{L} \right\rfloor = 0. \tag{8}$$

In this article, we introduce an additional degree of parallelism. In every clock cycle (e), instead of processing L consecutive bits of s, we process L_m bits from each of the r subvectors s_i. We set $L_m = L/r$, so that the total number of bits being concurrently processed are fixed to L. According to this scheme, at each execution cycle, instead of calculating and accumulating $\varrho_i^{(e)}(j,l)$, we calculate the partial sum $\delta_i^{(e)}(j)$ according to (9). The partial results are accumulated into $p_i(j)$ according to (10). The total number of bits being concurrently processed is again equal to L, but at each clock cycle, all circulants participate in the calculation of parity bits

$$\delta_i^{(e)}(j) = \sum_{l=1}^{r} s_l^{(e)L_m} W_{l,i}^{L_m}(j), \quad 0 \le e \le \frac{m}{L_m} - 1 \tag{9}$$

$$p_i(j) = \sum_{e=0}^{\frac{m}{L_m}-1} \delta_i^{(e)}\left(j \overset{m}{\oplus} eL_m\right). \tag{10}$$

Note that in this case, $\delta_i^{(e)}(j)$ is independent of the current circulant, in contrast to $\varrho_i^{(e)}(j,l)$ in (7), where the dependence on l means that for the calculation of each $p_i(j)$, the corresponding $\varrho_i^{(e)}(j,l)$ is a logical function of $L + \log_2(r)$ bits. On the contrary, this means that each $W_{l,i}^{L}$ needs to be calculated from a function generator with $\log_2(r)$ inputs. According to the introduced method, however, the value of $\delta_i^{(e)}(j)$ depends only on the rL_m bits of the input vector s.

The decoupling of $\delta_i^{(e)}(j)$ from l comes with a significant advantage for hardware implementation. If we define the truth function $\mathbb{1}$ as in (11), and set $w(\xi)$ as the ξth element of column vector $W_{l,i}^{L_m}(j \overset{m}{\oplus} eL_m)$, then (9) can be simplified as (12). This simplified statement of partial sums removes the need of a function generator for the elements of W and significantly reduces complexity

$$\mathbb{1}[x=1] = \begin{cases} 1, & x = 1 \\ 0, & x \neq 1 \end{cases} \tag{11}$$

$$\delta_i^{(e)}(j) = \sum_{l=1}^{r} \left(\sum_{\xi=1}^{L_m} s_l^{(e)L_m}(\xi) \mathbb{1}[w(\xi) = 1] \right). \tag{12}$$

Consequently, for a hardware implementation of this algorithm, at each cycle we need to calculate vector $\delta^{(e)}$ according to (12) and accumulate this partial result into a register, according to the summation indicated in (10).

The accumulation of $\delta_i^{(e)}(j)$ in (10) can be efficiently implemented with a linear feedback shift register (LFSR), an abstract block diagram of which is presented in Fig. 2. The input feed is given in (12). After m/L_m processing cycles, registers $u_i(j)$ have accumulated the expected result, according to (10).

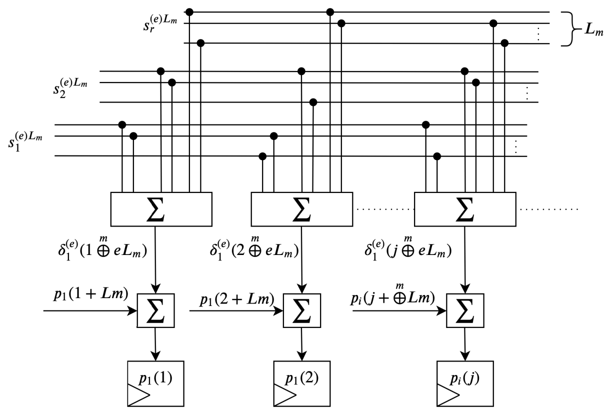

$s_r^{(e)L_m}$
L_m
$s_2^{(e)L_m}$
$s_1^{(e)L_m}$
Σ
$\delta_1^{(e)}(1 \overset{m}{\oplus} eL_m)$
$\delta_1^{(e)}(2 \overset{m}{\oplus} eL_m)$
$\delta_1^{(e)}(j \overset{m}{\oplus} eL_m)$
$p_1(1 + Lm)$
$p_1(2 + Lm)$
$p_i(j + \overset{m}{\oplus} Lm)$
$p_1(1)$
$p_1(2)$
$p_i(j)$

Fig. 2. LFSR architecture for QC vector-matrix multiplication.

TABLE I
RESOURCE AND PERFORMANCE ESTIMATIONS

2-input binary functions (2-input xor)	$cmrL_m/2$
Registers (FFs)	cm
Logic levels (2-input xor)	$\lceil log_2\,(rL_m/2) \rceil$
Encoding cycles	m/L_m

Table I lists the calculated resources and performance estimations. The input to each register of the LFSR is a function of the nonzero elements of column vector $W_{l,i}(j)$. For simplicity, we assume that $\sum_{j=1}^{m} W_{l,i}(j) = m/2$, which is a realistic approximation for all the dense QC matrices involved in CCSDS encoding operations, according to all encoding methods outlined in Section IV. Consequently, each $\delta_i^{(e)}(j)$ is a function of $rL_m/2$ input bits. The accumulation between successive execution cycles adds another variable to the sum of input parameters of each register to total $rL_m/2 + 1$. The logical resources required for this operation, in terms of two-input XOR functions, are $rL_m/2$, for each $u_i(j)$. Given that the total number of register bits is cm, the total logic resources needed are those displayed in Table I. According to the previous analysis, each LFSR input depends approximately on $rL_m/2 + 1$ bits. Consequently, a reasonable approximation for the two-input logic levels required is $\lceil \log_2(rL_m/2) \rceil$. The introduced architecture accomplishes the calculation of the cm bits of vector $p_i(j)$ in m/L_m cycles. Note that this value is independent of c and r.

IV. LDPC ENCODER ARCHITECTURES

In this section, we briefly describe all the applicable LDPC encoding methods. For each one, we propose architectures leveraging the algorithm introduced in the previous section. In all cases, we calculate analytically the resources required for a hardware implementation and the performance metrics in terms of critical path and number of cycles needed to encode one input block frame. These are compared with our estimations of the corresponding requirements if similar hardware implementations based on previous approaches were used. Typically, hardware resources estimations for each listed work are different from the corresponding authors' calculations,

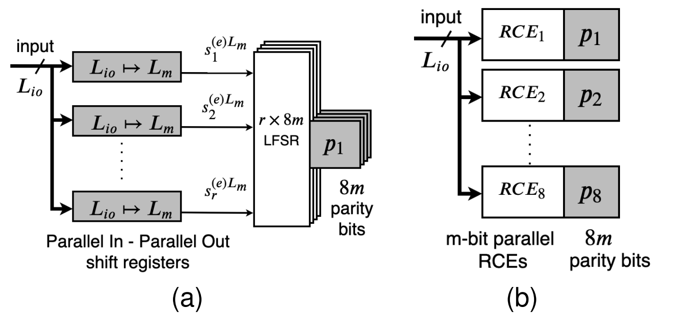


Fig. 3. Proposed implementation according to (a) the direct method and (b) compared to a parallel implementation of [14]. $L_{\text{io}} = rL_m$.

TABLE II
DIRECT METHOD ESTIMATIONS

Work	Rate	Logic functions	FFs	Logic levels	Enc. cycles
Proposed	1/2	8192χ+256	2048χ	3	64χ
Fig. 3(a)	2/3	8192χ+1024	1536χ	4	32χ
$L_{io}=rL_m$, L_m=2	4/5	8192χ+4096	1280χ	5	16χ
Based on [14]	1/2	19456χ	1024χ		64χ
Fig. 3(b)	2/3	10240χ	512χ	5	32χ
L_{io}=2r	4/5	5376χ	256χ		16χ

χ=1, 4, 16 for block lengths 1K, 4K, 16K correspondingly

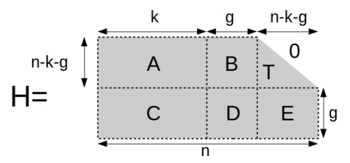


Fig. 4. Structure of H matrix for the R–U method.

since each work is based on different input–output bus architectures and different hardware primitives (e.g., RAM, ROM, and XOR gates). To have an independent basis for comparison in this article, the only combinatorial logic primitives are twoinput, single output binary logic functions and the only memory elements are flip-flops. The clock frequency of a hardware implementation is affected by the critical path of the combinatorial logic of the design. Consequently, in each one of the provided implementations, we estimate the maximum number of logic levels between flip-flops as the number of twoinput functions of the longest combinatorial path of the design. For the resource calculations, we have excluded any control logic introduced by a practical implementation of the architecture, focusing on the algorithm. Finally, the special case of C2 code is treated separately and compared to previous work.

In the subsequent analysis, we define the k-bit input binary vector as s and the n-bit generated codeword as c. Encoding, therefore, is described as the mapping $s \rightarrow c$, which is uniquely defined by the parity check matrix of the code H, so that the equation $cH^T = 0$ is satisfied. Generally, most LDPC codes are systematic, which means that c has the form $c = [s\ \ p]$, where subvector p consists of the $n-k$ parity bits.

A. Direct Method

The direct method involves the application of Gaussian elimination in order to calculate the generator matrix G from the null space of the parity-check matrix H of the code, by solving the equation $GH^T = 0$ in $\mathbb{F}_2$. A codeword can thus be calculated through the vector–matrix multiplication $c = sG$. For all the practical QC-LDPC codes, the generator matrix can be calculated in the systematic circulant form: $G = \left[I_k\ \ W_{n-k}\right]$, where I_k is the $k \times k$ identity matrix and W_{n-k} is an $r \times c$ array of dense cyclic submatrices. Note that according to Section II, the last $4m$ bits are punctured (not transmitted). Consequently, the encoders implementing this method need only store the $8rm$ bits of the first rows (or columns) of these circulants. The resulting $W_{n,k}$ matrix and consequently its constituent circulants are dense matrices.

The direct method implementation according to the proposed architecture is displayed in Fig. 3(a). The architecture introduced in Section III is used for the multiplication of $p = sW$. Additional logic is required in this case for rearranging the input data of vector s into $s_l^{(e)L_m}$, as required for the LFSR operation. This can be easily implemented by a series of r nonsymmetric parallel-in–parallel-out (PIPO) shift registers. Input data are provided to the encoder L_{io} bits at each clock cycle and stored in one of these shift registers, depending on the current subvector s_i. During calculation (as e increases from 0 to $m/L_m - 1$), L_m bits of each subvector shift register are shifted out. Processing of each input frame is concluded in m/L_m cycles, consequently, if $L_{\text{io}} = rL_m$, the input to the LFSR module never stagnates and the maximum throughput is achieved.

The architectures proposed in [12], [14] and [16] are also implementations of the direct encoding method. Compared to our previous work [12], the introduced architecture in this article is algorithmically equivalent to the subcases of $L_a = 8, 16, 32$ for rates 1/2, 2/3, and 4/5 correspondingly. The serial SRAA architecture proposed in [16] is limited in terms of maximum achievable throughput, whereas the parallel SRAA architecture results in large critical paths, because of large XOR operations. The architecture implemented in [14], which is recommended in CCSDS standard [7], processes input bits serially. However, a parallel implementation that processes L_{io} input bits at each clock cycle can be built as in Fig. 3(b) and compared to the proposed implementation of the direct method. Examples of resources and performance estimations are listed in Table II and compared to our proposed direct method encoder. For the same encoding cycles, the proposed architecture achieves shorter critical path and requires fewer combinatorial resources for rates 1/2 and 2/3, at the cost of additional flip-flops, which favors FPGA implementations. For rate 4/5, however, both architectures achieve the same throughput performance, whereas the fixed overhead of the PIPO registers required for the generation of vectors $s_l^{(e)L_m}$ dominates required resources.

B. R–U Method

The R–U method [20] solves the parity-check equation $Hc^T = 0$ with complexity that is approximately linear to the block length, provided that the parity-check matrix can be transformed into an approximate lower-triangular form, as in Fig. 4. For systematic codes, s has the form $c = [s\ p_1\ p_2]$, where p_1, p_2 are parity bits vectors of length g and $m - g$, respectively. If $\varphi = ET^{-1}B + D$ the parity bits are calculated according to the following equations:

$$p_1^T = \varphi^{-1}(ET^{-1}A + C)s^T \quad (13)$$

$$p_2^T = T^{-1}\big(As^T + Bp_1^T\big). \quad (14)$$

The above equations involve many sparse matrices, but only a single $g \times g$ dense, namely φ^{-1}, which is the critical factor affecting the performance of the encoder. The parity check matrix of many widely adopted LDPC codes has been specifically designed so that the parameter g is small, or even zero (as in the case of DVB-S2), or the matrix φ^{-1} has an efficient structure. For example, the φ matrix of the LDPC codes adopted for IEEE 802.11ac/n, 802.16e, and many other applications is the $g \times g$ identity matrix, which results in efficient hardware implementations of the encoders. For AR4JA codes, the parity check matrix can be transformed into approximate lower triangular QC form by shifting the last four circulants ($4m$ bits) by eight columns ($8m$ bits) to the left. Since these permuted bits are punctured, this permutation does not affect the encoder's output. The resulting dense φ^{-1} for AR4JA codes is a 4×4 array of $m \times m$ circulants, and matrix T^{-1} is the identity matrix.

The implementation of the R–U method according to the proposed architecture is displayed in Fig. 5(a). The four L_m-bit vectors $v_i^{(e)L_m}$ are calculated by rotating input vectors s_i by L_m bits at each clock cycle. The hardware implementation of this particular operation is feasible, since matrix $EA + C$ is sparse, and each $v_i^{(e)L_m}(j)$ depends only on at most one bit of each s_i, since $(EA + C)$ is QC. Vectors $v_i^{(e)L_m}(j)$ are stored in intermediate registers to reduce the critical path. After the m/L_m cycles required for the dense matrix multiplication, the $4m$ vector f^T is stored into the register of the LFSR. Vector b^T is calculated based on the addition of the permutations of f^T. Since node v_1 is not connected to node c_1 in Fig. 1, it is evident that for all AR4JA codes, the first $4m$ bits of matrix A are zero. Vector a^T is calculated in parallel with f^T and stored into the intermediate shift registers in the figure. In the final step of the process, the calculation of $b^T = Bf^T$ and addition of a^T are performed at the same clock cycle to form the parity bits.

In previous work, implementing the R-U method and applicable to CCSDS codes, the architecture proposed in [22] for Block-LDPC codes is not expected to result in an efficient implementation of CCSDS code encoders. The resources and performance metrics are dominated by the dense matrix operation involving matrix φ^{-1}. A more suitable approach would be based on the architecture described in [23], which leverages the SRAA modules introduced in [16] for φ^{-1} multiplication. A diagram of a CCSDS encoder based on that architecture is displayed in Fig. 5(b), in which the layered approach is followed for sparse matrix operations. Table III summarizes the estimated resources and performance metrics of the two encoder architectures displayed in Fig. 5. Although the architecture of 5(b) requires slightly less combinatorial resources and has a smaller critical path than the one proposed, it concludes encoding of one input frame in eight times more cycles, since the operations on each layer

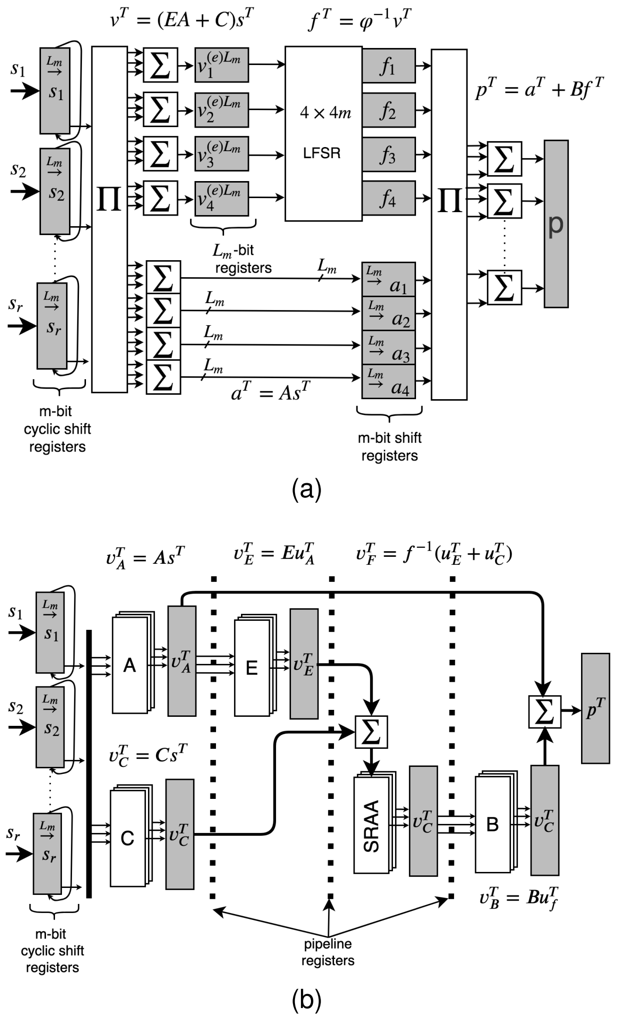


Fig. 5. Proposed implementation according to (a) the R–U method and (b) compared to the classical approach taken in [23]. All bits of vector s are considered to be available at the encoder's input.

TABLE III
R–U METHOD ESTIMATIONS

Work	Rate	Logic functions	FFs	Logic levels	Enc. cycles
Proposed	1/2	6144χ+56	3072χ+8	3	64χ
Fig. 5(a)	2/3	4096χ+184	2048χ+8	5	32χ
$L_m = 2$	4/5	3072χ+448	1536χ+8	6	16χ
Based	1/2	5120χ+26	8192χ	2	512χ
on [23]	2/3	3584χ+58	4608χ	3	256χ
Fig. 5(b)	4/5	2816χ+116	2816χ	4	128χ

χ=1, 4, 16 for block lengths 1K, 4K, 16K correspondingly

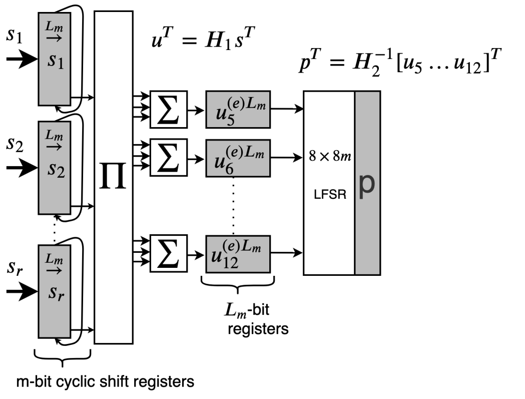


Fig. 6. Proposed implementation according to the partitioned-H method. The architecture of Fig. 2 is used for dense matrix operation involving H_2^{-1}. All bits of vector s are considered to be available at the encoder's input.

and the SRAA modules are executed serially. Consequently, the introduced architecture achieves considerably increased throughput performance.

C. Partitioned H Methods

This class of methods is based on the fact that, for systematic codes, the parity-check matrix can be partitioned into a $(n-k) \times k$ submatrix H_1 and a $(n-k) \times (n-k)$ submatrix H_2, where $H = [H_1\ H_2]$, so that the parity bits vector p^T can be calculated as follows:

$$p^T = H_2^{-1} H_1 s^T. \tag{15}$$

Submatrix H_1 is sparse and the vector $H_1 s^T$ can be easily calculated. For many practical codes, submatrix H_2^{-1} exhibits a regular structure, which facilitates the required calculations. A common structure in the parity-check matrix of many codes (IEEE 802.11 n/ac, 3GPP2, and DVB-S2) is the dual-diagonal: H_2 matrix or at least its rightmost part is a dual-diagonal matrix. For CCSDS codes, H_2^{-1} is a 12×12 array of dense $m \times m$ circulants. However, a simplification arising from the protograph structure of Fig. 1 is that node v_1 is not connected to c_1, which means that the first $4m$ rows of H_1 are zero. In addition, since the last $4m$ bits of the codeword are punctured, the last $4m$ rows of matrix H_2^{-1} can be omitted.

The block diagram of the proposed architecture implementing this method is displayed in Fig. 6. At each clock cycle, vectors $u_i^{(e)L_m}$ are calculated directly from input bits, in a way similar to $v_i^{(e)L_m}$ of the R–U method and stored into L_m-bits registers. These intermediate results are provided to an LFSR, which implements the binary multiplication with H_2^{-1}. The calculation of p^T is concluded after m/L_m cycles.

Table IV summarizes the involved estimations for the example case of $L_m = 2$. The existing implementations of the partitioned-H method, either directly or through triangular decomposition, require a specific structure of the parity check matrix, incompatible with CCSDS codes. Consequently, this is the first time that an encoder architecture for this method is proposed and thus no comparisons can be made.

TABLE IV
PARTITIONED-H METHOD ESTIMATIONS

Work	Rate	Logic functions	FFs	Logic levels	Enc. cycles
Proposed	1/2	10240χ+24	2048χ+16		64χ
Fig. 6	2/3	6144χ+88	1536χ+16	3	32χ
$L_m = 2$	4/5	4096χ+216	1280χ+16		16χ

χ=1, 4, 16 for block lengths 1K, 4K, 16K correspondingly

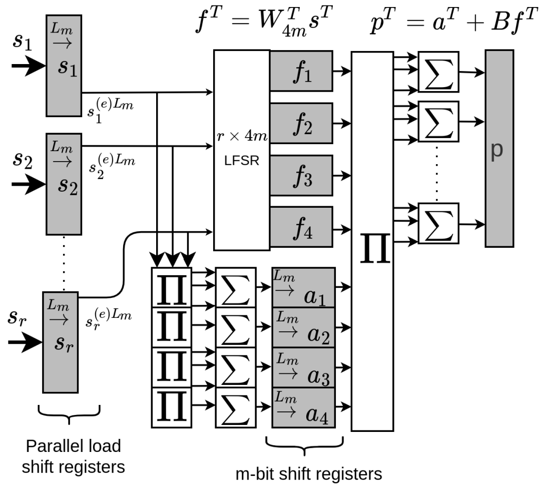


Fig. 7. Proposed implementation according to the hybrid method. The architecture of Fig. 2 is used for the dense matrix operation involving W_{4m}^T. All bits of vector s are considered to be available at the encoder's input.

D. Hybrid Method

Cohen and Parhi [30] proposed a hybrid approach for IEEE 802.3an codes, according to which the parity bits are calculated using a mix of the direct and the R–U method: the first subvector p_1 of g parity bits in R–U (13) is calculated from the generator matrix G, according to the direct method, avoiding thus the dense vector–matrix operations involving φ^{-1}, whereas calculation of p_2^T is done as in the R–U method, according to (14). We denote by W_{4m}^T as the last $4m$ columns of the generator matrix of the code and it is evident that $W_{4m} = \varphi^{-1}(ET^{-1}A + C)$.

In this article, we propose encoders for CCSDS codes based on this hybrid method, leveraging the introduced dense matrix multiplication algorithm. The block diagram of the proposed encoder in this case is depicted in Fig. 7. Vector p_1^T in the original work is renamed as f^T in this article. It is calculated from the generator matrix in an identical way as in the direct method. Intermediate vectors a^T and b^T as well as the parity bits are calculated as in the R–U method. Table V lists the resources and performance estimations for the encoder of Fig. 7. The architecture proposed in [30] is not suitable for CCSDS codes as is, and comparisons cannot be made.

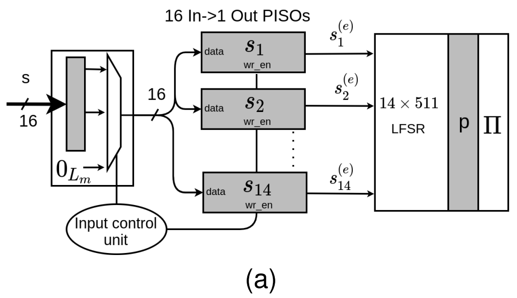
E. Special Case: C2 Code

For the C2 code, the only applicable method is the direct, at least without any modification of the parity-check matrix. Special manipulation is required, however, because of the

TABLE V
HYBRID METHOD ESTIMATIONS

Work	Rate	Logic functions	FFs	Logic levels	Enc. cycles
Proposed	1/2	8192χ+8	3072χ	3	64χ
Fig. 7	2/3	7168χ+40	2048χ	4	32χ
L_m=2, L_{io}=2r	4/5	6656χ+112	1536χ	5	16χ

χ=1, 4, 16 for block lengths 1K, 4K, 16K correspondingly



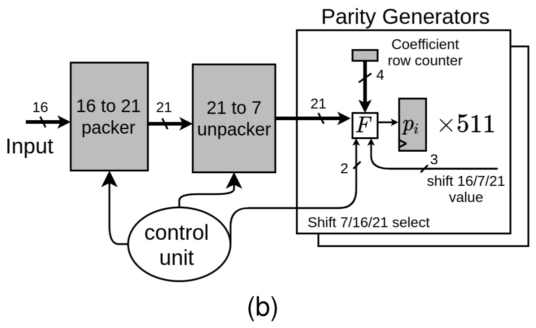


Fig. 8. Proposed stream input implementation for (a) C2 code and (b) compared to the packing–unpacking scheme proposed in [15].

problematic circulant size of the code, which is not a power of 2. In our work [12], we had introduced an efficient scheduling of the stream of input bits, by adding one zero bit every 511 input bits. The effect of that addition was counterbalanced by performing one less rotation of the parity bits shift register. In this article, we applied the architecture depicted in Fig. 3 by setting $m = 511$ and $r = 14$. A block diagram of the proposed encoder is displayed in Fig. 8(a). Input bits are supplied to the encoder in pieces of L_{io} bits and are padded with zeros, as necessary, by setting $s_i^{(m/L_m)-1} = 0_{L_m}$, where 0_{L_m} is a sequence of L_m zeros. The effect of these extra $14L_m$ zeros added to each circulant can be balanced by a permutation of the calculated parity bits by L_m positions to the left.

In order to address the problematic circulant size of the code, Miles *et al.* [15] proposed a packing–unpacking scheme, as displayed in Fig. 8(b). Input data are packed into groups of 21 bits, before they participate in the corresponding parity calculation. An architecture, which is algorithmically equivalent to the recursive convolutional encoders (RCEs) mentioned in [14], implements the cumulative parity calculations. At each clock cycle, the RCEs perform a shift or a shift-accumulate operation of 7, 16, or 21 bits. In Fig. 8(b), we model the input of each RCE register (flip-flop) as a binary function generator of 30 parameters. Note also that because of the difference between the input bus size (16 bit) and the shift-accumulate step (21 or 7 bit), latent cycles are introduced in the parity generators' operation.

TABLE VI
C2 CODE ESTIMATIONS

Work	Logic functions	FFs	Logic levels	Enc. cycles
Proposed (Fig. 8(a))	7602	8176	3	510
[15] (Fig. 8(b))	29638	1107	5	448

Table VI compares the analytical estimations of the proposed architecture and the one based on [15]. The two architectures offer different tradeoffs: Fig. 8(a) is optimized for combinatorial resources utilization and logic levels, whereas Fig. 8(b) for flip-flop use and it concludes encoding in approximately 12% less cycles. The efficiency of each architecture, therefore, depends on the targeted technology and the parameter which needs to be optimized. Targeting high throughput on FPGA technology, in which combinatorial logic is mapped to LUTs and flip-flops are an abundant resource, the hardware implementation of the proposed architecture achieves a higher clock rate. This is because of the fewer logic levels and the lower routing delays imposed by the total footprint of the proposed encoder. The higher clock rate compensates for the increased number of cycles required for encoding so that the actual throughput of the proposed architecture is higher. In our measurements, targeting Virtex-5/XC5VLX110T-1 FPGA, we were able to achieve a 30% higher clock rate with the proposed architecture, compared to that depicted in Fig. 8.

V. IMPLEMENTATION RESULTS

In this section, we demonstrate practical CCSDS encoder implementations, based on the architectures proposed in Section IV. The different encoding methods offer various trade-offs between combinatorial logic and flip-flop usage and critical path logic levels. Note also that the implementations of the R–U, partitioned-H, and hybrid methods described in the previous section require that all input bits are available at the encoder's input, while the direct method can encode a stream of input data. The PIPO shift registers, however, can be substituted by parallel load shift registers, with approximately the same resources. Consequently, comparisons between the different encoding methods can be made, based on the data of Tables II–V.

For each encoding method, based on the comparisons of the analytical estimations of the previous section, we selected the most efficient architecture for FPGA implementation. For rate 1/2 AR4JA codes, we developed hardware implementations based on the R–U method, according to Fig. 5(a). For the other rates of AR4JA codes, we implemented the partitioned-H method of Fig. 6, with $L_m = 4$. For C2 code, we implemented the architecture of Fig. 8(a). Control logic for the operation of the encoders has also been included in the implemented design. FPGA-in-the-loop verification and validation has been performed on the implemented encoders, based on

TABLE VII
AR4JA IMPLEMENTATION RESULTS (SYNTHESIZED DESIGN)

Work	Rate	Codeword length (bits)	Encoding Method	FPGA technology/ Demonstrated device	Resources LUTs	Resources Flip-Flops	Clock (MHz)	Throughput (Gbps)
[12]	1/2	2048	Direct	Virtex-5/XC5VLX110T-1	3258	2176	230	3.59
[13]	4/5	5120	Direct	Virtex-7/XC7VX485T-1	101173	141411	200	8
Proposed $L_m = 4$	1/2	2048	R-U	Virtex-5/XC5VLX110T-1	2772	3163	375	12
	2/3	3072	Partitioned-H		2221	1762	360	11.52
	4/5	1280	Partitioned-H		1890	1428	380	12.16
	1/2	8192	R-U	Virtex-7/XC7VX485T-1	8945	12420	495	15.84
	2/3	6144	Partitioned-H		6304	6481	460	14.72
	4/5	5120	Partitioned-H		4543	5691	515	16.48
	1/2	32768	R-U	ZynqUltraScale+ MPSoC XCZU9EG-2	37682	49190	380	12.16
	2/3	24576	Partitioned-H		28922	25380	600	19.2
	4/5	30480	Partitioned-H		23459	20940	780	24.96

TABLE VIII
C2 IMPLEMENTATION RESULTS (SYNTHESIZED DESIGN)

Work	Targeted FPGA technology/ Demonstrated device	Resources LUTs	Resources Flip-Flops	Resources BRAMs	Clock (MHz)	Throughput (Gbps)
[12]	Virtex-5/XC5VLX110T-1	9128	1156	0	200	3.12
[31]	Kintex-7/XC7325T	54747	92233	38	297	2.97
[32]	Virtex-5/XC5VLX30-1	9.2K	N/A	N/A	164	1.14
[33]	Virtex-6/XC6VLX240T-1	N/A*	N/A*	0	418	0.418
[34]	Artix-7/100T-1	6873	3219	1	239	1.55
[35]	Various FPGA/ASIC	N/A	N/A	0	200	1.6
Proposed	Virtex-5/XC5VLX110T-1	3338	1340	0	260	4.16

*The design takes up 290 slices. One Virtex-6 slice contains 4 LUTs and 8 Flip-Flops.

a GNU/Octave bit-accurate model, targeting Zynq UltraScale+ multi-processor system-on-chip (MPSoC) technology and specifically the XCZU9EG-2 FPGA of ZCU102 evaluation kit. All the implemented encoders were designed as IP cores for the targeted FPGA family.

Implementation results for AR4JA codes are displayed in Table VII. Although Zynq UltraScale+ MPSoC was targeted for on-chip verification, Virtex-5 and Virtex-7 devices were also targeted for synthesis, so that implementation results can be comparable to the other implementations given in Table VII. The implemented architectures are listed among all the recent implementations targeting specifically AR4JA CCSDS codes, to which direct comparisons can be made. Because of the optimizations introduced in this article, the efficiency of the introduced architecture results in more than three times higher throughput, with comparable resources, compared to our previous work [12]. Note, however, that the hardware encoder in [12] implemented additional functions of the data link layer of the TM-SDLP protocol (randomization and synchronization). The large XOR operations over 2048 bits, which are introduced by the architecture in [13] result in a large amount of required resources and poor timing. Finally, to the best of our knowledge, there is no other documented implementation of 16-K AR4JA CCSDS codes.

Table VIII compares the encoder based on Fig. 8(a) with prior work targeting CCSDS C2 code. The proposed encoder implements all the functions of the TM-SDLP protocol. The parallel-in–serial-out (PISO) registers in Fig. 8(a) are mapped to LUT RAM, hence the difference in flip-flop count between estimations in Table VI and actual count in Table VIII. Our previous work in [12] implements the direct encoding method, based on different scheduling of the input bits. Instead of the PISO registers of Fig. 8(a), it is based on an ping-pong buffer at the encoder input, which handles the boundaries between the 511-bit circulants. Compared to that work, the architectural optimizations of the current work result in an increase in throughput, while at the same time minimizing the required resources. The work in [31] also implements the direct encoding method. It buffers an entire input frame and partitions it into 14 sub-vectors of 511 bits each. All parity bits are being calculated in parallel. It requires, however, significantly more resources than the encoder of the current work, while at the same time achieving lower throughput performance, even on a more advanced Kintex-7 FPGA. Prior work listed in [32]–[35] refers to commercial products. The encoder in [32] has 8-bit input–output interfaces and implements the direct encoding method. It stores two circulant tables: one for processing input bits which correspond to the same 511-bit circulant of the generator matrix and another for the cases when the eight input bits span two circulants. This implementation requires a large amount of resources. A low complexity and low-throughput encoder is provided in [33]. It implements the direct encoding method and the input–output buses are bit-serial. Another encoder for C2 code is provided in [34]. It also implements the direct encoding method and input–output bus is 8 bits. Block RAM is used for input–output buffering. Finally, for [35], no information about the underlying architecture is provided other than what is displayed in Table VIII. In all cases, our implementations achieve state-of-the-art performance, with a fraction of the required resources.

VI. Conclusion

In this article, we have introduced a novel architecture for the efficient multiplication of a bit vector with a dense QC binary matrix and proposed encoder architectures for all the applicable LDPC encoding methods proposed so far. For each method, we have derived analytical quantitative measures of complexity and resource budget. Compared to existing approaches, the proposed encoders achieve state-of-the-art throughput performance for the specified codes, while at the same time keeping resource utilization low. The proposed encoders have been implemented on different FPGA technologies, verified and validated on Zynq UltraScale+ MPSoC hardware.

References

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

[2] Y. Fang, G. Bi, Y. L. Guan, and F. C. M. Lau, "A survey on protograph LDPC codes and their applications," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 4, pp. 1989–2016, 4th Quart., 2015.

[3] Y. Fang, G. Han, G. Cai, F. C. M. Lau, P. Chen, and Y. L. Guan, "Design guidelines of low-density parity-check codes for magnetic recording systems," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 2, pp. 1574–1606, 2nd Quart., 2018.

[4] Y. Fang, P. Chen, L. Wang, and F. C. M. Lau, "Design of protograph LDPC codes for partial response channels," *IEEE Trans. Commun.*, vol. 60, no. 10, pp. 2809–2819, Oct. 2012.

[5] P. Chen, Z. Xie, Y. Fang, Z. Chen, S. Mumtaz, and J. J. P. C. Rodrigues, "Physical-layer network coding: An efficient technique for wireless communications," *IEEE Netw.*, pp. 1–7, 2019.

[6] Y. Fang, P. Chen, G. Cai, F. C. M. Lau, S. C. Liew, and G. Han, "Outage-limit-approaching channel coding for future wireless communications: Root-protograph low-density parity-check codes," *IEEE Veh. Technol. Mag.*, vol. 14, no. 2, pp. 85–93, Jun. 2019.

[7] *TM Synchronization Channel Coding*, CCSDS Standard 131.0-B-3, Sep. 2017.

[8] G. Tzimpragos, C. Kachris, D. Soudris, and I. Tomkos, "A low-complexity implementation of QC-LDPC encoder in reconfigurable logic," in *Proc. 23rd Int. Conf. Field Program. Logic Appl.*, Sep. 2013, pp. 1–4.

[9] N. A. F. Neto, J. R. S. de Oliveira, W. L. A. de Oliveira, and J. C. N. Bittencourt, "VLSI architecture design and implementation of a LDPC encoder for the IEEE 802.22 WRAN standard," in *Proc. 25th Int. Workshop Power Timing Modeling, Optim. Simulation (PATMOS)*, Sep. 2015, pp. 71–76.

[10] X. Wang, T. Ge, J. Li, C. Su, and F. Hong, "Efficient multi-rate encoder of QC-LDPC codes based on FPGA for WIMAX standard," *Chin. J. Electron.*, vol. 26, no. 2, pp. 250–255, Mar. 2017.

[11] T. T. B. Nguyen, T. Nguyen Tan, and H. Lee, "Efficient QC-LDPC encoder for 5G new radio," *Electronics*, vol. 8, no. 6, p. 668, Jun. 2019.

[12] D. Theodoropoulos, N. Kranitis, and A. Paschalis, "An efficient LDPC encoder architecture for space applications," in *Proc. IEEE 22nd Int. Symp. Line Testing Robust Syst. Design (IOLTS)*, Jul. 2016, pp. 149–154.

[13] Z. Wang, X. Hao, C. Lin, and Q. Wu, "An efficient hardware LDPC encoder based on partial parallel structure for CCSDS," in *Proc. IEEE 18th Int. Conf. Commun. Technol. (ICCT)*, Oct. 2018, pp. 136–139.

[14] K. Andrews, S. Dolinar, and J. Thorpe, "Encoders for block-circulant LDPC codes," in *Proc. Int. Symp. Inf. Theory (ISIT)*, Sep. 2005, pp. 2300–2304.

[15] L. H. Miles, J. W. Gambles, G. K. Maki, W. E. Ryan, and S. R. Whitaker, "An 860-Mb/s (8158,7136) low-density parity-check encoder," *IEEE J. Solid-State Circuits*, vol. 41, no. 8, pp. 1686–1691, Aug. 2006.

[16] Z. Li, L. Chen, L. Zeng, S. Lin, and W. Fong, "Efficient encoding of quasi-cyclic low-density parity-check codes," *IEEE Trans. Commun.*, vol. 53, no. 11, p. 1973, Nov. 2005.

[17] S.-W. Yen, S.-Y. Hung, C.-L. Chen, H.-C. Chang, S.-J. Jou, and C.-Y. Lee, "A 5.79-Gb/s energy-efficient multirate LDPC codec chip for IEEE 802.15.3c applications," *IEEE J. Solid-State Circuits*, vol. 47, no. 9, pp. 2246–2257, Sep. 2012.

[18] D. Chen, P. Chen, and Y. Fang, "Low-complexity high-performance low-density parity-check encoder design for China digital radio standard," *IEEE Access*, to be published.

[19] F. Wang, P. Zhang, X. Wan, and J. Liu, "Design of a multi-rate quasi-cyclic low-density parity-check encoder based on pipelined rotate-left-accumulator circuits," in *Proc. 7th Int. Congr. Image Signal Process.*, Oct. 2014, pp. 1105–1109.

[20] T. J. Richardson and R. L. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 638–656, Feb. 2001.

[21] D.-U. Lee, W. Luk, C. Wang, and C. Jones, "A flexible hardware encoder for low-density parity-check codes," in *Proc. 12th Annu. IEEE Symp. Field-Program. Custom Comput. Mach.*, Apr. 2004, pp. 101–111.

[22] H. Zhong and T. Zhang, "Block-LDPC: A practical LDPC coding system design approach," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 52, no. 4, pp. 766–775, Apr. 2005.

[23] H. Zhang, J. Zhu, H. Shi, and D. Wang, "Layered approx-regular LDPC: Code construction and encoder/decoder design," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 55, no. 2, pp. 572–585, Mar. 2008.

[24] P. Zhang, S. Yu, C. Liu, and L. Jiang, "Efficient encoding of QC-LDPC codes with multiple-diagonal parity-check structure," *Electron. Lett.*, vol. 50, no. 4, pp. 320–321, Feb. 2014.

[25] Y. Kaji, "Encoding LDPC codes using the triangular factorization," *IEICE Trans. Fundam. Electron., Commun. Comput. Sci.*, vol. E89-A, no. 10, pp. 2510–2518, Oct. 2006.

[26] J.-N. Su, Z. Jiang, K. Liu, X.-Y. Zeng, and H. Min, "An efficient low complexity LDPC encoder based on LU factorization with pivoting," in *Proc. 6th Int. Conf. (ASIC)*, vol. 1, Oct. 2005, pp. 168–171.

[27] J. Hu and K. Jiang, "The improved LU-based decomposition algorithm for sparse matrix of LDPC code," in *Frontiers in Computer Education* (Advances in Intelligent and Soft Computing), vol. 133, S. Sambath and E. Zhu, Eds. Berlin, Germany: Springer, 2012, pp. 867–874.

[28] H. Yin, W. Du, and N. Zhu, "Design of improved LDPC encoder for CMMB based on SIMD architecture," in *Proc. IEEE 3rd Int. Conf. Inf. Sci. Technol. (ICIST)*, Mar. 2013, pp. 1292–1295.

[29] A. Mahdi and V. Paliouras, "A low complexity-high throughput QC-LDPC encoder," *IEEE Trans. Signal Process.*, vol. 62, no. 10, pp. 2696–2708, May 2014.

[30] A. E. Cohen and K. K. Parhi, "A low-complexity hybrid LDPC code encoder for IEEE 802.3an (10GBase-T) Ethernet," *IEEE Trans. Signal Process.*, vol. 57, no. 10, pp. 4085–4094, Oct. 2009.

[31] W. Ren and H. Liu, "The design and implementation of high-speed codec based on FPGA," in *Proc. 10th Int. Conf. Commun. Softw. Netw. (ICCSN)*, Jul. 2018, pp. 427–532.

[32] *LCE01C CCSDS (8160,7136) LDPC Encoder*, 1st ed., Small World Commun., Payneham South, SA, Australia, Apr. 2015. [Online]. Available: http://www.sworld.com.au/products/lce01c.html

[33] *LDPC NASA Encoder/Decoder IP Core Specificaton*, 2nd ed., IPrium LLC, Tomsk, Russia, Sep. 2014. [Online]. Available: https://www.iprium.com/bins/pdf/iprium_ug_ldpc_nasa_codec.pdf

[34] *COM-1811SOFT CCSDS LDPC C2 Code Encoder/Decoder VHDL Source Code Overview/IP Core Overview*, MSS, Oct. 2019. [Online]. Available: https://comblock.com/download/com1811soft.pdf

[35] Creonic GmbH. *CCSDS (8160, 7136) LDPC Encoder Decoder Product Brief*. Accessed: Feb. 25, 2019. [Online]. Available: https://www.creonic.com/wpcontent/uploads/PB_Creonic_CCSDS_ LDPC_FEC_IP.pdf