Low-complexity encoder implementation for LDPC codes in CCSDS standard

Ruixue Wang¹, Weigang Chen¹, and Changcai Han¹, a)

Abstract The traditional direct LDPC encoder in CCSDS standard for space application needs to store the first row in each submatrix of the generator matrix, making the circuit implementation complex. To solve this problem, a low-complexity encoder for LDPC codes is implemented in this letter. The encoder stores the vector in random-access memory (RAM). To implement the multiplication of sparse matrix and vector with limited hardware resources, the encoder takes the row indexes of nonzero entries in each column of sparse matrix as the write address of the RAM. Moreover, the shift-register-adder-accumulator is exploited to implement the multiplication of the dense core matrix and the vector, greatly reducing the storage and computation complexity. The LDPC encoder with the code rate of 1/2 in the CCSDS standard is implemented on Xilinx XC6VLX240T FPGA chip, and the implementation results indicate that the proposed encoder consumes 50% less hardware resources than the traditional direct encoder.

Keywords: LDPC codes, space applications, CCSDS, FPGA, encoder

Classification: Integrated circuits (memory, logic, analog, RF, sensor)

1. Introduction

Low-density parity-check (LDPC) codes have a remarkable error correction capability near the Shannon limit [1, 2] using iterative decoding on very sparse parity-check matrices [3, 4, 5]. The Consultative Committee for Space Data Systems (CCSDS) has standardized a number of protograph-based quasi-cyclic LDPC (QC-LDPC) codes for space communication protocols in [6]. In application scenarios like space telemetry, considering the precious hardware resources available in satellites [7, 8], the LDPC encoder should be implemented with very concise circuits. For some type of LDPC codes, the parity-check matrix is usually exploited to implement efficient and low complexity encoding [9, 10]. The implementation of the LDPC encoders in CCSDS standard usually uses the dense generator matrix directly [11, 12, 13, 14]. However, the straightforward implementation of these LDPC encoders suffers from high storage and computation overhead [15, 16, 17, 18].

The encoder architecture proposed in [19] does not need to store the dense block matrix, which realizes the bit-wise matrix vector multiplication (BMVM) without cyclic shifter. However, the use of a large number of I/O pins to generate the parity check bits may exceed the resource limit of field programmable gate array (FPGA). In [20], an efficient multi-rate encoder featured with high encoding speed and minimum hardware usage is proposed for the IEEE 802.22 Wireless Regional Area Network (WRAN) standard. In fifth generation (5G) new radio (NR), the bidiagonal architecture proposed in [21] is exploited to reduce the complexity of the encoder. Based on the triangular decomposition of the parity part in the parity check matrix, a low-complexity and high throughput QC-LDPC encoder architecture is proposed in [22]. However, these encoder architectures require a specific structure of parity check matrix for the code, which cannot be satisfied by CCSDS codes. For the LDPC codes adopted by CCSDS, the encoding is usually based on a direct multiplication of the information bit vector and the dense generator matrix derived from sparse parity check matrix [23], which can be implemented with a shift-register-adder-accumulator (SRAA) circuit [24, 25].

Previous works in [26, 27, 28, 29, 30] target CCSDS codes. A variety of encoding circuits based on shift registers have been proposed in [26]. The encoding complexity of these encoding circuits is linearly proportional to the number of parity check bits of the code, or the total bits of the code in case of a parallel approach. In [27], the proposed partial parallel encoder architecture involves wide XOR operations over a significant number of bits and requires much register resources for the shift registers. The high throughput encoder described in [28] exploits the parallel recursive convolutional encoder circuit structure to reduce the use of system registers at the cost of a four fold increase in the multiplication and accumulation units. The architecture proposed in [29] leverages the inherent parallelism of the quasi-cyclic structure by concurrently processing multiple bits, which achieves high throughput and low complexity. By partitioning and decomposing the generator matrix, our previous work obtains a quite small dense core matrix and several sparse matrices [30]. Since the size of the dense core matrix is one quarter of the generator matrix, and the SRAA structure can be adopted for the vector multiplication, achieving 50% less storage and computation complexity than the generator matrix for encoding. Besides, the multiplication of sparse matrix and vector can be implemented by a register-based scheme or random access memory (RAM)-based scheme. The encoding complexity of this two schemes has a linear relationship with the codeword length, leading to lower storage requirements and computation complexity.

In this letter, a low-complexity LDPC encoder for CCSDS standard codes is implemented and evaluated. As for the multiplication of sparse matrix and vector, its circuit implementation stores the vector in RAM and takes the row...
The dimensions of the submatrix \( H \) code rate of AR4JA codes can be depicted by \( g \) and 16384 bits) over three code rates: 1/2, 2/3 and 4/5. The codes are the combination of three block length (1024, 4096 Accumulate Repeat-4 Jagged-Accumulate (AR4JA) class of For deep-space communications, nine codes belonging to the 2. Encoding for low-density parity-check codes

For deep-space communications, nine codes belonging to the Accumulate Repeat-4 Jagged-Accumulate (AR4JA) class of LDPC codes are defined in the CCSDS standard. These codes are the combination of three block length (1024, 4096 and 16384 bits) over three code rates: 1/2, 2/3 and 4/5. The code rate of AR4JA codes can be depicted by \( g/(g+2) \), where \( g = 2, g = 4 \) and \( g = 8 \) corresponds to the code rate of 1/2, 2/3 and 4/5, respectively. The parity check matrices \( H \) consists of an array of circulant sparse submatrices with the dimension of \( M \times M \), where \( M \) is a parameter dependent on the block length and the code rate, and \( w = M/4 \) is set.

The parity check matrix of LDPC codes adopted by CCSDS is composed of two parts, i.e., \( H = [ H_m \ H_e ] \). The dimensions of the submatrix \( H_m \) and \( H_e \) are \( 3M \times (M \times g) \) and \( 3M \times 3M \), respectively. The submatrix \( H_e \) of LDPC codes for various code rates is of the following form:

\[
H_e = \begin{bmatrix}
I_M & 0_M & I_M + \Pi_1 \\
0_M & I_M & \Pi_2 + \Pi_3 + \Pi_4 \\
0_M & \Pi_7 + \Pi_8 & I_M
\end{bmatrix},
\]

where the identity matrix \( I_M \), the all-zero matrix \( 0_M \) and the permutation matrix \( \Pi_k \), \( k = 1, 2, \ldots, 8 \) all have the dimension of \( M \times M \). The method for calculating the permutation matrix is introduced in [6]. According to [30], the inverse of the submatrix \( H_e \) can be calculated by

\[
H_e^{-1} = \begin{bmatrix}
I_M & \Pi_1 & (I_M + \Pi_1)T(I_M + \Pi_1) \\
0_M & (I_M + \Pi_1 + \Pi_2 + \Pi_3)T(I_M + \Pi_1) & (I_M + \Pi_2 + \Pi_3 + \Pi_4)T(I_M + \Pi_1) \\
0_M & (I_M + \Pi_2 + \Pi_3)T(I_M + \Pi_1) & T(I_M + \Pi_1)
\end{bmatrix},
\]

where the dense core matrix \( T \) [30] has the dimension of \( M \times M \) and is of the form:

\[
T = I_M + (\Pi_2 + \Pi_3) \cdot (\Pi_2 + \Pi_3 + \Pi_4)^{-1}.
\]

According to the equation \( H \cdot e^T = 0 \), the following expression can be obtained

\[
t = H_m \cdot m^T = \begin{bmatrix} 0 & t_1^T & t_2^T \end{bmatrix}^T_{3M \times 1}.
\]
 According to Eq. (13), the vector $t_1$ and vector $t_3$ can be directly calculated from the information bit vector $m$ and the permutation matrices $\Pi_5$, $\Pi_6$, $\Pi_7$ and $\Pi_8$ without calculating the intermediate vector $t_2$, thus saving $M$ clock cycles. The simplified LDPC encoder architecture with the code rate of $R = 1/2$ is shown in Fig. 1. Specifically, to implement the multiplication of the permutation matrix and the vector, the vector is stored in RAM by controlling the write address of the RAM, and the traditional SRAA structure is adopted to implement the multiplication of the dense core matrix $T$ and the vector $t_3$.

As shown in Fig. 1, the encoding processes can be divided into five steps. It is assumed that the information bit vector $m_1$ and $m_2$ can be cached at the same time.

**Step 1:** In the first $M$ clock cycles, the information bit vector $m_1$ is stored into three RAMs, and the row indexes of the nonzero entries in each column of matrices $I_M$, $\Pi_1$ and $\Pi_5$ are used as the write addresses. Meanwhile, the information bit vector $m_2$ is stored into five RAMs, and the row indexes of the nonzero entries in each column of matrices $I_M$, $\Pi_5$, $\Pi_6$, $\Pi_7$ and $\Pi_8$ are used as the write addresses.

**Step 2:** The output of the RAMs storing $m_1^T$, $\Pi_1m_1^T$, $\Pi_5m_1^T$, $\Pi_6m_1^T$, $\Pi_7m_1^T$ and $\Pi_8m_1^T$ is passed through a seven-input XOR to compute the intermediate vector $t_3$, where the read addresses are in ascending order. This process consumes one clock cycle for reading RAM.

**Step 3:** Multiply the dense core matrix $T$ by the intermediate vector $t_3$ to compute the intermediate vector $t_4$, which is called “intermediate vector $t_4$ computing unit”. This process requires $M$ clock cycles.

**Step 4:** Store the intermediate vector $t_4$ into five RAMs, and the row indexes of the nonzero entries in each column of matrices $I_M$, $\Pi_1$, $\Pi_5$, $\Pi_6$ and $\Pi_7$ are used as the write addresses. This process consumes $M$ clock cycles.

**Step 5:** The output of the RAMs storing $t_4$ and $\Pi_1t_4$ is passed through a two-input XOR to calculate the parity check bit vector $p_1$. The output of the RAMs storing $m_1$ and $m_2$ is passed through a two-input XOR to calculate the intermediate vector $t_1$. The vector $t_1$ and the output of the RAMs storing $\Pi_1t_4$, $\Pi_3t_4$ and $\Pi_4t_4$ are passed through a four-input XOR to calculate the parity check bit vector $p_2$. This step requires $M + 1$ clock cycles.

According to the above five steps, the entire encoding process requires $3M + 2$ clock cycles, without considering the caching of information bit vector $m_1$ and $m_2$.

### 3.2 Low complexity circuit implementation for the simplified LDPC encoder architecture

The multiplication of the permutation matrix and the vector in Fig. 1 is implemented by storing the vector in RAM, and the row indexes of the nonzero entries in each column of the permutation matrix are taken as the write address of the RAM. The method for implementing $\Pi_5m_1^T$ is illustrated in Fig. 2, where the permutation matrix is composed of $4 \times 4$ circulant submatrices with the dimension of $w \times w$, and each row and column has only one nonzero entry. It can be seen from Fig. 2(a) that the information bit vector $m_1^T$ is divided into four groups of $w$-bit vectors $m_2^{20}, m_2^{21}, m_2^{22}$, $m_2^{23}$. These vectors are rearranged by the submatrices $\Pi_1^0, \Pi_1^1, \Pi_1^2, \Pi_1^3$ of the permutation matrix $\Pi_1$ to obtain $\Pi_1^0m_2^{20}, \Pi_1^0m_2^{21}, \Pi_1^0m_2^{22}, \Pi_1^0m_2^{23}$. Fig. 2(b) shows that the calculation of $\Pi_1^0m_2^{22}$ is implemented by using the row indexes of the nonzero entries in each column of submatrix $\Pi_1^0$ as the write address to store the vector $m_2^{22}$ in RAM.

As shown in Fig. 3, the calculation of $\Pi_5m_1^T$ can be implemented by controlling the write and read addresses of the RAM, where $d = \log_2M$. The row indexes of the nonzero entries in the first column of submatrices $\Pi_5^0, \Pi_5^1, \Pi_5^2$ and $\Pi_5^3$ are sequentially loaded as the value of $v$ every $w$ clock cycles.

As a core module of the encoder, the computation of
intermediate vector \( \mathbf{t}_1 \) implemented by the SRAA circuit is exhibited in Fig. 4, where matrix \( \mathbf{T} \) is composed of a \( 4 \times 4 \) array of circulant dense submatrices with the dimension of \( w \times w \). The first column of each circulant submatrix is stored in four ROMs with width of \( w \) and depth of four. At the rising edge of the 1, \( w + 1 \), \( 2w + 1 \) and \( 3w + 1 \) clock cycles, the values in the four ROMs are loaded into four circulant shift registers all with width of \( w \), i.e., \( sfr1, sfr2, sfr3 \) and \( sfr4 \). The four circulant shift registers are rotated one bit to the right at the rising edge in each clock, and the dense core matrix \( \mathbf{T} \) is generated after \( M \) clock cycles. The intermediate vector \( \mathbf{t}_3 \) is delivered to the \( \mathbf{t}_4 \) computing unit bit by bit. Denoting the single bit in \( \mathbf{t}_1 \) as \( \mathbf{t}_1 \), if \( t_3 \) is 1, the contents in the four circulant shift registers are accumulated bit by bit with the contents in the shift registers with width of \( w \), i.e., \( \text{reg1, reg2, reg3 and reg4} \). If \( t_3 \) is 0, the contents of four shift registers remain unchanged. The intermediate vector \( \mathbf{t}_4 \) obtained after \( M \) clock cycles is stored in the four shift registers, and sequentially output bit by bit by the multiplexer in \( M \) clock cycles.

3.3 Analysis of various codeword lengths and code rates

With minor modifications, the encoder architecture in [30] and the simplified encoder architecture described above can be applied to LDPC codes with various codeword lengths and code rates. Specifically, different codeword lengths require no modification of the encoder architecture and the circuit implementation, while different code rates corresponds to the same submatrix \( \mathbf{H}_m \) and different permutation matrices in the submatrix \( \mathbf{H}_m \). Therefore, for LDPC codes with various code rates, the difference in the encoder architecture lies in RAMs that store the information bit vector \( \mathbf{m} \). According to Eq. (4), for the LDPC codes with code rates of 2/3 and 4/5, the information bit vector \( \mathbf{m} \) are divided into four and eight parts, respectively. Under various code rates, the encoder architecture in [30] differs in calculating the intermediate vectors \( \mathbf{t}_1 \) and \( \mathbf{t}_2 \), while the simplified encoder architecture differs in calculating the intermediate vectors \( \mathbf{t}_1 \) and \( \mathbf{t}_3 \). The number of RAMs required by the two encoder architectures for various code rates is listed in Table I. It can be observed from Table I that the LDPC codes with high code rates require more RAMs for circuit implementation. The reason is that the higher the code rate, the larger the matrix \( \mathbf{H}_m \), and the more complicated the calculation of \( \mathbf{t}_1 \), \( \mathbf{t}_2 \) and \( \mathbf{t}_3 \). For the LDPC codes with code rate of 1/2, the simplified encoder architecture should be first adopted; for the LDPC codes with code rate greater than 1/2, the encoder architecture should be chosen based on a trade-off between complexity and latency according to specific requirements.

### Table I Number of required RAMs by two encoder architectures for various code rates.

| Code rate | [30] | Simplified |
|-----------|------|------------|
| 1/2       | 9+4  | 5+8        |
| 2/3       | 9+12 | 5+24       |
| 4/5       | 9+28 | 5+56       |

4. Complexity comparison and implementation results

In this section, the storage complexity is compared by the number of bits required to store for the three encoding methods, and the computation complexity is compared by the number of XOR and AND functions required for encoding. The results of the implemented LDPC encoder and a detailed comparison with the traditional direct encoder in terms of hardware resources are also reported.

4.1 Comparison of storage and computation complexity

In the subsequent analysis, the direct encoding method is exploited to multiply the information bit vector \( \mathbf{m} \) by the generator matrix \( \mathbf{W} \) to obtain the parity check bit vector \( \mathbf{p} \). The two-level encoding method proposed in [13] first calculates the intermediate vector \( \mathbf{t} \) with the sparse matrix \( \mathbf{H}_m \), and then calculates the parity check bit vector \( \mathbf{p} \) with the dense matrix \( \mathbf{H}_c^{-1} \). The specific memory bits needed by the three encoding methods are listed in Table II. In the CCSDS standard, \( M \) can take values such as 128, 256, 512, 1024, 2048, 4096 and 8192 for space telemetry application. In terms of the circuit implementation, only the first row or the first column of the dense matrix should be stored according to the quasi-cyclic characteristic of the matrix. As for the sparse matrices, only the location of nonzero entries in the first row (or column) of the submatrix needs to be stored. It can be seen from the Table II that at high code rates, the two-level encoding method performs significantly better than the direct encoding method in terms of memory bits. The encoding method used in this letter consumes much
less memory bits than the other two encoding methods at any code rate and codeword length.

| Code rate | Direct | Two-level | This work |
|-----------|--------|-----------|-----------|
| 1/2       | 16M    | $20\log_2(M/4) + 16M$ | $60\log_2(M/4) + 4M$ |
| 2/3       | 32M    | $52\log_2(M/4) + 16M$ | $92\log_2(M/4) + 4M$ |
| 4/5       | 64M    | $116\log_2(M/4) + 16M$ | $156\log_2(M/4) + 4M$ |

Furthermore, the computation complexity of the three encoding methods is listed in Table III. The following analysis takes the two-input XOR functions and two-input AND functions into consideration. The direct encoding method exploits the SRAA architecture to implement the multiplication of the information bit vector \(m\) and the dense matrix \(W\), so the number of XOR and AND functions are both \(2M\). The two-level encoding method exploits the architecture in [30] to implement the calculation of the intermediate vector \(t\), which requires three XOR functions. The multiplication of the dense matrix \(H_i^\perp\) and the intermediate vector \(t\) is implemented with the SRAA architecture, which requires \(2M\) XOR functions and \(2M\) AND functions. As for the method proposed in this letter, the calculation of the intermediate vectors \(t_1\) and \(t_3\) and the parity check bit vectors \(p_1\) and \(p_2\) requires a total of eleven XOR functions. The multiplication of the dense core matrix \(T\) and the intermediate vector \(t_2\) is implemented with the SRAA architecture, which requires \(M\) XOR functions and \(M\) AND functions. Considering that the minimum value of \(M\) is 128, it can be seen from the above analysis that the computation complexity of the proposed encoding method is about 1/2 of the direct encoding method and the two-level encoding method.

| Operation   | Direct | Two-level | This work |
|-------------|--------|-----------|-----------|
| XOR function | \(2M\) | \(2M + 3\) | \(M + 11\) |
| AND function | \(2M\) | \(2M\) | \(M\) |

### 4.2 Implementation results

The simplified encoder and the traditional direct encoder are implemented on the XC6VLX240T FPGA of Virtex-6 series manufactured by Xilinx (San Jose, CA, USA). The LDPC codes in the CCSDS standard with the code rate \(R = 1/2\), and the codeword length \(N = 2,048\) bits and \(N = 32,768\) bits are taken for performance comparison.

The FPGA implementation results of the traditional direct encoder and the encoder proposed in this work are summarized in Table IV, where the working clock frequency of the encoders is set to 200 MHz. As introduced in Section 3, the encoder mainly involves multiplying sparse matrix by vector and multiplying dense core matrix by vector. Since the size of the dense core matrix \(T\) is one quarter of the generator matrix \(W\), only the first row or column of the circulant submatrix needs to be stored, thus computing the intermediate vector \(t_0\) with the SRAA architecture can achieve 50% less storage and computation overhead than the traditional direct encoder. The use of RAM to implement the multiplication of sparse matrix and vector requires two counters to generate the write address and read address, thus requiring fewer Flip-Flops and LUTs. Therefore, it can be observed from Table IV that the Flip-Flops and LUTs consumed by the core module account for more than 85% of the proposed encoder. Compared with the traditional direct encoder, the proposed encoder reduces the consumption of Flip-Flops and LUTs by 56.5% for \(N = 2,048\) bits and 50.3% for \(N = 32,768\) bits. Besides, for the code length of 32,768, the proposed encoder can reduce the RAM resources by 47%.

### 5. Conclusion

The traditional direct encoder occupies much hardware resources. To solve this problem, a low-complexity LDPC encoder for space applications is implemented in this letter. The RAM is used to cache the vector for multiplication of sparse matrix and vector, and the write address and read address of the RAM are generated by two counters, which consumes quite less hardware resources. Meanwhile, the SRAA circuit is exploited to implement the multiplication of dense core matrix and vector. Since the dense core matrix is one quarter of the generator matrix in size, the storage and computation complexity is significantly decreased. The LDPC encoder with the code rate of 1/2 is implemented on Xilinx XC6VLX240T FPGA chip, and the implementation results indicate that the proposed encoder consumes 50% less hardware resources than the traditional direct encoder.

### Acknowledgments

This work was supported in part by National Natural Science Foundation of China (No. 61671324).

### References

[1] T.J. Richardson and R.L. Urbanke: “The capacity of low-density parity-check codes under message-passing decoding,” IEEE Trans. Inf. Theory 47 (2001) 599 (DOI: 10.1109/18.910577).

[2] M. Erozu, et al.: “DVB-S2 low density parity check codes with near Shannon limit performance,” Int. J. Satell. Commun. Netw. 22 (2004) 269 (DOI: 10.1002/sat.787).

[3] D.J.C. MacKay and R.M. Neal: “Near Shannon limit performance of low density parity-check codes,” Electron. Lett. 33 (1997) 457 (DOI: 10.1049/el:19970362).

[4] D.J.C. MacKay: “Good error-correcting codes based on very sparse matrices,” IEEE Trans. Inf. Theory 45 (1999) 399 (DOI: 10.1109/18.748992).

[5] P. Li, et al.: “Low density parity-check codes with semi-random parity check matrix,” Electron. Lett. 35 (1999) 38 (DOI: 10.1049/el:19990065).

[6] TM Synchronization and Channel Coding: CCSDS 131.0-B-3 (2017) https://public.ccsds.org/default.aspx.
[7] K. Andrews, et al.: “Encoders for block-circulant LDPC codes,” ISIT (2005) 2300 (DOI: 10.1109/ISIT.2005.1523758).

[8] D. Theodoropoulos, et al.: “An efficient LDPC encoder architecture for space applications,” IOLTS (2016) 149 (DOI: 10.1109/IOLTS.2016.7604689).

[9] T.J. Richardson and R.L. Urbanke: “Efficient encoding of low-density parity-check codes,” IEEE Trans. Inf. Theory 47 (2001) 638 (DOI: 10.1109/18.910579).

[10] H. Yin, et al.: “Design of improved LDPC encoder for CMNB based on SIMD architecture,” ICIST (2013) 1292 (DOI: 10.1109/ICIST.2013.6747774).

[11] F. Wang, et al.: “Design of a multi-rate quasi-cyclic low-density parity-check encoder based on pipelined rotate-left accumulator circuits,” CISP (2014) 1105 (DOI: 10.1109/CISP.2014.7003945).

[12] Y.M. Jung, et al.: “7.7 gbps encoder design for ieee 802.11ac qc-ldpc codes,” J. Semicond. Technol. Sci. 14 (2014) 419 (DOI: 10.5573/JSTS.2014.14.4.419).

[13] G. Ge and L. Yin: “Multi-rate LDPC encoder for high-speed satellite data transmissions,” J. Tsinghua Univ: Nat. Sci. Ed. 56 (2016) 656.

[14] H. Yasotharan and A.C. Carusone: “A flexible hardware encoder for systematic low-density parity-check codes,” MWSCAS (2009) 54 (DOI: 10.1109/MWSCAS.2009.5236155).

[15] A. Mahdi and V. Palious: “On the encoding complexity of quasi-cyclic LDPC codes,” IEEE Trans. Signal Process. 63 (2015) 6096 (DOI: 10.1109/TSP.2015.2463258).

[16] Z. Li, et al.: “Efficient encoding of quasi-cyclic low-density parity-check codes,” IEEE Trans. Commun. 54 (2006) 71 (DOI: 10.1109/TCOMM.2005.861667).

[17] L.H. Miles, et al.: “An 860-Mb/s (8158,7136) low-density parity-check encoder,” IEEE J. Solid-State Circuits 41 (2006) 1686 (DOI: 10.1109/JSSC.2006.877253).

[18] W. Ren and H. Liu: “The design and implementation of high-speed codec based on FPGA,” ICCSN (2018) 427 (DOI: 10.1109/ICCSN.2018.8488239).

[19] G. Tzimpragos, et al.: “A low-complexity implementation of QC-LDPC encoder in reconfigurable logic,” 23rd International Conference on Field Programmable Logic and App. (2013) 1 (DOI: 10.1109/FPL.2013.6645587).

[20] X. Wang, et al.: “Efficient multi-rate encoder of QC-LDPC codes based on FPGA for WIMAX standard,” Chin. J. Electron. 26 (2017) 250 (DOI: 10.1049/cje.2017.01.006).

[21] T.T.B. Nguyen, et al.: “Efficient QC-LDPC encoder for 5G new radio,” Electronics 8 (2019) 668 (DOI: 10.3390/electronics8060668).

[22] A. Mahdi and V. Palious: “A low-complexity-high throughput QC-LDPC encoder,” IEEE Trans. Signal Process. 62 (2014) 2696 (DOI: 10.1109/TSP.2014.2314435).

[23] K.S. Andrews, et al.: “The development of turbo and LDPC codes for deep-space applications,” Proc. IEEE 95 (2007) 2142 (DOI: 10.1109/JPROC.2007.905132).

[24] H. Zhang, et al.: “Layered approx-regular LDPC: code construction and encoder/decoder design,” IEEE Trans. Circuits Syst. I, Reg. Papers 55 (2008) 572 (DOI: 10.1109/TCSI.2008.916433).

[25] P. Zhang, et al.: “Efficient encoding of QC-LDPC codes with multiple-diagonal parity-check structure,” Electron. Lett. 50 (2014) 320 (DOI: 10.1049/el.2013.2390).

[26] T. Liang, et al.: “Efficient encoding of quasi-cyclic low-density parity check codes,” IAEAC (2018) 1189 (DOI: 10.1109/IAEAC.2018.8577709).

[27] Z. Wang, et al.: “An efficient hardware LDPC encoder based on partial parallel structure for CCSDS,” ICCT (2018) 136 (DOI: 10.1109/ICCT.2018.8599970).

[28] J. Xie, et al.: “High throughput multi-code LDPC encoder for CCSDS standard,” ASICON (2019) 1 (DOI: 10.1109/ASICON47005.2019.8983513).

[29] D. Theodoropoulos, et al.: “Efficient architectures for multigigabit CCSDS LDPC encoders,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 28 (2020) 1118 (DOI: 10.1109/TVLSI.2020.2975050).

[30] W. Chen, et al.: “Low-complexity encoder for LDPC codes in space applications,” Electron. Lett. 55 (2019) 1241 (DOI: 10.1049/el.2019.2417).