A Miniaturized LDPC Encoder: Two-Layer Architecture for CCSDS Near-Earth Standard

Jiaming Liu and Quanyuan Feng, *Senior Member, IEEE*

***Abstract*—Quasi-cyclic low-density parity-check (QC-LDPC) codes, which have been adopted by the Consultative Committee for Space Data Systems (CCSDS), are widely used in Deep-Space (AR4JA) and Near-Earth (C2) communications. A large number of encoder architectures with CCSDS recommended standard have been proposed. But the existing architectures are not efficient enough in high-throughput implementations, many architectures have a lot of logical resources and registers. In this brief, we introduce a novel architecture with low resource utilization. A grouping algorithm is used to extract the common subexpressions (CS) of the encoding algorithm. Similar circuit structures are integrated through a two-layer architecture, which further reduces logical resources. For the special size of the generator's matrix, we introduce a preprocessing method. In addition, configuration registers are used to replace the control unit. Implemented and verified on FPGA, the proposed architecture achieves a throughput of 4.69 Gbps using only 1658 LUTs and 1038 FFs. Compared with the previous architectures, this architecture achieves lower resource utilization and multi-Gbps throughput.**

***Index Terms*—Encoder, CCSDS, LDPC codes, FPGA.**

I. Introduction

LOW-DENSITY parity-check (LDPC) codes, invented by Gallager in 1961 [1], have effective parallel structures. The excellent error-correction performance of this channel coding technique approximates Shannon's theoretical limit. Although the sparse parity-check matrices of Mackey [2] have excellent performance, they are random.

Until QC- LDPC codes were proposed by Tanner *et al.* [3], whose parity check matrix was composed of several circular shift sub-matrices. The regular structure of QC-LDPC reduced the complexity of hardware implementation. QC-LDPC codes have also become one of the primary choices for Very Large Scale Integration (VLSI) implementation of LDPC codes. These codes have been adopted by many communication standards, such as 802.11n, DVB-S2, Flash memory storage, etc.

In Deep-Space and Near-Earth communication, QC-LDPC codes was also adopted by CCSDS. For Near-Earth applications, high data rates and excellent reliability are important requirements, the encoder architectures are challenged. Reference [4] proposes an architecture based on the R-U method, which entirely removes the forward substitution operation. Reference [5] exhibits the advantages of shift-register-add-accumulator (SRAA) and a parallel Recursive Convolutional Encoder (RCE) in general encoders, these architectures have strong flexibility. Reference [6] shows an encoder architecture for subsequent compression of involved matrix based on factorization, different codes and multi-rate codes were accepted. However, the above structure is difficult for C2 code to achieve optimal area resources and throughput. Many dedicated encoders for C2 code are proposed in [7]–[9], a large number of decoders for QC-LDPC are also shown in [10]–[12]. According to the structural characteristics of C2 code, they reconstruct the generator polynomial to save resources.

A novel architecture was proposed in this brief, which effectively reduces logical resources and uses the fewest registers.

Our Contributions

1) A novel two-layer architecture is proposed, which can effectively merge similar logic circuits.
2) A method is introduced to handle the inconvenience caused by the special code rate of C2 code.
3) A simplified circuit for input control unit is proposed, which effectively avoids counters and finite state machines, and reduces resources at the same time.

The remainder of this brief is organized as follows: CCSDS C2 code and encoding algorithm are described in Section II, integrated architecture and two-layer architecture are proposed in Section III. The Stream input and control method are exhibited in Section IV, hardware implementations are presented in Section V. Finally, we draw the conclusions in Section VI.

II. CCSDS C2 Code

In Near-Earth standard, a basic LDPC code (C2) with dimensions (8176, 7154) is defined by CCSDS and its code rate is 0.875. The code (H) is composed of 2×16 cyclic sub-matrices, the size of which are 511×511. The first row of the sub-matrix is obtained by the positions of the two ones

Manuscript received December 20, 2020; revised January 6, 2021; accepted January 22, 2021. Date of publication January 26, 2021; date of current version June 29, 2021. This work was supported in part by the Important Project of the National Natural Science Foundation of China under Grant 62090012, Grant 62031016, and Grant 61831017; and in part by the Sichuan Provincial Science and Technology Important Projects under Grant 2019YFG0498, Grant 2020YFG0282, Grant 2020YFG0452, and Grant 2020YFG0028. This brief was recommended by Associate Editor C. W. Sham. *(Corresponding author: Quanyuan Feng.)*

The authors are with the Institute of Microelectronics, Southwest Jiaotong University, Chengdu 611756, China (e-mail: fengquanyuan@163.com).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSII.2021.3054334.

Digital Object Identifier 10.1109/TCSII.2021.3054334

in the first row, and the row vector expands the entire cyclic sub-matrix by cyclic right-shift.

$$H = \begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} & \cdots & A_{1,16} \\ A_{2,1} & A_{2,2} & A_{2,3} & \cdots & A_{2,16} \end{bmatrix} \tag{1}$$

At present, many encoding algorithms have been proposed, such as direct method, L-U method, Partitioned H Method [13], piecewise systematic encoding [14] and R-U method. Although R-U and Partitioned H Methods have good effects in pipeline operations, the direct method is the best choice for C2 code [15]. In the direct method, the parity bit sequence p only needs to calculate the multiplication $p = sB$ of the bit vector s with the matrix B. The vector p is 1022 bits, the matrix B is composed of 14×2 cyclic sub-matrices, the size of which are 511×511.

$$B = \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \\ & \vdots \\ B_{14,1} & B_{14,2} \end{bmatrix} \tag{2}$$

In the general case, the bit vector s is divided into 14 blocks, and each block contains 511 consecutive bits, as shown in (3) and (4). Product operations are performed on 14 blocks respectively, the calculation results (P_i) of 14 blocks need to be summed to get the parity bit sequence p, as shown in (5).

$$s = [s_1 \ s_2 \ s_3 \ \cdots \ s_{14}] \tag{3}$$

$$s_i = [s_{i,1} \ s_{i,2} \ s_{i,3} \ \cdots \ s_{i,511}] \tag{4}$$

$$p = \sum_{i=1}^{14} P_i = \sum_{i=1}^{14} (s_i \ [B_{i,1} \ B_{i,2}]) \tag{5}$$

In the above algorithm, p is the sum of P_i, and each P_i is a set of mutually independent operations, then 14 sets of P_i are effectively calculated in parallel. The direct method can give full play to the advantages of parallel computing, which is suitable for hardware implementation because of its parallelism.

III. Proposed Optimization Strategy

A. Basic Architecture

According to Section II, the algorithm is reconstructed to adapt to the cyclic structure. We define a matrix b, which consists of the first rows of the sub-matrices in B. The first rows ($B_{i,1,f}, B_{i,2,f}$) of the sub-matrices $[B_{i,1} \ B_{i,2}]$ are b_i, and the vectors b_i contains from $b_{i,1}$ to $b_{i,1022}$, as shown in (6) and (7).

$$b = \begin{bmatrix} B_{1,1,f} & B_{1,2,f} \\ B_{2,1,f} & B_{2,2,f} \\ & \vdots \\ B_{14,1,f} & B_{14,2,f} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{14} \end{bmatrix} \tag{6}$$

$$b_i = [b_{i,1} \quad b_{i,2} \quad b_{i,3} \quad \cdots \quad b_{i,1022}] \tag{7}$$

At the same time, we define circ(A, x) as the cyclic right-shift function, A is the vector (1×511) to be shifted, and x($x > 0$) is the number of cyclic shifts by one bit, as shown in (8). The expression of the parity bit sequence p can be updated to (10).

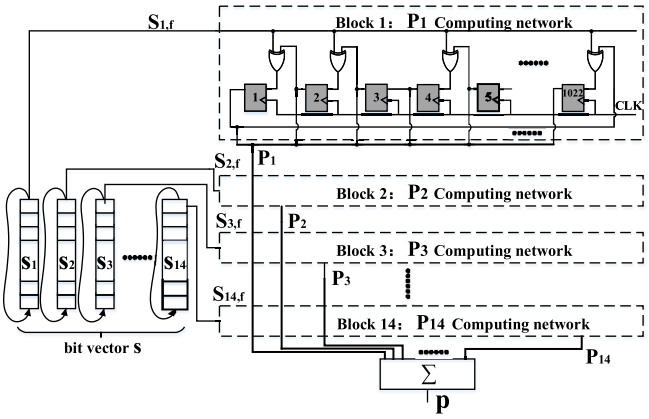

Fig. 1. Basic architecture based on (10) and (11), each block uses a single input ($s_{i,f}$), $s_{i,f}$ is the first bit of s_i.

TABLE I
Resources and Performance Estimations

Index	Value
2-input XOR gates	20284
Flip-Flops	14308
Encoding cycles	511

$$circ(\mathrm{A}, x) = [A_{512-x} \ \cdots \ A_1 \ \cdots \ A_{511-x}] \tag{8}$$

$$circ(b_i, x) = [circ(B_{i,1,f}, \ x) \ circ(B_{i,2,f}, x)] \tag{9}$$

$$p = \sum_{i=1}^{14} \left(\sum_{j=1}^{511} (s_{i,j} \times circ(b_i, j)) \right) \tag{10}$$

The shift registers are used to continuously shift the temporary value of p to the left. The effect of this circular left-shift is defined as p_{circ}, which is equivalent to shifting b_i circularly to the right. After 511 left shift operations, the expression of p_{circ} can be represented by (11).

$$p_{circ} = \sum_{i=1}^{14} \left(\sum_{j=1}^{511} (s_{i,j} \times b_i) \right) \tag{11}$$

Based on the above algorithm, the hardware implementation result is shown in Fig. 1. According to the bit vector s, 14 parallel blocks are divided to calculate p. In each parallel block, 511 shift accumulations are performed in the shift registers. Due to the structure of the cyclic right-shift sub-matrix, the shift registers update the previous temporary data by summing with the previous input data. In this way, the right shift of the logic circuit was simulated by the shift registers that cyclically shift to the left. The first row of the sub-matrix is used to determine the logic in the block circuit. When the element of the first row is 1, the XOR gate is used for summation, otherwise, two registers are directly connected. For the entire architecture, the p_i of each block obtained by parallel calculation, and then each p_i is summed through the large bit-width adder to obtain the final output p.

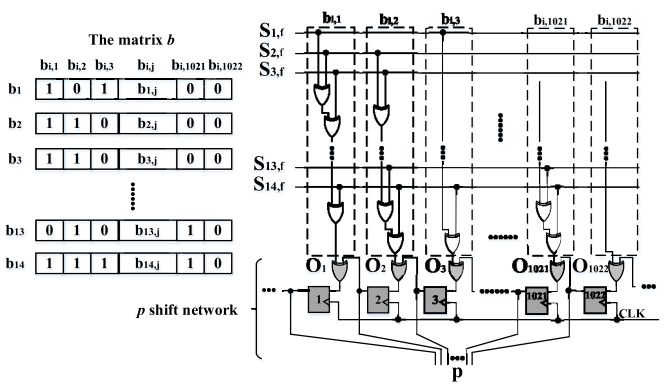

Fig. 2. Proposed integrated architecture based on (12) and the structure of the matrix b.

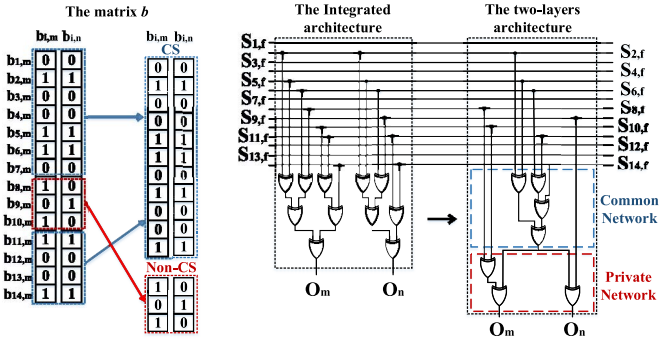

Fig. 3. Proposed two-layer architecture based on (14) and (15), when $8 \leq i \leq 10$, $b_{i,m} \neq b_{i,n}$, otherwise, $b_{i,m} = b_{i,n}$.

Table I shows the resources and performance evaluation of the above architecture. Such an architecture requires 20284 two-input XOR gates and 14308 registers. For the parallel block structure, more registers are used because each block is independent of each other. Each block has a different number of 2-input XOR gates, which has 500 2-input XOR gates and 1022 registers on average. The large bit-width adder also has a large number of logics (13286 2-input XOR gates).

B. Register Optimization

In the general case, when the input bit vectors s_1 to s_{14} are updated asynchronously, each block often needs 1022 registers to temporarily store data, which ensure that each bit of the final summation has a one-to-one correspondence. However, when the input bit vectors s_1 to s_{14} are updated synchronously, the highly parallel block circuits can be combined. An ingenious method is adopted to reduce the registers, as shown in (12).

$$p_{circ} = [p_1\ p_2\ p_3\ \cdots\ p_{1022}] = \sum_{j=1}^{511}\left(\sum_{i=1}^{14}(s_{i,j} \times b_i)\right) \quad (12)$$

It must be noted that (12) is the same as (11), but the order of summation is different. Therefore, the 14 blocks can be summed first, and then 511 cyclic shifts can be performed. A lot of registers are reduced to temporarily store operation results.

According to (12), we reconstruct the architecture, as shown in Fig. 2. When s_i is updated synchronously, we adopt an integrated structure instead of a block structure. In the previous scheme, the results of each block were calculated and temporarily stored, and the sum was finally calculated. In the scheme in Fig. 2, the sum of all blocks is directly calculated and stored. Compared with the basic architecture, the integrated architecture reduces a lot of registers, and all the registers are in p shift network. Since each block is no longer summed in the integrated structure, this architecture does not rely on the large bit-width adder, effectively reducing the 2-input XOR gates.

C. Logic Resource Optimizations

In the integrated structure, there are a lot of the same logic in each part of the circuit from $b_{i,1}$ to $b_{i,1022}$, and they have the same input, so a solution is proposed to remove the same logics. We assume that the same circuit exists in $b_{i,m}$ and $b_{i,n}$, and the output results (O_m and O_n) of the logic circuit are shown in (13).

TABLE II
THE DISTRIBUTION OF COMMON SUBEXPRESSIONS IN C2 CODE

Number of CS	14	13	12	11	10
Number of groups	26	225	208	40	8

For O_m and O_n, they have the same product term $s_{i,f}$. We assume that there is a constant j ($1 \leq j \leq 14$), when $b_{j,m} = b_{j,n}$, the product term $b_{j,m(n)} \times s_{j,f}$ is the one of common subexpressions (CS) of O_m and O_n, as shown in (14). O_m and O_n have at most 14 common subexpressions, when $b_{i,m} = b_{i,n} (1 \leq i \leq 14)$. The i of the common subexpressions need not be continuous. When $b_{j,m} \neq b_{j,n}$, the product term $b_{j,m(n)} \times s_{j,f}$ is the one of non-common subexpressions (non-CS) of O_m and O_n, as shown in (15).

$$CS = \sum_{b_{i,m}=b_{i,n}} \left(b_{i,m(n)} \times s_{i,f}\right) \quad (13)$$

$$O_{m(n)} = CS + \sum_{b_{i,m}\neq b_{i,n}} \left(b_{i,m(n)} \times s_{i,f}\right) \quad (14)$$

We assume that when $8 \leq i \leq 10$, $b_{i,m} \neq b_{i,n}$, otherwise, $b_{i,m} = b_{i,n}$. According to (14) and (15), we divide the logic circuit of b_i into two layers, as shown in Fig. 3. One layer is called the common network, which is the hardware implementation of the common subexpressions, and the other layer is called the private network, which is designed for non-common subexpressions. By sharing the same calculation circuit, the repetitive structure in the circuit is reduced.

IV. Proposed Encoding Architecture

In this section, we apply the optimization method proposed in the previous section to C2 code. An effective preprocessing method is proposed to solve the problem that the input width is not a multiple of 8 or 16 at a code rate of 7/8. At the same time, the configuration register is used instead of the input control unit, reducing the resources of the input control module.

TABLE III
CONTROL INSTRUCTIONS AND FUNCTIONS

Control Instruction [C_0 C_1]	00	01	10	11
Function	Idle	Wait	Calculation	Output valid

Algorithm 1 Grouping Algorithm

Initialization:
$O \ni \{O_1, O_2, \cdots\cdots, O_{1022}\}$
Grouping:
for i = 14 **to** 1 **do**
  for j = 1 **to** 1022 **do**
    for k = 1 **to** 1022 **do**
      if ($O_j \in O$ **and** CS of O_j and O_k = i **and** j≠k) **then**
        O_j and O_k are grouped together
        O_j and O_k are removed from O ($O_{j(k)} \notin O$)
      end if
    end for
  end for
end for

A. Two-Layer Architecture

According to Section III, more CS needs to be found in the C2 code. When the number of CS are larger, the optimization of the logic circuit is better. A grouping algorithm is used for grouping, 1022 logic outputs (O_1 to O_{1022}) are divided into 511 groups, each group has 2 logic outputs (Algorithm 1).

The O_j and O_k with the largest number of CS will be defined as a group. After the algorithm is executed, more than 99.2% (507/511) of the groups have CS between 10 and 14, but the remaining (4/511) groups have poor results ($\mathrm{CS}\{O_{784}, O_{955}\} = 9$, $\mathrm{CS}\{O_{503}, O_{901}\} = 9$, $\mathrm{CS}\{O_{770}, O_{991}\} = 8$, $\mathrm{CS}\{O_{966}, O_{976}\} = 7$).

Therefore, we split the groups with the number of CS less than 10 and reorganize them into other assigned groups. The new groups formed by this method have 3 (original 2 + reorganized 1) logic outputs, which ensure that the number of CS is greater than 10 in the 3 logic outputs ($\mathrm{CS}\{O_{48}, O_{49}, \mathbf{O_{911}}\} = 12$, $\mathrm{CS}\{O_{48}, O_{49}, \mathbf{O_{911}}\} = 12$, $\mathrm{CS}\{O_{162}, O_{818}, \mathbf{O_{966}}\} = 12$, $\mathrm{CS}\{O_{345}, O_{527}, \mathbf{O_{955}}\} = 11$, $\mathrm{CS}\{O_{387}, O_{832}, \mathbf{O_{530}}\} = 11$, $\mathrm{CS}\{O_{147}, O_{168}, \mathbf{O_{770}}\} = 11$, $\mathrm{CS}\{O_{273}, O_{238}, \mathbf{O_{901}}\} = 11$, $\mathrm{CS}\{O_{306}, O_{399}, \mathbf{O_{784}}\} = 11$, $\mathrm{CS}\{O_{21}, O_{74}, \mathbf{O_{976}}\} = 10$). This method effectively eliminates groups.

B. Pretreatment and Control Unit

In C2 code, there are (14 × 2) QC sub-matrices, which means that the maximum degree of parallelism is 14. For the common interface width and bus width, the degree of parallelism is a challenge, which is not an integer power of 2. In this brief, we propose a method to preprocess the input sequence. By dividing the input vector into 14 parts, and the input vector $\{s_{1,f}, s_{2,f} \cdots s_{14,f}\}$ is composed of each part in turn. C_0 and C_1 are defined as control instructions, which occupy 2 bits. These two parts effectively form the 16-bit input, which inserts 2-bit instructions into the 14-bit input vector. And 4 instructions are defined in this architecture.

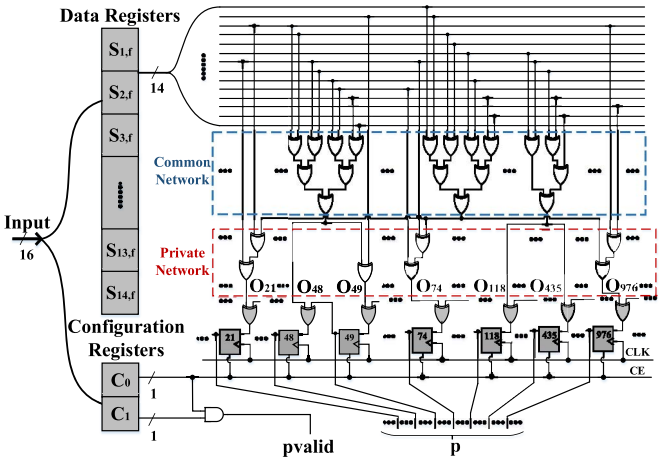

Fig. 4. The overall architecture of the LDPC encoder, the figure shows a two-layer architecture with 3 groups as an example.

TABLE IV
C2 CODE ESTIMATIONS

Work	Logic functions	Flip-Flops	Encoding cycles
[8]	29638	1107	448
[13]	7602	8176	510
Proposed (Fig. 2)	6998	1038	511
Proposed (Fig. 4)	4896	1038	511

The proposed method adds additional instructions to each set of input vectors, the control instructions are placed at the end of the input to replace the control unit. The architecture does not use complex control structures such as counters, effectively reducing resources in control unit. This method is shown in Table III. When $C_0 = 1$ and $C_1 = 0$, the input vector at the same time will be calculated. When C_0 and C_1 are both 1, the output is the final result, so the C_0 and C_1 of the last group of inputs must be 1. When $C_0 = 0$ and $C_1 = 1$, it means that there is still an encoding process to be completed, and it is waiting for a valid input vector.

Since the input registers are controlled under the same clock, the input vector $s_{i,f}$ must be changed synchronously with the control instruction. When [C_0 C_1] are changed to 10 and 11, the input vector to be encoded must be changed synchronously with the control instruction, which ensures the correct timing. When [C_0 C_1] are changed to 00 and 01, no matter the synchronized input vector is any value, it will not be calculated.

C. Overall Architecture of the LDPC Encoder

The overall architecture of the proposed LDPC encoder is shown in Fig. 4. The 16 registers are directly assigned by the 16-bit input, which is divided into 14 data registers and 2 configuration registers. The data registers are connected to the parity calculation circuit, and the calculation circuit is divided into a common network and a private network. In Fig. 4, taking 3 groups as an example, the 7 logic outputs contained in the 3 groups. For the configuration registers, when C_0 is 1, all shift

TABLE V
C2 HARDWARE IMPLEMENTATIONS

Work	FPGA technology	Flip-Flops	LUTs	BRAMs	Encoding cycles	Clock (MHz)	Throughput (Gbps)
[7]	Kintex7/XC7K325T	92233	54747	38	180	297	2.97
[13]	Virtex-5/XC5VLX110T-1	1340	3338	0	510	260	4.16
[16]	Virtex-5/XC5VLX110T-1	1156	9128	0	N/A	200	3.12
[17]	Artix-7/100T-1	3219	6873	1	N/A	239	1.55
[18]	Virtex-5/XC5VLX30T-1	9.2K	N/A	N/A	1020	164	1.14
Proposed (Fig. 4)	Virtex-5/XC5VLX110T-1	1038	1658	0	511	335	4.69

registers are driven to perform cyclic calculations, otherwise, all inputs maintain the current value, and the shift register is no longer triggered. *pvalid* is the product of C_0 and C_1 in the 2-element domain to determine whether the output p is valid.

Table IV compares the encoder based on Fig. 4 and Fig. 2 with existing work targeting C2 code. A large number of logic resources are used in [8], which effectively reduces a lot of flip-flops. The encoding cycles are reduced by the packer-unpacker technology. Reference [13] is optimized for logical resources utilization, but the number of flip-flops is much larger than [8]. The proposed architecture based on Fig. 2 effectively reduces logical resources. Compared with [8], the logics are reduced by 76%. Compared with [13], the flip-flops are reduced by 87%. Compared with the proposed architecture based on Fig. 2, the proposed architecture based on Fig. 4 has approximately 30% less logical resources.

V. IMPLEMENTATION RESULT

The entire architecture based on Fig. 4 is implemented on FPGA, verified and validated on Virtex-5/XC5VLX110T-1 hardware. Table V shows the performance estimates of the proposed architecture based on Fig. 4 and the existing architecture based on C2 code. In [7], the entire input vector is cached in the registers, and the 7154 input frames are calculated in parallel. This architecture means that more resources are required. Reference [13] proposes a variety of architectures for all CCSDS standards, our architecture is based on C2 code, and compares the architecture based on C2 code in [13]. Reference [16] adopted a ping-pong buffer at the encoder input, and both methods have fewer registers than [7]. Reference [17] and [18] are mature products that have been commercialized. Block RAM is used in [17] as a buffer for input and output. Two circulant tables are used in [18], these architectures require a lot of resources.

The proposed encoder architecture based on Fig. 4 uses 1038 flip-flops and 1658 LUTs. In this brief, the highest clock frequency is 335 MHz, and the throughput is 4.69 Gbps. The entire encoding process only takes 511 clock cycles. Compared to the above architectures, the proposed architecture based on Fig. 4 has a lower resource utilization and higher throughput.

VI. CONCLUSION

In this brief, we introduce a novel encoding architecture for the CCSDS Near-Earth standard. By extracting common subexpressions to merge similar circuits, this method is suitable for highly parallel matrix multiplication networks that have the same product terms.

REFERENCES

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

[2] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inf. Theory*, vol. 45, no. 2, pp. 399–431, Mar. 1999.

[3] R. M. Tanner, D. Sridhara, A. Sridharan, T. E. Fuja, and D. J. Costello, "LDPC block and convolutional codes based on circulant matrices," *IEEE Trans. Inf. Theory*, vol. 50, no. 12, pp. 2966–2984, Dec. 2004.

[4] 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.

[5] J. Xie, S. Li, Y. Chen, Q. Zhang, and X. Zeng, "High throughput multi-code LDPC encoder for CCSDS standard," in *Proc. ASICON*, Chongqing, China, 2019, pp. 1–4.

[6] 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.

[7] W. Ren and H. Liu, "The design and implementation of high-speed codec based on FPGA," in *Proc. ICCSN*, Chengdu, China, 2018, pp. 427–532.

[8] 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.

[9] J. Wang, S. Yuan, Y. Zhou, and G. Zhang, "Codec implementation of QC-LDPC code in CCSDS near-earth standard," in *Proc. ICCCS*, Shanghai, China, 2020, pp. 575–579.

[10] 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.

[11] Q. Lu, J. Fan, C.-W. Sham, W. M. Tam, and F. C. M. Lau, "A 3.0 Gb/s throughput hardware-efficient decoder for cyclically-coupled QC-LDPC codes," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 63, no. 1, pp. 134–145, Jan. 2016.

[12] Z. Zhong, Y. Huang, Z. Zhang, X. You, and C. Zhang, "A flexible and high parallel permutation network for 5G LDPC decoders," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 12, pp. 3018–3022, Dec. 2020.

[13] D. Theodoropoulos, N. Kranitis, A. Tsigkanos, and A. Paschalis, "Efficient architectures for multigigabit CCSDS LDPC encoders," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 5, pp. 1118–1127, May 2020.

[14] C.-F. Tseng and J.-H. Tarng, "Low-complexity and piecewise systematic encoding of non-full-rank QC-LDPC codes," *IEEE Commun. Lett.*, vol. 19, no. 6, pp. 897–900, Jun. 2015.

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

[16] D. Theodoropoulos, N. Kranitis and A. Paschalis, "An efficient LDPC encoder architecture for space applications," in *Proc. IOLTS*, Sant Feliu de Guixols, Spain, 2016, pp. 149–154.

[17] *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

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