electronics

MDPI

Article

Research and Implementation of High-Efficiency and Low-Complexity LDPC Coding Algorithm

Xiong Liao [1,2], Junxiong Guo [1,*], Zhenghua Luo [1,2,*], Yanghui Xu [3] and Yingjun Chu [4]

[1] School of Electronic Information and Electrical Engineering, Chengdu University, Chengdu 610106, China; liaoxiong@cdu.edu.cn
[2] Sichuan Time Frequency Synchronization System and Application Engineering Technology Research Center, Chengdu 610106, China
[3] Chengdu Jinjiang Electronic System Engineering Co., Ltd., Chengdu 610051, China; jecdz@jec784.com
[4] The Fifth Research Institute of Telecommunications Science and Technology Co., Ltd., Chengdu 610021, China; chuyj@5ritt.com
* Correspondence: guojunxiong@cdu.edu.cn (J.G.); luozhenghua@cdu.edu.cn (Z.L.)

Abstract: In this work, we proposed a high-efficiency and low-complexity encoding algorithm and its corresponding implementation structure during the design and implementation process of an LDPC encoder and decoder. This proposal was derived from extensive research on and analysis of standard encoding algorithms and recursive iterative encoding algorithms, specifically targeting the problem of high computational complexity in encoding algorithms. Subsequently, we combined binary phase-shift keying modulation mode and additive white Gaussian noise channel transmission with the min-sum decoding algorithm to realize the (1536, 1024) LDPC codec. This codec was uniformly quantized with a (6, 2) configuration, executed eight iterations, and achieved a 2/3 code rate in the IEEE802.16e standard. At the bit error rate (BER) of 10^{-5}, the codec's BER obtained by the proposed coding algorithm was about 0.25 dB lower than the recursive-iterative coding algorithm and was about 1.25 dB lower than the standard coding algorithm, which confirms the correctness, effectiveness, and feasibility of the proposed algorithm.

Keywords: LDPC code; high-efficiency and low-complexity coding algorithm; MSA; IEEE802.16e

Citation: Liao, X.; Guo, J.; Luo, Z.; Xu, Y.; Chu, Y. Research and Implementation of High-Efficiency and Low-Complexity LDPC Coding Algorithm. *Electronics* **2023**, *12*, 3696. https://doi.org/10.3390/electronics12173696

Academic Editors: Kajetana Snopek, Wojciech Wojtasiak and Grzegorz Pastuszak

Received: 26 July 2023
Revised: 25 August 2023
Accepted: 29 August 2023
Published: 1 September 2023

1. Introduction

LDPC codes, which are linear block codes characterized by sparse matrices, offer performance that approaches the Shannon limit. They possess remarkable flexibility and low error leveling, and exhibit low decoding complexity. LDPC codes have a highly adatable structure that allows for complete parallel operations. Their hardware implementation complexity is also low, enabling high throughput. LDPC codes also hold the potential for high-speed decoding and demonstrate excellent error correction performance in sudden situations. Notably, LDPC codes do not require correlation during encoding and decoding processes. A single LDPC code can be widely utilized across various channels and has undergone rigorous theoretical analysis to ensure its verifiability. In comparison to Turbo codes, LDPC codes exhibit superior performance in terms of good distance, low complexity, and high parallel decoding methods [1].

Initially proposed by Gallager in 1962, LDPC codes, as a superior-performance linear block codes, did not garner significant research attention due to restricted research and implementation conditions at the time. It was only in 1993, when Berrou et al. [2] developed the Turbo codes and achieved significant research results, that the LDPC codes were reintroduced to researchers. In 1996, Mackey et al. [2] conducted in-depth investigations and analysis on LDPC codes, revealing their superior coding performance, which even surpassed that of Turbo codes, particularly in scenarios involving long code lengths [3].

The check matrix of LDPC codes is characterized by sparsity, allowing them to overcome the enormous decoding operations in the case of long block codes. Consequently, efficient decoding algorithms can be applied. Additionally, LDPC codes integrate flexible structures, low decoding complexity, complete concurrent operation, and straightforward hardware implementation [4]. This paper, motivated by the advantages of LDPC codes and aiming to address the high computational complexity of standard coding algorithms, proposes a high-efficiency and low-complexity coding algorithm. Unlike traditional rapid coding algorithms, our proposed approach is achieved by analyzing and studying the check matrices. Experimental evaluation demonstrates that the developed algorithm effectively reduces implementation complexity while ensuring desirable coding performance.

2. LDPC Code in the IEEE802.16e Standard

The LDPC code in the IEEE802.16e standard [5] is obtained by the quasi-cyclic constructive method, affording high coding performance, low computational complexity, and easy hardware implementation [6–9]. The LDPC codes with the same code rate but different lengths have consistently structured check matrices that extend the basic check matrix via a square matrix [4]. The basic check matrix of the LDPC code is denoted as H_b of dimension $m_b \times n_b$. The element in H_b corresponds to the submatrix of the check matrix H (H could be obtained only by updating and extending the elements in H_b). The column number of H_b is fixed to 24, i.e., $n_b = 24$. For the standard LDPC code with a 2/3 code rate, the row number of H_b is eight, i.e., $m_b = 8$. The basic check matrix H_b of the LDPC code in the standard form is:

$$H_b = \begin{bmatrix} H_{b1} & H_{b2} \end{bmatrix} \tag{1}$$

In Equation (1), H_{b1} size is $m_b \times k_b (k_b = n_b - m_b)$ and H_{b2} is $m_b \times m_b$ [8].

In submatrices H_{b1} and H_{b2} of H_b, the element values are -1 or a non-negative integer. In the updating and extension of H_b towards H, the elements with a -1 value are replaced by $z \times z$ zero matrices, and the elements with non-negative integer values are replaced by the square matrix after the cyclic shift of a $z \times z$ unit matrix [8]. z is the extension factor $z = n/n_b$, n denotes the length of the LDPC code, and n_b is the column length of H_b. The submatrix H_{b2} has the following structural characteristics:

In the first column of H_{b2}, the values of $h(1)$, $h(r)$, and $h(m_b)$ are non-negative integers, $h(1) = h(m_b)$, and $r \in [2, m_b - 1]$. Except for the first row, tail row, and row r, the element values at the other positions are -1. Apart from the first column, the other parts constitute a quasi-double-diagonal structure (Equation (2)). The basic check matrix of the LDPC code with a code rate of 2/3 in the standard form is displayed in Equation (3).

$$\boldsymbol{H_{b2}} = \begin{bmatrix} \boldsymbol{h(1)} & 0 & & & & & & & \\ -1 & 0 & 0 & & & & & & \\ \vdots & & 0 & 0 & & & -1 & & \\ -1 & & & 0 & \ddots & & & & \\ \boldsymbol{h(r)} & & & & \ddots & \ddots & & & \\ -1 & & & & & \ddots & 0 & & \\ \vdots & & & -1 & & & 0 & 0 & \\ -1 & & & & & & & 0 & 0 \\ \boldsymbol{h(m_b)} & & & & & & & & 0 \end{bmatrix} \tag{2}$$

$$\boldsymbol{H_b} = \begin{bmatrix} 3 & 0 & -1 & -1 & 2 & 0 & -1 & 3 & 7 & -1 & 1 & 1 & -1 & -1 & -1 & -1 & 1 & 0 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & 1 & -1 & 36 & -1 & -1 & 34 & 10 & -1 & -1 & 18 & 2 & -1 & 3 & 0 & -1 & 0 & 0 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & 12 & 2 & -1 & 15 & -1 & 40 & -1 & 3 & -1 & 15 & -1 & 2 & 13 & -1 & -1 & -1 & 0 & 0 & -1 & -1 & -1 & -1 \\ -1 & -1 & 19 & 24 & -1 & 3 & 0 & -1 & 6 & -1 & 17 & -1 & -1 & -1 & 8 & 39 & -1 & -1 & -1 & 0 & 0 & -1 & -1 & -1 \\ 20 & -1 & 6 & -1 & -1 & 10 & 29 & -1 & -1 & 28 & -1 & 14 & -1 & 38 & -1 & -1 & 0 & -1 & -1 & -1 & 0 & 0 & -1 & -1 \\ -1 & -1 & 10 & -1 & 28 & 20 & -1 & -1 & 8 & -1 & 36 & -1 & 9 & -1 & 21 & 45 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 \\ 35 & 25 & -1 & 37 & -1 & 21 & -1 & -1 & 5 & -1 & -1 & 0 & -1 & 4 & 20 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 \\ -1 & 6 & 6 & -1 & -1 & -1 & 4 & -1 & 14 & 30 & -1 & 3 & 36 & -1 & 14 & -1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 \end{bmatrix} \quad (3)$$

3. LDPC Coding Algorithm

3.1. Standard Coding Algorithm

The check matrix of the LDPC code is set as H and has a dimension of $m \times n$. The information bit length, the information codeword (CW) vector, and the post-coding CW vector are k, u, and c, respectively [5–9]. After Gaussian elimination, the check matrix H is presented as:

$$H = \begin{bmatrix} P_{m\times k} & I_{m\times m} \end{bmatrix} \quad (4)$$

The following generator matrix can be obtained according to the check constraint equation $G \cdot H^T = 0$:

$$G = \begin{bmatrix} I_{k\times k} & P^T{}_{k\times m} \end{bmatrix} \quad (5)$$

When u is known and c can be obtained through $c = u \cdot G$, thus completing the coding process of the input information.

The code rate is set as $R = k/n$, and the following can be obtained when one frame of code is generated:

The number of multiplication operations is:

$$k \cdot n = R \cdot n^2$$

The number of additive operations is:

$$(k-1) \cdot n = R \cdot (n - 1/(2R))^2 - 1/(4R)$$

The computational complexity of the standard coding algorithm is $O(n^2)$. When the information code involves tens of thousands of bits, the coding complexity will significantly affect implementation. Nevertheless, standard LDPC coding has been rarely used [9–13].

3.2. Recursive-Iterative Coding Algorithm

The recursive-iterative coding algorithm uses a quasi-double-diagonal structure of the check matrix, simplifying the coding operations to a certain extent [5,14–17]. The check bit CW generated in the coding process is set as m, the check vector as p, the post-coding CW length as n, and the CW vector as c.Z_q is a $z \times z$ square matrix, selected as a zero matrix ($q = -1$) or a square matrix (q is a non-negative integer) after the cyclic rightward shift of a unit matrix. The algorithm flow is described as follows:

① u and p are segmented according to the length z:

$$u = [u_1^T \quad u_2^T \quad \cdots \quad u_{k_b}^T] \quad (6)$$

$$p = [p_1^T \quad p_2^T \quad \cdots \quad p_{m_b}^T] \quad (7)$$

In Equations (6) and (7):

$$u_i = [u((i-1)z+1)\ u((i-1)z+2\ \cdots\ u(iz)]^T, i = 1, 2, \cdots, k_b \quad (8)$$

$$p_i = [p((i-1)z+1)\ p((i-1)z+2\ \cdots\ p(iz)]^T, i = 1, 2, \cdots, m_b \quad (9)$$

② All u_i and p_i are longitudinally spliced to obtain:

$$\begin{bmatrix} u_1 & u_2 & \cdots & u_{k_b} \end{bmatrix}^T = u^T \tag{10}$$

$$\begin{bmatrix} p_1 & p_2 & \cdots & p_{m_b} \end{bmatrix}^T = p^T \tag{11}$$

Each sub-vector p_i of the check vector p is obtained through the coding process, after which the following are acquired:

$$c = \begin{bmatrix} u & p \end{bmatrix} \tag{12}$$

③ $H_2 \cdot p^T = H_1 \cdot u^T$ is obtained according to the check equation $H \cdot c^T = 0$. The element located at position (i, j) in the sub-matrix H_{b1} of the basic check matrix H_b is denoted as $H_{b1}(i, j)$. According to the forms of Z_q and H_{b2}, the following are calculated:

$$\begin{bmatrix} \boldsymbol{Z}_{h(1)} & \boldsymbol{Z}_0 & & & & & & & \\ \boldsymbol{Z}_{-1} & \boldsymbol{Z}_0 & \boldsymbol{Z}_0 & & & & & & \\ \vdots & & \boldsymbol{Z}_0 & \boldsymbol{Z}_0 & & & \boldsymbol{Z}_{-1} & & \\ \boldsymbol{Z}_{-1} & & & \boldsymbol{Z}_0 & \ddots & & & & \\ \boldsymbol{Z}_{h(r)} & & & & \ddots & \ddots & & & \\ \boldsymbol{Z}_{-1} & & & & & \ddots & \boldsymbol{Z}_0 & & \\ \vdots & & & \boldsymbol{Z}_{-1} & & & \boldsymbol{Z}_0 & \boldsymbol{Z}_0 & \\ \boldsymbol{Z}_{-1} & & & & & & & \boldsymbol{Z}_0 & \boldsymbol{Z}_0 \\ \boldsymbol{Z}_{h(m_b)} & & & & & & & & \boldsymbol{Z}_0 \end{bmatrix} \begin{bmatrix} \boldsymbol{P}_1 \\ \boldsymbol{P}_2 \\ \vdots \\ \boldsymbol{P}_{m_b} \end{bmatrix} = \begin{bmatrix} \boldsymbol{Z}_{H_{b1}(1,1)} & \boldsymbol{Z}_{H_{b1}(1,2)} & \cdots & \boldsymbol{Z}_{H_{b1}(1,k_b)} \\ \boldsymbol{Z}_{H_{b1}(2,1)} & \boldsymbol{Z}_{H_{b1}(2,2)} & \cdots & \boldsymbol{Z}_{H_{b1}(2,k_b)} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{Z}_{H_{b1}(m_b,1)} & \boldsymbol{Z}_{H_{b1}(m_b,2)} & \cdots & \boldsymbol{Z}_{H_{b1}(m_b,k_b)} \end{bmatrix} \begin{bmatrix} \boldsymbol{u}_1 \\ \boldsymbol{u}_2 \\ \vdots \\ \boldsymbol{u}_{k_b} \end{bmatrix} \tag{13}$$

First, the equation set of Equation (13) is unfolded to obtain m_b equations, which are added to acquire:

$$p_1 = \left(Z_{h(1)} + Z_{h(r)} + Z_{h(m_b)}\right)^{-1} \cdot \sum_{i=1}^{m_b} \sum_{j=1}^{k_b} Z_{H_{b1}(i,j)} \cdot u_j \tag{14}$$

p_1 is then substituted into Equation (13), and from the first equation $\left(Z_{h(1)} \cdot P_1 + Z_0 \cdot P_2 = \sum_{j=1}^{k_b} Z_{H_{b1}(1,j)} \cdot u_j\right)$ we obtain:

$$p_2 = \sum_{j=1}^{k_b} Z_{H_{b1}(1,j)} \cdot u_j + Z_{h(1)} \cdot p_1 \tag{15}$$

p_2 is then substituted into Equation (13), and from the second equation $\left(Z_0 \cdot P_2 + Z_0 \cdot P_3 = \sum_{j=1}^{k_b} Z_{H_{b1}(2,j)} \cdot u_j\right)$ we calculate p_3. The remaining ones can be similarly calculated with the iterative formula of p_i obtained as:

$$p_i = p_{i-1} + \sum_{j=1}^{k_b} Z_{H_{b1}(i-1,j)} \cdot u_j, i = 3, 4, \cdots, r, r+2, r+3, \cdots, m_b \tag{16}$$

$$p_{r+1} = p_r + \sum_{j=1}^{k_b} Z_{H_{b1}(r,j)} \cdot u_j + Z_{h(r)} \cdot p_1 \tag{17}$$

The iterative formula of each sub-vector p_i in the check vector p is obtained through Equations (16) and (17), thus finally obtaining:

$$p = \begin{bmatrix} p_1^T & p_2^T & \cdots & p_{m_b}^T \end{bmatrix} \tag{18}$$

After p is obtained, and since the input u is known, we calculate c from Equation (12), thereby completing the coding operation.

3.3. High-Efficiency and Low-Complexity Coding Algorithm (HE-LC)

When the standard coding algorithm generates CWs via a generator matrix, the deficiency lies in the check matrix's sparsity, which is affected during the Gaussian elimination process. Consequently, the coding complexity is directly proportional to the square of the code length, consuming enormous hardware resources and imposing an excessive time delay [12,13,17,18]. Nevertheless, employing the recursive-iterative coding algorithm allows for the check vector's iterative formula to be obtained through the check matrix's quasi-double-diagonal structure, which aggravates the computational complexity of the check matrix processing by simplifying the coding process.

Based on the research and analysis of the check matrix, we developed a high-efficiency and low-complexity coding algorithm. Specifically, through the row-column replacement, an approximate upper triangular structure (Figure 1) appears in the middle submodule of the check matrix. Next, we partitioned the check matrix so that the upper triangular matrix of the middle submodule forms an independent submatrix. Thereby, efficient iterative coding is evident for this submatrix's special structure, guaranteeing through preprocessing the check matrix's sparsity and enabling iterative coding.

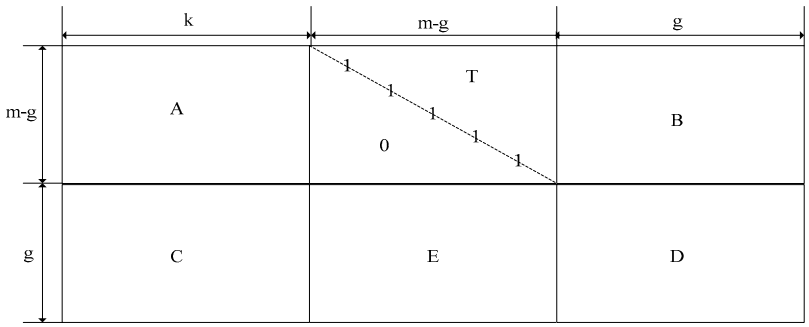

Figure 1. Approximate upper triangular structural form of a check matrix.

In Figure 1, A is an $(m-g) \times k$ matrix, T is an $(m-g) \times (m-g)$ upper triangular matrix, B is an $(m-g) \times g$ matrix, C is a $g \times k$ matrix, E is a $g \times (m-g)$ matrix, and D is a $g \times g$ matrix. The basic check matrix is given as:

$$H_b = \begin{bmatrix} A_{(m_b-g)\times k_b} & T_{(m_b-g)\times(m_b-g)} & B_{(m_b-g)\times g} \\ C_{g\times k_b} & E_{g\times(m_b-g)} & D_{g\times g} \end{bmatrix} \tag{19}$$

In the basic check matrix H_b with a standard code rate of 2/3, we set $n_b = 24$, $m_b = 8$, $k_b = 16$, and $g = 1$.

The post-transformation matrix is obtained by left multiplicating the basic check matrix H_b with a full-rank square matrix $\begin{bmatrix} I_{m-g} & 0 \\ -ET^{-1} & I_g \end{bmatrix}$:

$$\hat{H}_b = \begin{bmatrix} A & T & B \\ -ET^{-1}A + C & 0 & -ET^{-1}B + D \end{bmatrix} \tag{20}$$

In Equation (20), I_{m-g} and I_g represent the unit matrices of size $(m-g) \times (m-g)$ and $g \times g$, respectively.

The CW vector is set to $c = [u \quad p_1 \quad p_2]$, where u is an information bit, and p_1 and p_2 are check bits. The following are obtained through the check equation $H_b \cdot c^T = 0 \Leftrightarrow \hat{H}_b \cdot c^T = 0$:

$$\begin{cases} Au^T + Tp_1^T + Bp_2^T = 0 \\ (-ET^{-1}A + C)u^T + (-ET^{-1}B + D)p_2^T = 0 \end{cases} \tag{21}$$

Namely:

$$p_2^T = -\varphi^{-1}(-ET^{-1}A + C)u^T \tag{22}$$

$$p_1^T = -T^{-1}(Au^T + Bp_2^T) \tag{23}$$

where $\varphi = -ET^{-1}B + D$ is a $g \times g$ full-rank square matrix (to be replaced by a $z \times z$ square matrix).

The structural features of the φ matrix are studied and analyzed, followed by MATLAB-based simulation, revealing that the φ matrix preserved a unit matrix within a binary field. For the LDPC code utilizing a standard code rate of 2/3, the MATLAB simulation results of the φ matrix are illustrated in Figure 2.

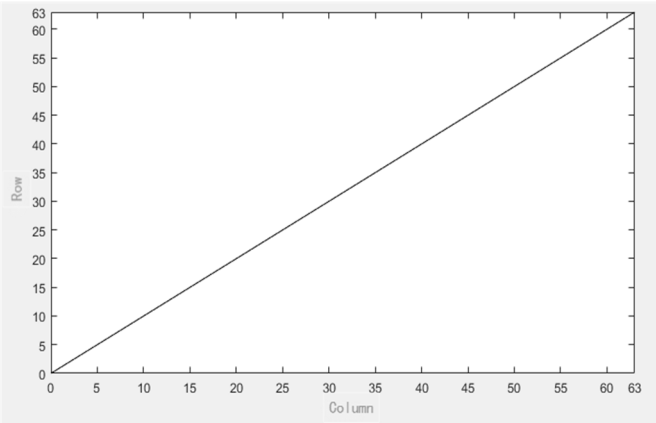

Figure 2. Simulation diagram of the φ matrix generation.

Through comprehensive analysis, the following are obtained:

$$p_2^T = -(-ET^{-1}A + C)u^T \tag{24}$$

$$p_1^T = -T^{-1}(Au^T + Bp_2^T) \tag{25}$$

Adopting the high-efficiency and low-complexity coding algorithm in the coding process, the computational complexity of p_2 and p_1 is $o(n + g^2)$ and $o(n)$, respectively. Hence, the proposed algorithm mitigates the implementation complexity while achieving an efficient coding performance. Moreover, it reduces the coder's hardware needs, facilitating hardware implementation. The implementation flow of the high-efficiency and low-complexity coding algorithm is presented in Table 1.

Table 1. Implementation flow of the high-efficiency and low-complexity coding algorithm.

Step	Computational Formula
Step I	Calculate: $f_1 = Au^T$,$f_2 = Cu^T$
Step II	Calculate: $f_3 = T^{-1}f_1$,$f_4 = Ef_3$
Step III	Calculate: $p_2 = f_4 + f_2$
Step IV	Calculate: $f_5 = Bp_2^T$
Step V	Calculate: $f_6 = f_1 + f_5$
Step VI	Calculate: $p_1 = T^{-1}f_6$

In the coding algorithm process, the maximum computational complexity lies in the first step $f_1 = Au^T$, where the matrix multiplication is solved by performing a cyclic shift operation on the u^T matrix. This results in a linear relationship between the computational complexity and the code length. Using the same method, f_2 and f_5 can be obtained.

For an identity matrix $E_{n\times n}$ and any column vector $A_{n\times 1}$, if we cycle up $A_{n\times 1}$ bit of column vector q to obtain $B_{n\times 1}$, and cycle right $E_{n\times n}$ bit of column vector q to obtain $E'_{n\times n}$, then $E'_{n\times n} \cdot A_{n\times 1} = B_{n\times 1}$.

For the calculation of step 2 $f_3 = T^{-1}f_1$ and step 6 $p_2 = T^{-1}f_6$, if the inverse matrix method is used to solve, it will have a high computational complexity, and it is highly likely to damage the sparsity of the matrix, which is not conducive to hardware implementation. For binary encoding, the forward permutation method can be used to solve this type of operation. In the implementation process, the calculation of $f_3 = T^{-1}f_1$ and $p_2 = T^{-1}f_6$ can be simply achieved using XOR gates. At the same time, the calculation of $f_6 = f_1 + f_5$ and $p_1 = f_4 + f_2$ can also be achieved using XOR gates, and, finally, the complete codeword can be obtained by combining the information sequence and verification sequence in an integer order.

The computational complexity analysis of check bits p_1 and p_2 in efficient and low-complexity encoding algorithms is shown in Tables 2 and 3. From the observation and analysis of Tables 2 and 3, it can be seen that the computational complexity of the improved, efficient, and low-complexity encoding algorithm exhibits a linear relationship with the size of the code-length value.

Table 2. Operational complexity analysis table of p_1.

Operating Steps	Complexity
Au^T	$O(n)$
Cu^T	$O(n)$
$T^{-1}Au^T$	$O(n)$
$ET^{-1}Au^T$	$O(n)$
$(ET^{-1}A + C)u^T$	$O(n)$

Table 3. Operational complexity analysis table of p_2.

Operating Steps	Complexity
Au^T	$O(n)$
Bp_1^T	$O(n)$
$Au^T + Bp_1^T$	$O(n)$
$T^{-1}(Au^T + Bp_1^T)$	$O(n)$

4. LDPC Decoding Algorithm

4.1. Log-Likelihood Ratio-Belief Propagation (LLR–BP) Algorithm

The LDPC soft-decision decoding algorithm with the BP algorithm serving as the basic algorithm affords a short operation time, high resource utilization rate, and high decoding throughput rate. The iterative message of the BP algorithm is generally expressed in the form of probability or LLR. The advantage of the LLR–BP algorithm over the probability-based BP algorithm lies in transforming many multiplication operations into additive operations, thus considerably relieving the implementation complexity of decoding while preserving the decoding performance [5,18–29].

This paper introduces the min-sum decoding algorithm (MSA), based on the LLR–BP algorithm, and combines it with the high-efficiency and low-complexity coding algorithm to complete the design and implementation of a coder-decoder (codec).

The input information of the decoder after AWGN channel transmission was set as $y_i = (y_1, y_2, \cdots, y_n)$ and $y_i = x_i + n_i$, where $x_i (i = 1, \cdots, n)$ represents a symbol sequence, with the CW c_i having a value of {0, 1} that is mapped into {+1,−1}. n_i denotes the AWGB with a mean value of 0 and variance of σ^2.

In the k-th decoding iteration, $\lambda_n^{(k)}$ denotes the posterior LLR of the input data n-th bit, $\lambda_{mn}^{(k)}$ is the information transmitted from n to m, and $\Lambda_{mn}^{(k)}$ is the information transmitted from m to n. The implementation flow of the LLR-BP algorithm is presented below:

① Initiation: $n \in \{1, 2, \cdots, N\}$, $\lambda_{mn}^{(0)} = \lambda_n^{(0)}$, $m \in M(n)$, and $\lambda_{mn}^{(0)} = \lambda_n^{(0)}$, $m \in M(n)$ are initialized. The initial LLR received by the decoder after channel transmission is expressed as:

$$\lambda_n^{(0)} = \log \frac{p(x_i = 0|y_i)}{p(x_i = 1|y_i)} = \frac{2y_i}{\sigma^2} \tag{26}$$

In Equation (26), $\lambda_n^{(0)}$ is the initial value of n, $M(n)$ represents the set of all m connected to n, and $N(m)$ is the set of all n connected to m.

② Updating the check nodes: For each n and $n \in N(m)$, the following equation is calculated:

$$\Lambda_{mn}^{(k)} = \left(\prod_{n' \in N(m) \backslash n} sign\left(\lambda_{mn'}^{(k-1)}\right) \right) \Phi^{-1} \left(\sum_{n' \in N(m) \backslash n} \Phi\left(\left| \lambda_{mn'}^{(k-1)} \right| \right) \right) \tag{27}$$

In Equation (27), the functions $sign(x)$ and $\Phi(x)$ are defined as:

$$sign(x) = \begin{cases} +1 & x \geq 0 \\ -1 & x < 0 \end{cases} \tag{28}$$

$$\Phi(x) = \Phi^{-1}(x) = \lg\left(\frac{e^x + 1}{e^x - 1} \right) \; x > 0 \tag{29}$$

where $N(\mathrm{m})\backslash n$ denotes that the set of all variable nodes after n is deleted from the set $N(m)$, and $M(n)\backslash m$ denotes that the set of all check nodes after m is deleted from the set $M(n)$.

③ Updating the variable nodes: For each $m \in M(n)$ we calculate:

$$\lambda_{mn}^{(k)} = \lambda_n^{(0)} + \sum_{m' \in M(n) \backslash m} \Lambda_{m'}^{n} \tag{30}$$

The posterior LLR of each variable node is:

$$\lambda_n^{(k)} = \lambda_n^{(0)} + \sum_{m \in M(n)} \Lambda_{mn}^{(k)} \tag{31}$$

④ Decoding decision:

$$\hat{x}_n^{(k)} = \begin{cases} 0 & \lambda_n^{(k)} \geq 0 \\ 1 & \lambda_n^{(k)} < 0 \end{cases} \tag{32}$$

According to the check formula, the corrector of the CW $\hat{x}^{(k)}$ obtained through decoding is:

$$S = H \cdot (\hat{x}^{(k)})^T \tag{33}$$

If $S = 0$, the decoding has succeeded and the process ends by outputting the decoded CW $\hat{x}^{(k)}$ as an effective value. If $S \neq 0$, steps ②, ③, and ④ are repeated until the preset maximum number of iterations is reached.

For the LLR–BP algorithm, the nonlinear function $\Phi(x)$ is implemented through a lookup table during the hardware implementation process. The quantification of $\Phi(x)$ directly impacts the decoder's functional implementation and resource consumption.

4.2. Min-Sum Algorithm(MSA)

The MSA algorithm, a simplified form of the LLR-BP algorithm, performs the following simplification during the check nodes update [22–32]:

$$\Lambda_{mn}^{(k)} = \left(\prod_{n' \in N(m) \backslash n} sign\left(\lambda_{mn'}^{(k-1)}\right) \right) \left(\min_{n' \in N(m) \backslash n} \left| \lambda_{mn'}^{(k-1)} \right| \right) \tag{34}$$

The MSA algorithm detours the implementation of function $\Phi(x)$ by solving the minimum value and performing the additive operation, which, to some extent, reduces the implementation complexity of decoding. This paper uses the MSA algorithm as the decoding implementation algorithm, with its implementation flow depicted in Figure 3.

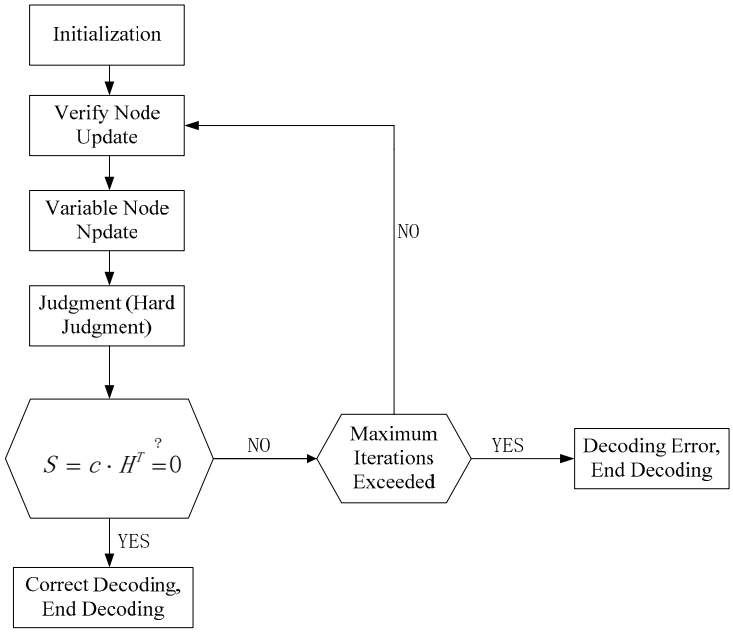

Figure 3. Implementation flow of MSA algorithm.

5. Design and Implementation of the Codec

5.1. Performance Analysis of Codec Algorithm

In the subsequent trials, we combined several coding algorithms with the MSA algorithm to design the (1536, 1024) LDPC codec uniformly quantized on (6, 2), involving eight iterations and a 2/3 code rate in the IEEE802.16e standard. In any case, we considered BPSK modulation and AWGN channel transmission. The corresponding codec performance simulation results are illustrated in Figure 4.

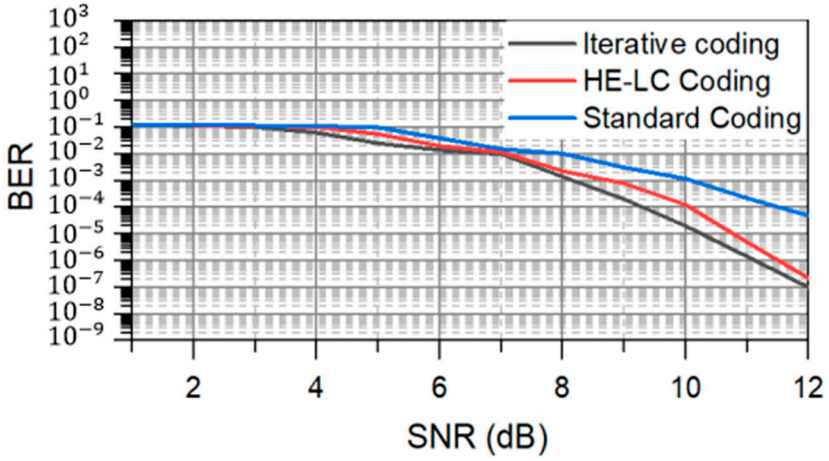

Figure 4. Performance simulation diagram of different coding algorithms.

At a bit error rate (BER) of 10^{-5}, the codec's BER obtained by the high-efficiency and low-complexity coding algorithm is about 0.25 dB lower than the recursive-iterative coding algorithm and about 1.25 dB lower than the standard coding algorithm. The simulation results presented in Figure 3 conclude that the LDPC codec implemented by the high-efficiency and low-complexity coding algorithm is characterized by high coding

performance and low complexity. As a reminder, this paper designs and implements the high-efficiency and low-complexity coding algorithm used by the LDPC coder, whereas the decoder employs the MSA algorithm.

5.2. Coder Design and Implementation

5.2.1. Coder Design

The coder structure (Figure 5) designed and implemented using the high-efficiency and low-complexity coding algorithm comprises an input/output random access memory (RAM) module, vector adder (VA) module, matrix-vector multiplier (MVM) module, CW generator (CWG) module, cache module, forwarder replacer (FS) module, and control module [28–35].

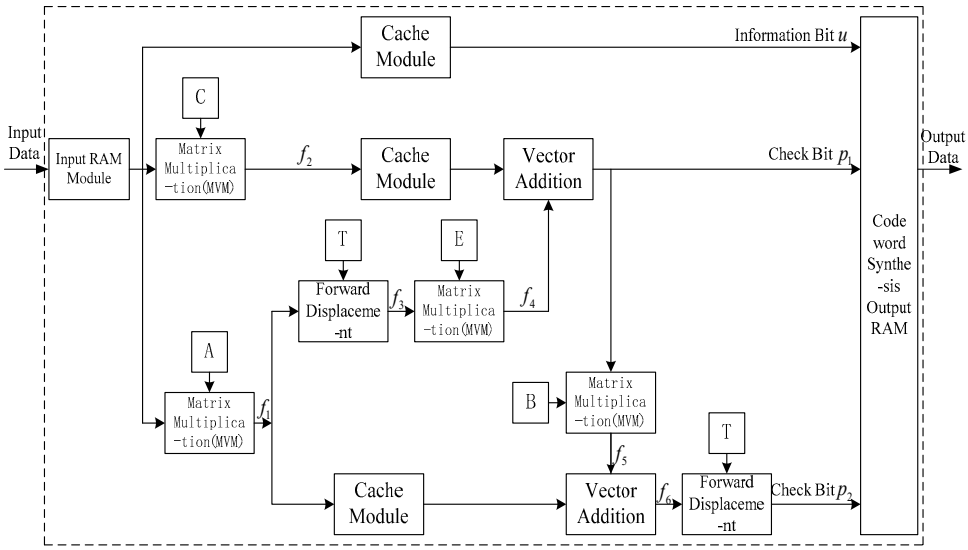

Figure 5. Functional block diagram of coder implementation.

① Input/output RAM module

During the encoding process, in order to continuously process the data and reduce the time complexity of operations, the ping-pong pipeline method shown in Figure 6 is used to store the data. The input RAM and output RAM are both dual-port RAM. By using the ping-pong operation, not only can data be seamlessly and continuously processed, but storage space can also be saved, and hardware implementation is easy.

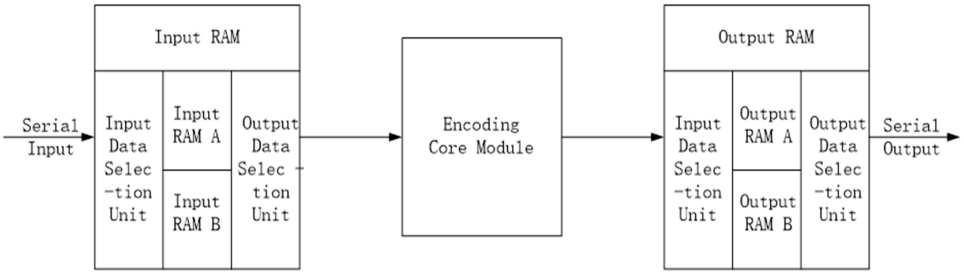

Figure 6. Principle diagram of Table Tennis Pipeline Mode.

② Matrix Multiplier (MVM) Module

The multiplier in the encoder is executed in parallel, consisting of a cyclic shifter and a modulo two adder. Its structural principle diagram is shown in Figure 7. The high 4 bits of the register store the row number of the check matrix, the middle 4 bits store the column number of the check matrix, and the low 7 bits store the circularly shifted value. By performing a cyclic shift operation on the vectors in the corresponding information

bits, a non-zero element and information bit multiplication operation can be completed. Then, the operation of multiplying the entire row of the matrix by the information bits can be performed, that is, multiplying all non-zero elements in this row by the information bits. Finally, performing the modulo 2 addition operation can complete the multiplication operation of the matrix.

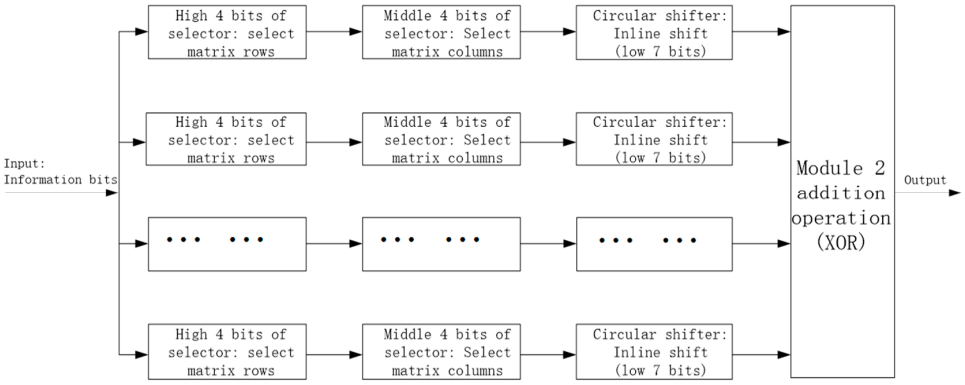

Figure 7. Schematic diagram of matrix multiplier.

In the design and implementation process of the encoder, how to effectively store the information of the verification matrix and how to effectively reduce the storage amount are the core issues of the research topic. From Figure 7, it can be seen that the elements of each row in the base check matrix are stored as 15 bits in a fixed row, which not only saves a lot of storage resources but also facilitates hardware implementation. For the LDPC code studied in the project, using traditional methods to store the verification matrix requires 4.42×10^5 bits resource space, whereas using the above methods only requires 8.38×10^3 bits resource space.

③ Forward Displacer (FS) Module

In the design process of the encoder, the forward displacement method was used to solve $f_3 = T^{-1} f_1$, and the calculation process can be completed using XOR operation. It has the characteristics of low computational complexity, low resource consumption, and easy hardware implementation. The calculation process is as follows:

Set up $f_1 = (v_1, v_2, \cdots, v_{m_b-1})$, $f_3 = (c_1, c_2, \cdots, c_{m_b-1})$, There are:

$$f_3 = T^{-1} f_1 \Rightarrow T f_3 = f_1 \Rightarrow \begin{cases} c_1 = v_1 \\ c_1 + c_2 = v_2 \\ c_2 + c_3 = v_3 \\ \vdots \\ c_{m_b-2} + c_{m_b-1} = v_{m_b-1} \end{cases} \Rightarrow \begin{cases} c_1 = v_1 \\ c_2 = v_2 + v_1 \\ c_3 = v_3 + v_2 \\ \vdots \\ c_{m_b-1} = v_{m_b-1} + v_3 + v_2 + v_1 \end{cases} \tag{35}$$

The computational process was executed in parallel, which not only reduces computational complexity but also minimizes computational latency. When solving $p_2 = T^{-1} f_6$, the same method can be used for calculation.

④ Code word generator (CWG) module

The function of the codeword generator module is to combine the calculated check bits p_1 and p_2 with existing information bits into a complete codeword, and then store it in the output RAM according to the synthesis rule $[u, p_1, p_2]$ of the codeword. The schematic diagram is shown in Figure 8.

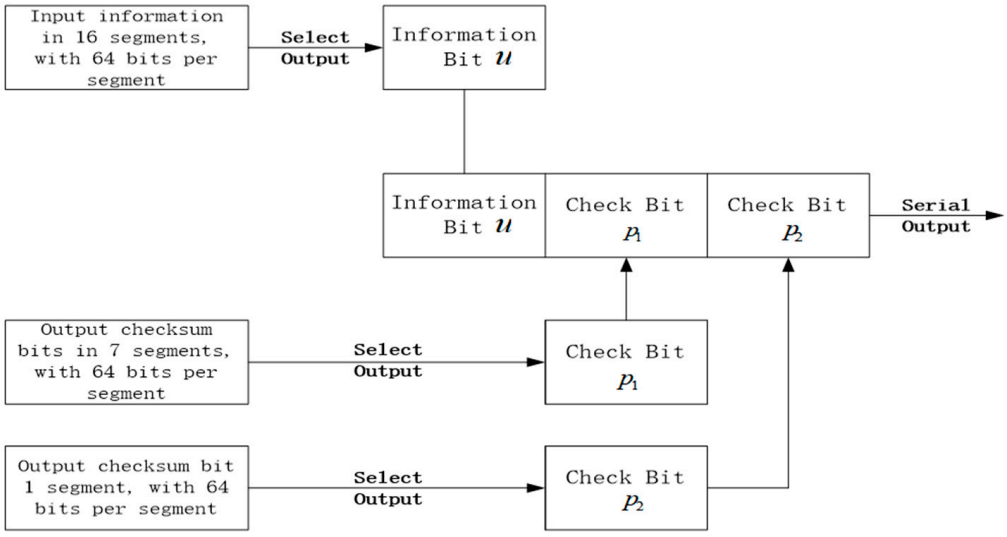

Figure 8. Principle diagram of codeword generator.

⑤ Other modules

In addition to the modules discussed above, the encoder consists of a vector adder module (VA), a control module (Control), and a cache module (Cache). The function of the VA module is to perform addition operations between vectors; the function of the control module is to generate the control signals required during the encoding process; and the function of the cache module is to coordinate with the next step of data synchronization.

5.2.2. Coder Implementation

The correctness and feasibility of the coding algorithm and design flow are first verified via MATLAB simulation during the coder implementation process. Then, each module is designed and implemented in the Vivado integrated environment according to the coder's functional block diagram, followed by the hardware design for this coder. Finally, the functional simulation and board-level test are conducted for the designed and implemented coder, thus completing the coder implementation that meets the demands and verifies the effectiveness and feasibility of the research design.

① MATLAB simulation results

In order to verify the correctness and feasibility of the designed encoder, the M files of each module in the encoding process are first written in MATLAB. Then, a string of 1024 bit binary metadata information is input in MATLAB for encoding processing. After the encoding is completed, the 1536 bits data output is compared and analyzed with theoretical values to determine the correctness of the design.

Figure 9 shows the simulation diagram of the input and output data of the designed encoder during the simulation verification process. Through the diagram, the input information of the encoder, the verification information generated by the encoder, and the output codeword information after encoding can be seen. By comparing and analyzing the software simulation results and theoretical calculation results, it was found that the two are completely identical, which further verifies the correctness and feasibility of the encoder designed in this article.

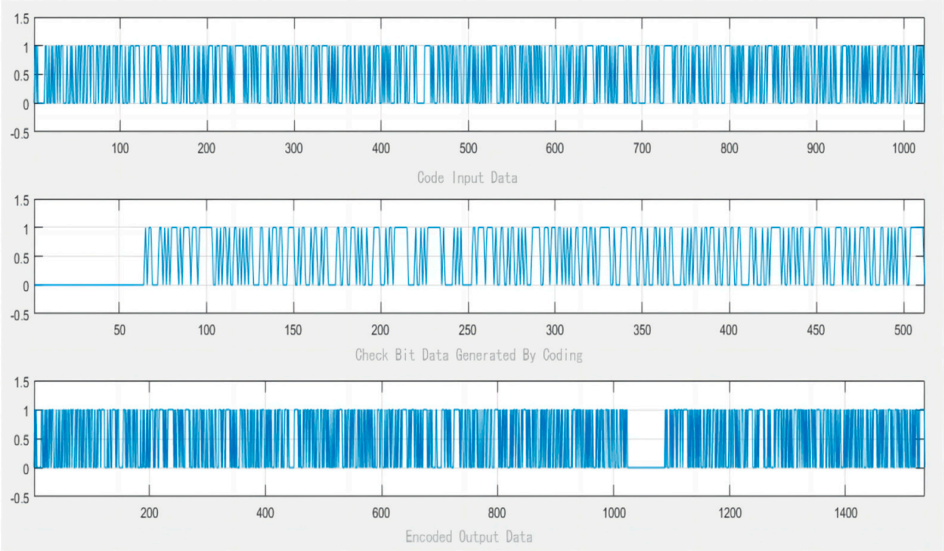

Figure 9. The coder's MATLAB simulation results.

② FPGA Implementation Results

The correctness and feasibility of the encoder design process were verified through MATLAB simulation implementation. On the basis of software simulation implementation, first, according to the encoder implementation schematic, we performed Verilog HDL programming on each module in Vivado2018.2; then, we called each module through the top-level file to complete the hardware design of the encoder; and finally, through functional simulation and board level testing, the implementation of an encoder that meets the performance requirements of the project was completed.

The performance indicators during the encoder testing and verification process are as follows: the input data transmission rate was 2 Mbps and the system clock frequency was 150 MHz (the highest operating frequency of the system is 250 MHz). The functional simulation diagram during the hardware design and implementation process of the LDPC encoder is shown in Figures 10 and 11 and Table 4.

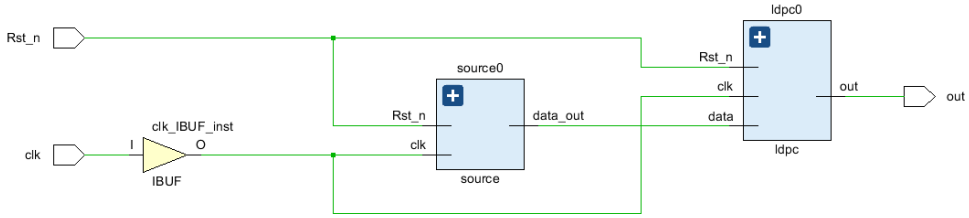

Figure 10. RTL-level simulation diagram of LDPC encoder.

Figure 10 is the RTL-level simulation implementation diagram of the encoder. In order to simulate and verify the designed and implemented encoder, a signal source module was added during the encoder design process to generate input data for the encoder verification testing process. Table 4 shows the resource usage report of the encoder. Through observation and analysis, it can be seen that the encoder implementation process requires 10752 LUTs, accounting for 20.00% of the total quantity, and 12658 Registers, accounting for 12.00% of the total quantity. Figure 11 shows the simulation test results of the encoder, which is designed using an efficient and low-complexity encoding algorithm. The system clock frequency clk is 150 MHz, where u represents the 1024 bit data input by the encoder, p_1 and p_2 represent the 64 bit and 448 bit verification data generated by the encoder, and u_1 represents the 1536 bit data output after encoding. According to the observation and analysis in Figure 11, it can be seen that the encoder takes 5632 clock cycles to complete the encoding of a frame of data. During the encoding process, using the

ping-pong pipeline operation mode can save a lot of time and improve the encoding speed. The input and output data of the encoder are consistent with the MATLAB simulation results, verifying the correctness and feasibility of the encoder hardware design.

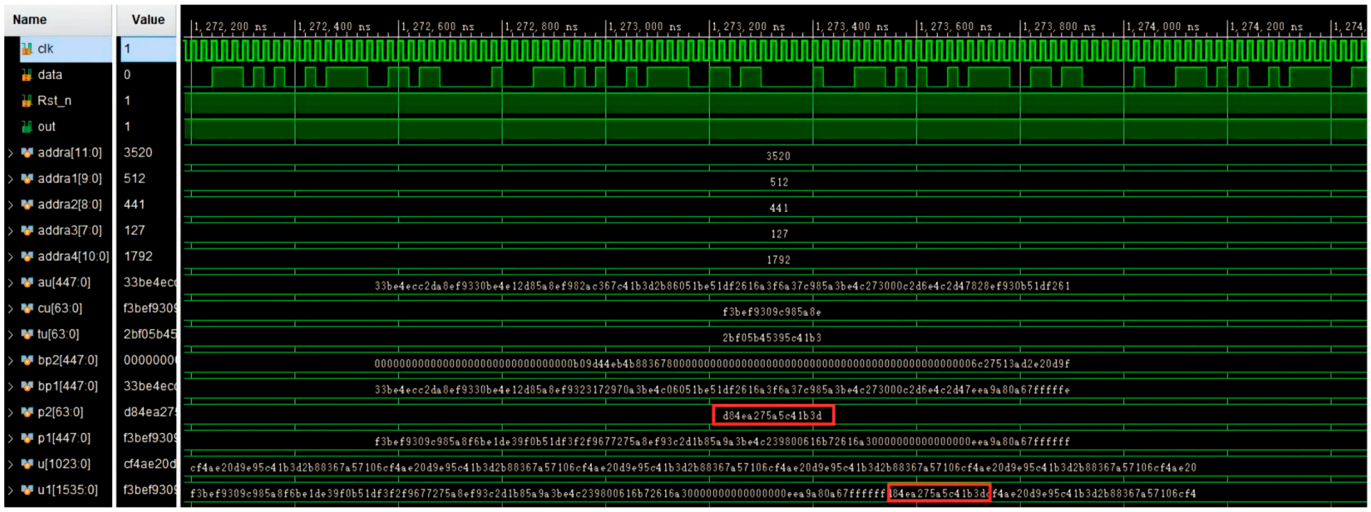

Figure 11. The coder's hardware test results. The solid boxes represent The data information of the check bit p_2 generated during the encoding process.

Table 4. Encoder resource use report. * represent The name annotation of logical devices in Xilinx Zynq7020 FPGA hardware resources can be deleted.

Site Type	Used	Fixed	Available	Util%
Slice LUTs *	10,752	0	53,200	20.00
LUT as Logic	10,752	0	53,200	20.00
LUT as Memory	0	0	17,400	0.00
Slice Registers	12,658	0	106,400	12.00
Register as Flip Flop	12,658	0	106,400	12.00
Register as Latch	0	0	106,400	0.00
F7 Muxes	1904	0	26,600	7.00
F8 Muxes	888	0	13,300	7.00

5.3. Decoder Design and Implementation

5.3.1. Decoder Design

The decoding process is conducted by MSA algorithm and corresponding serial implementation structure is shown in Figure 12. For detail, the serial structure of the MSA algorithm comprises a variable node processor (VNP), check node processor (CNP), intermediate message RAM, control unit, and all types of data storage devices.

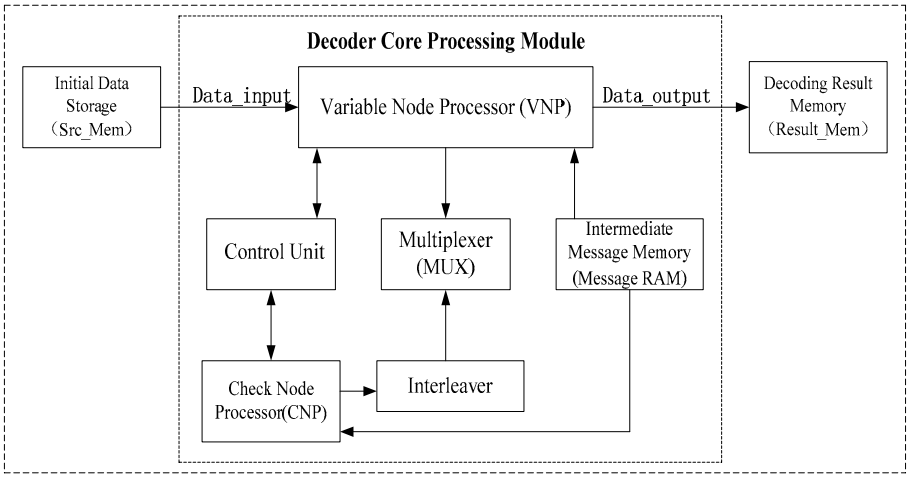

Figure 12. Functional block diagram of the decoder designed through the serial structure of the MSA algorithm.

① Variable Node Processor (VNP)

The variable node processor has two functions: one is to input and output data information, and the other is to update the information of variable nodes. During the initialization phase, due to the fact that all data stored in Message RAM is 0, after the first sum operation, the input data is sequentially the corresponding channel information. VNP completes the update function of variable node information according to Equation (30).

② Verify Node Processor (CNP)

The function of the verification node processor is to update the information of the verification node, and the CNP completes the update function of the verification node information according to Equation (34).

③ Message RAM

The intermediate information storage is a dual-port RAM, with one port read-only and the other port write-only. VNP and CNP sequentially read and write Message RAM through the control of enable signals. The capacity of Message RAM is set to 24576 bits, which means that the bit width of RAM is 4 and the depth is 6144.

④ Control Unit

The control unit completes the control functions of the decoding system, including controlling the number of iterations, controlling the working state of nodes, and controlling the output of decoding end code words.

⑤ Interleaver

During the implementation of the decoder, only one Message RAM is used to store data. When storing VNP and CNP data, an interleaver is needed to interleave and reorder the CNP data to improve the error performance of the decoder. The data relationships in the interleaver are uniquely determined by the check matrix.

⑥ Other module units

Initial data storage (Src-Mem): This is used to store data to be decoded, with a data width of 4 bits and a depth of 6144.Decoding Result Memory (Result-Mem): This is used to store the data output after decoding, with a data width of 4 bits and a depth of 6144.

5.3.2. Decoder Implementation

In the decoder implementation process, the correctness and feasibility of the decoding algorithm and design flow completely adopt the decoder's design implementation and

evaluation, meeting the demands and verifying the correctness and feasibility of the research design.

① MATLAB simulation results

In order to verify the correctness and feasibility of the decoder design, the M files of each component module of the decoder design were first written on MATLAB, and then the M files of the entire decoder were simulated. Through the research and analysis of the simulation waveform, the correctness of the designed decoder was determined. In the MATLAB simulation process, the input of the decoder is the data information transmitted through the channel after encoding and modulation. During the simulation process, the channel is set as an additive Gaussian white noise channel, and the signal-to-noise ratio is set to 3 dB. The MATLAB simulation waveform is shown in Figures 13 and 14.

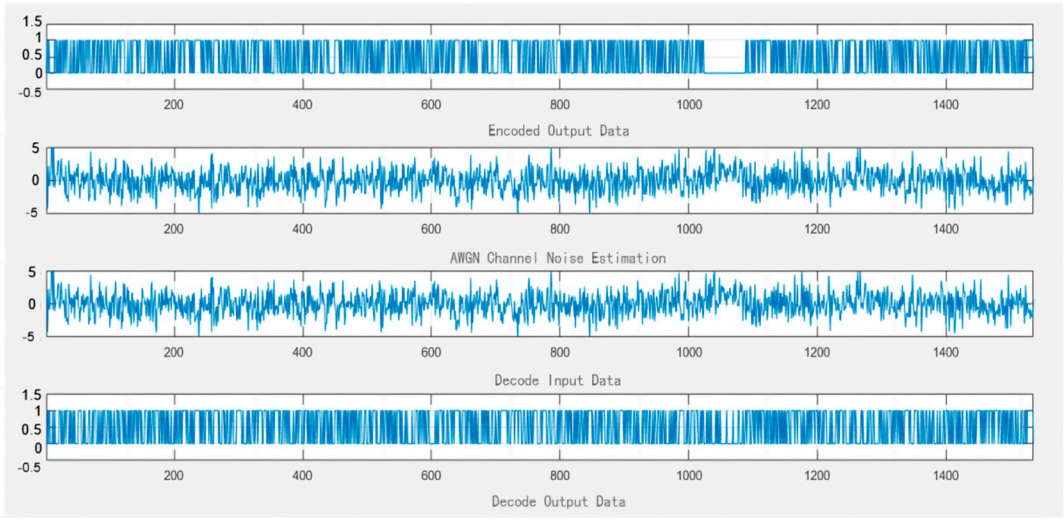

Figure 13. MATLAB simulation results of decoder.

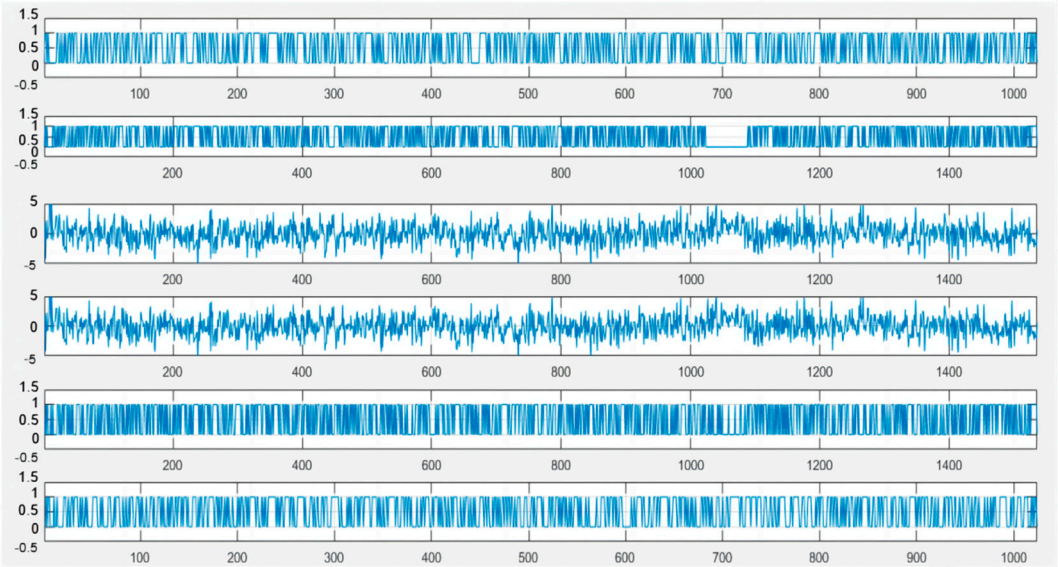

Figure 14. MATLAB simulation results of codec.

② FPGA Implementation Results

The correctness and feasibility of the decoder design process were verified through MATLAB simulation implementation. On the basis of software simulation implementation, first, according to the decoder implementation schematic, we performed Verilog HDL

programming on each module in Vivado2018.2; then, we called each module through the top-level file to complete the hardware design of the decoder; and, finally, through functional simulation and board level testing, the implementation of a decoder that meets the performance requirements of the project was completed.

The performance indicators during the decoder testing and verification process were as follows: the input data transmission rate was 2 Mbps and the system clock frequency was 150 MHz (the highest operating frequency of the system is 250 MHz). The functional simulation diagram during the hardware design and implementation process of the LDPC decoder is shown in Figures 15 and 16 and Table 5.

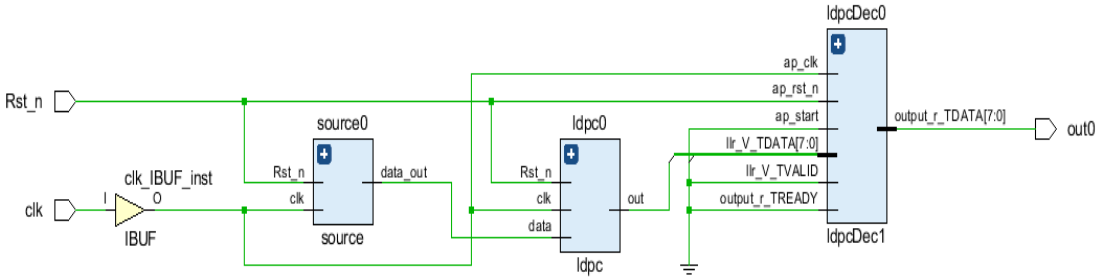

Figure 15. RTL-level simulation results of coder and decoder.

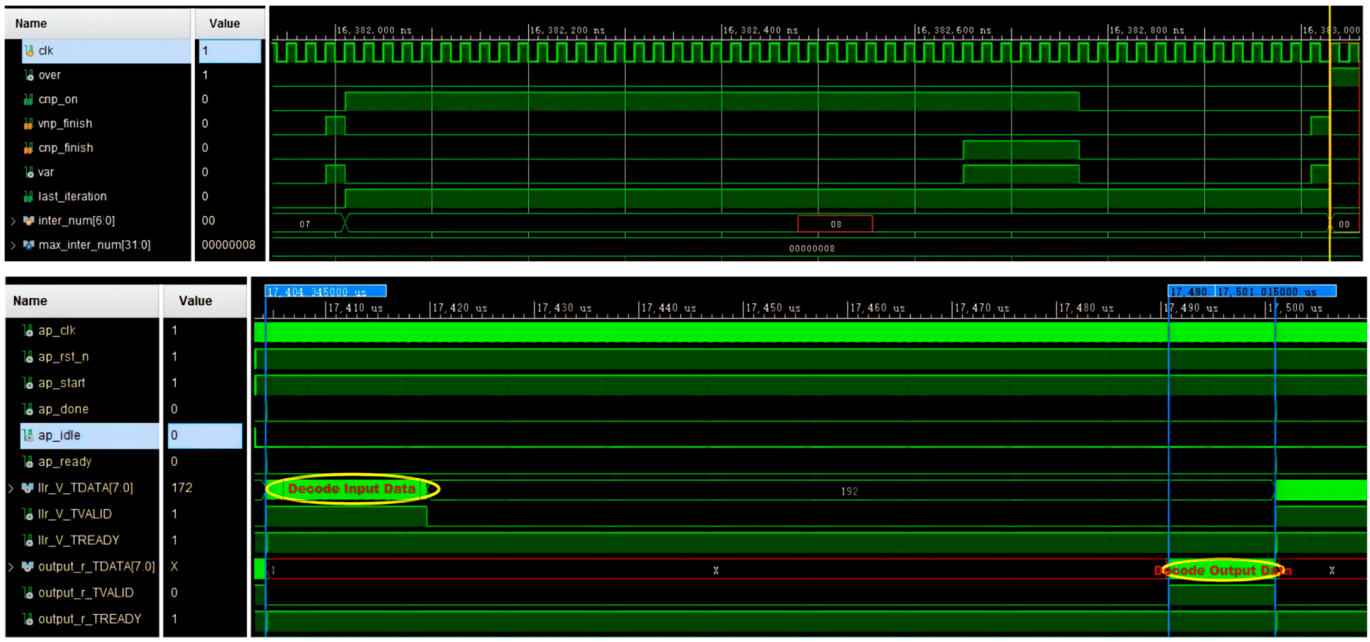

Figure 16. Hardware test results of codec.

Figure 16 shows the RTL-level simulation results of the encoder and decoder, and Table 5 shows the resource usage report of the bit encoder and decoder.

Figure 16 shows the simulation test results of the decoder. The decoder was designed using the minimum sum decoding algorithm, with 8 decoding iterations and (4,2) uniform quantization method. The system clock frequency clk was 150 MHz—over is the end of decoding processing flag, and a high level indicates the end of decoding processing; Cnp_ The on enable signal determines whether the next state is variable node update processing or verification node update processing, vnp_Finish is the identification signal for the completion of variable node update processing, cnp_Finish is the completion signal of the verification node update processing, last_Iteration marks the signal for the last iteration, indicating the last iteration cycle, and prompts the variable node processor to output the decoding result after, processing_Num is the number of decoding iterations, and

the maximum number of iterations for the designed decoder is 8; AP_Rst_N is the system reset signal, with high-level reset; AP_Done outputs the enable signal for the decoding result, and llr_ V_TDATA [7:0] is the 1024 bit initialization data input to the decoder. The number of iterations during the decoder design and implementation process is set to 8, and the output is_R_TDATA [7:0] is the 1536 bit decoding data output after decoding is completed. From research on and analysis of the LDPC code verification matrix used, combined with the decoding simulation results of the decoder, it can be seen that the decoder needs to take 5632 clock cycles to complete the decoding process of a frame of data. During the decoding implementation process, each variable node and verification node need to be calculated one by one. The input and output data of the decoder are consistent with the MATLAB simulation results, verifying the correctness and feasibility of the decoder hardware design.

Table 5. Coder and decoder resource usage report. * represent The name annotation of logical devices in Xilinx Zynq7020 FPGA hardware resources can be deleted.

Site Type	Used	Fixed	Available	Util%
Slice LUTs *	36,689	0	53,200	69.00
LUT as Logic	36,022	0	53,200	68.00
LUT as Memory	667	0	17,400	4.00
LUT as Distributed RAM	171	0		
LUT as Shift Register	496	0		
Slice Registers	28,536	0	106,400	27.00
Register as Flip Flop	28,536	0	106,400	27.00
Register as Latch	0	0	106,400	0.00
F7 Muxes	2480	0	26,600	9.00
F8 Muxes	920	0	13,300	7.00

6. Conclusions

LDPC codes are highly regarded as linear block codes due to their exceptional coding performance, low decoding complexity, flexible structure, and easy hardware implementation. Consequently, they have become a prominent research topic in the field of channel coding. To address the challenge of high computational complexity in implementing standard encoding algorithms, this study proposes an efficient and low-complexity encoding algorithm through extensive research and analysis of check matrices, building upon the investigation of standard encoding algorithms and recursive iterative encoding algorithms.

By combining the LLR-BP algorithm with the minimum sum decoding algorithm, we successfully implemented an (1536, 1024) LDPC encoder and decoder with a 2/3 code rate using (6, 2) uniform quantization. The iteration count was set at 8, adhering to the IEEE 802.16e standard. This implementation was tested through software simulation on MATLAB and hardware testing on the Xilinx Zynq7020 FPGA platform. The implementation results obtained from both software simulation and hardware testing demonstrate that when utilizing the minimum sum decoding algorithm in conjunction with BPSK modulation and AWGN channel transmission, the proposed efficient and low-complexity encoding algorithm achieves approximately a 0.25 dB improvement in encoding performance compared to the recursive iterative encoding algorithm. Moreover, it achieves around a 1.25 dB improvement compared to the standard encoding algorithm. These findings further validate the correctness, effectiveness, and feasibility of the proposed algorithm. The experimental results highlight that employing efficient and low-complexity encoding algorithms not only reduces computational complexity and logical delays during the encoder implementation process, but also enhances encoding performance and data transmission reliability. This bears significant theoretical and practical research implications

for advancing the widespread application and rapid development of LDPC codes in the realm of digital communication.

Author Contributions: Methodology, X.L., J.G. and Z.L.; Software, Y.X.; Data curation, Y.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China under Grant no. 61703060, the National Engineering Research Center for Oil & Gas Drilling Equipment under grant no. 202307, the Sichuan Science and Technology Plan Project (grant nos. 2020YFS0507, 2021YFG0361, 2021YFS0311, 2023YFS0426).

Data Availability Statement: The data that support the findings of this study are available from the corresponding author upon reasonable request.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Liao, X.; Yu, N.; Luo, Z.; Ren, X.; Fang, A. Research and Design of LDPC Codes Based on MATLAB/Simulink. *J. Chengdu Univ. (Nat. Sci. Ed.)* **2017**, *36*, 272–275+303.
2. Gallager, R.G. Low Density Parity Check Codes. *IRE Trans. Inf. Theory* **1962**, *8*, 2018–2220. [CrossRef]
3. MacKay, D.J.; Neal, R.M. Near Shannon Limit Performance of Low Density Parity Check Codes. *Electron. Lett.* **1996**, *32*, 1645–1646. [CrossRef]
4. Xu, R. *Design of LDPC Decoder for CCSDS Deep Space Communication Standard*; Xi'an University of Technology: Xi'an, China, 2010; pp. 17–18.
5. Yang, X. *Turbo and LDPC Codec and Their Applications*; People's Posts and Telecommunications Publishing House: Beijing, China, 2010.
6. Wang, M. *Design and FPGA Implementation of High Speed LDPC Codec*; University of Electronic Science and Technology: Chengdu, China, 2016.
7. Guo, H. *Research on Implementation of LDPC Codes Based on IEEE802.16e Standard*; Harbin Institute of Technology: Harbin, China, 2010.
8. Fan, K. *Research and Implementation of LDPC Coding and Decoding Technology Based on IEEE802.16e*; Xi'an University of Electronic Science and Technology: Xi'an, China, 2009.
9. Wu, Z.; Zhang, L.; Zhong, Z.; Liu, R. Reconstruction of LDPC Sparse Check Matrix under High Bit Error Rate. *J. Commun.* **2021**, *42*, 1–10.
10. Du, G. An overview of the principle and application of LDPC codes. *China New Commun.* **2012**, *14*, 25–33.
11. Li, P.; Qi, F.; He, D.; Li, J. Design of IEEE 802.16e standard LDPC encoder based on FPGA. *Mod. Navig.* **2022**, *13*, 212–217+222.
12. Guodong, W.A.; Jinming, L.I.; Zhiwang, Z.H.; Denghui, T.I. Design and Implementation of LDPC Encoder Based on FPGA. *J. Meas. Sci. Instrum.* **2021**, *12*, 12–19.
13. Xue, W.; Yu, H.; Wang, J.; Shu, F. Optimization of High Efficiency LDPC Decoder and Implementation of FPGA. *Data Acquis. Process.* **2018**, *33*, 1101–1111.
14. Sun, N. *Research on Parity Check Matrix Construction and Decoding Optimization Algorithm of LDPC Codes*; Shandong University: Jinan, China, 2019.
15. Liao, P. *Research and Design Implementation of LDPC Code High Speed Decoder in Deep Space Communication*; Yanshan University: Qinghuangdao, China, 2022. [CrossRef]
16. Shi, S.; Wang, R.; Li, H.; Han, C. Implementation of Multipath Parallel Encoder for LDPC Code. *J. Electron. Meas. Instrum.* **2021**, *35*, 83–89. [CrossRef]
17. Shao, B. *Research and Implementation of LDPC Code in 5G Communication System*; Xi'an University of Electronic Science and Technology: Xi'an, China, 2022. [CrossRef]
18. Richardson, T.J.; Urbanke, R.L. Efficient encoding of low-density parity-check codes. *IEEE Trans. Inf. Theory* **2001**, *47*, 638–655. [CrossRef]
19. Lee, J.H.; Sunwoo, M.H. Low-Complexity High-Throughput Bit-Wise LDPC Decoder. *J. Signal Process. Syst.* **2019**, *91*, 855–862. [CrossRef]
20. Zhang, C.; Su, K. *Practice of Digital Signal Processing and Engineering Application of FPGA*; China Railway Press: Beijing, China, 2013.
21. Guo, L.; Chen, H. LDPC coding and decoding method based on FPGA for IEEE 802.16e. *Autom. Technol. Appl.* **2017**, *36*, 49–53.
22. Shan, B.; Li, Z. Design and performance analysis of improved LDPC decoding scheme. *Comput. Eng. Des.* **2019**, *40*, 1507–1511.
23. Chen, F. *Low Complexity Deep Learning LDPC Decoding*; Central South University for Nationalities: Wuhan, China, 2021. [CrossRef]
24. Yang, H. *Research and Implementation of LDPC Decoder in Satellite Communication*; Xi'an University of Electronic Science and Technology: Xi'an, China, 2023. [CrossRef]

25. Luo, X. *Research on Hybrid Decoding Algorithms for LDPC Codes*; University of Electronic Science and Technology: Chengdu, China, 2022. [CrossRef]
26. Wang, D. *Improvement of Decoding Algorithm Based on LDPC Code and FPGA Implementation*; Nanjing University of Information Engineering: Nanjing, China, 2021. [CrossRef]
27. Wang, L.; Li, J. Design and Implementation of LDPC Decoder Based on FPGA. *Electron. Meas. Technol.* **2022**, *45*, 22–27. [CrossRef]
28. Li, J.; Chen, B. FPGA Implementation of QC-LDPC Decoder Based on Minimum Sum Algorithm. *Appl. Sci. Technol.* **2020**, *47*, 35–40.
29. Chen, F.; Liu, Y.; Tang, C. A Low Complexity Normalized Minimum Sum Decoding Algorithm for LDPC Codes. *J. Chongqing Univ. Posts Telecommun. (Nat. Sci. Ed.)* **2020**, *32*, 92–98.
30. Sun, J.; Li, J. LDPC Minimum Sum Decoding Algorithm and Its IC Physical Design. *J. Meas. Sci. Instrum.* **2023**, *14*, 108–115.
31. Li, J.; Zhang, P.; Wang, L.; Wang, G. An FPGA LDPC decoder for optimizing scaling factors in NMS decoding algorithms. *J. Meas. Sci. Instrum.* **2022**, *13*, 398–406.
32. Yang, P.; Jun, B.; No, J.S.; Park, H. A new two-stage decodingscheme with unreliable path search to lower the error-floor for low-density parity-check codes. *IET Commun.* **2017**, *11*, 2173–2180. [CrossRef]
33. Han, X. *Design and Optimization of LDPC Codec in High Speed WLAN System*; Beijing University of Posts and Telecommunications: Beijing, China, 2014.
34. Wang, H.; Guo, D. Design of LDPC decoder based on FPGA. *J. Lul. Univ.* **2019**, *9*, 34–40.
35. Gu, S.; Luo, Z.; Chu, Y.; Xu, Y.; Guo, J. A Suboptimal Optimizing Strategy for Velocity Vector Estimation in Single-Observer Passive Localization. *Sensors* **2023**, *23*, 5940. [CrossRef]