applied sciences

MDPI

Article

An Improved Low-Density Parity-Check Decoder and Its Field-Programmable Gate Array Implementation

Hao-Yu Wang [1,2], Zhong-Xun Wang [1,2,*] and Shuo Shang [1,2]

[1] School of Physics and Electronic Information, Yantai University, Yantai 264005, China; 17854114455@163.com (H.-Y.W.); 17860390982@163.com (S.S.)
[2] Shandong Data Open Innovation Application Laboratory of Smart Grid Advanced Technology, Yantai University, Yantai 264005, China
* Correspondence: ytdxwzx@163.com

Abstract: Based on the IEEE 802.16e standard's (672,336) LDPC code and the normalized Min-Sum decoding algorithm, this paper designs and implements an LDPC decoder that optimizes the channel information. The correction factor for check nodes is converted into a correction factor for the initial channel information, replacing the optimization of check node information with that of initial channel information. This achieves decoding performance equivalent to the traditional normalized Min-Sum decoding algorithm. Different correction factor values vary in complexity during FPGA implementation, as they involve different amounts of shift-add operations. For NMS decoding requiring a high number of shift-add operations to achieve optimal correction values, this can be converted into an LDPC decoding algorithm optimized for channel information, reducing computational overhead without sacrificing performance. A partially parallel improved decoder was designed and implemented on an FPGA, and its feasibility was verified using the Vivado simulation platform.

Keywords: LDPC; NMS; FPGA

Citation: Wang, H.-Y.; Wang, Z.-X.; Shang, S. An Improved Low-Density Parity-Check Decoder and Its Field-Programmable Gate Array Implementation. *Appl. Sci.* **2024**, *14*, 5162. https://doi.org/10.3390/app14125162

Academic Editor: Alexander Barkalov

Received: 7 May 2024
Revised: 5 June 2024
Accepted: 7 June 2024
Published: 13 June 2024

Correction Statement: This article has been republished with a minor change. The change does not affect the scientific content of the article and further details are available within the backmatter of the website version of this article.

1. Introduction

In 1962, Gallager first introduced Low-Density Parity-Check (LDPC) codes as a type of linear block code characterized by their sparse parity-check matrix [1], which distinguishes them from general linear block codes. In 1981, Tanner proposed using bipartite graphs to describe the parity-check matrix [2]. Based on the concept of short cycles in Tanner graphs, MacKay and Davey performed a detailed analysis of the theory and performance of LDPC codes [3,4], proving that LDPC codes can approach the Shannon limit as code lengths increase [5]. They also proposed the sum-product decoding algorithm [6].

Subsequently, Fossorier and Eleftheriou developed BP decoding algorithms in the probability domain and LLR domain, respectively [7]. However, because the hyperbolic tangent operations in the check nodes were overly complex and impractical for hardware implementation, the Min-Sum (MS) decoding algorithm was introduced as an alternative, using the minimum value of variable nodes to replace the hyperbolic tangent operations in check nodes. While the MS algorithm significantly reduces computational complexity compared to earlier algorithms, it also results in notable performance degradation [8]. Thus, a series of algorithms were proposed to compensate for this loss, including Normalized Min-Sum (NMS) decoding [9], Offset Min-Sum (OMS) decoding [10], Adaptive Min-Sum decoding [11], and Self-Correcting Min-Sum decoding [12,13].

In theory, the longer the LDPC code, the higher its performance. However, longer codes also lead to higher memory consumption and greater computational overhead. Moreover, the random distribution of the parity-check matrix complicates circuit implementation. To reduce hardware implementation complexity, Quasi-Cyclic LDPC (QC-LDPC) codes

were proposed, dividing the parity-check matrix into multiple regularly patterned submatrices. Their storage and addressing characteristics significantly simplify hardware implementation, enhancing practical utility. As a result, QC-LDPC codes have quickly been adopted into various mobile communication standards. Standards across diverse communication fields have been developed, including the DVB-S2 standard for satellite broadcasting [14], the IEEE 802.16 standard for wireless metropolitan area networks [15,16], and the CCSDS standard for deep space communication.

Research on LDPC codes with floating-point calculations tends to require significant hardware resources for implementation. Therefore, a quantized Min-Sum decoding algorithm using amplified, rounded floating-point numbers was proposed. Building on the nature of quantization and the structure of the NMS decoding algorithm, this paper presents an improved NMS decoding algorithm with correction factor transfer. This improvement enables some eligible LDPC decoding implementations to reduce computational overhead without compromising performance.

The remainder of this paper is structured as follows. Section 2 introduces the NMS algorithm and describes the structure of LDPC codes in the IEEE 802.16e standard. In Section 3, a Normalized Min-Sum algorithm with transfer correction factors is proposed based on the NMS algorithm, and theoretical comparisons demonstrate reduced computational overhead under certain conditions. Section 4 details the architecture of the FPGA implementation for an LDPC decoder, which utilizes an enhanced algorithm tailored for the IEEE 802.16e standard. Section 5 validates the algorithm's feasibility using MATLAB and simulation of the FPGA decoder via the VIVADO software platform, confirming that the improved algorithm achieves performance comparable to the original. Section 6 summarizes the conclusions of the paper.

2. LDPC Code Structure and Decoding Algorithm

2.1. LDPC Decoding Algorithm

The LDPC decoding methods can be broadly categorized into two classes: hard-decision algorithms based on bit-flipping decoding and soft-decision algorithms. Hard-decision decoding algorithms, due to their low complexity, are suitable for hardware implementation but cannot achieve optimal decoding performance. Soft-decision algorithms, while more complex to implement in hardware, can theoretically achieve performance close to the Shannon limit.

To reduce the difficulty of hardware implementation, a simplified version of the Min-Sum (MS) algorithm was proposed, giving rise to a series of improved MS algorithms, such as the Normalized Min-Sum (NMS) and Offset Min-Sum (OMS) algorithms, which retain low implementation complexity.

As a type of linear block code, an LDPC code is defined by a parity-check matrix H consisting of m rows and n columns. It is represented by a Tanner graph that includes n variable nodes and m check nodes. Here, the i-th variable node, denoted as v_i, represents the i-th bit of the transmitted codeword. The j-th check node, referred to as c_j, corresponds to the j-th parity-check equation. A connection between the i-th variable node v_i and the j-th check node c_j is established when $H_{ij} = 1$. Taking the Formula (1) matrix as an example, the corresponding set of parity-check equations is given by equation Formula (2), and the associated Tanner graph is denoted as Figure 1. Tanner graph corresponding to Matrix H, which visually demonstrates the encoding and decoding process of the LDPC code.

$$H = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{bmatrix} \quad (1)$$

$$\begin{cases} c_1 + c_3 + c_4 + c_8 + c_{10} + c_{11} = 0 \\ c_3 + c_5 + c_6 + c_8 + c_9 + c_{12} = 0 \\ c_1 + c_2 + c_4 + c_6 + c_9 + c_{10} = 0 \\ c_1 + c_2 + c_4 + c_7 + c_{11} + c_{12} = 0 \\ c_3 + c_5 + c_6 + c_7 + c_9 + c_{10} = 0 \\ c_2 + c_5 + c_7 + c_8 + c_{11} + c_{12} = 0 \end{cases} \tag{2}$$

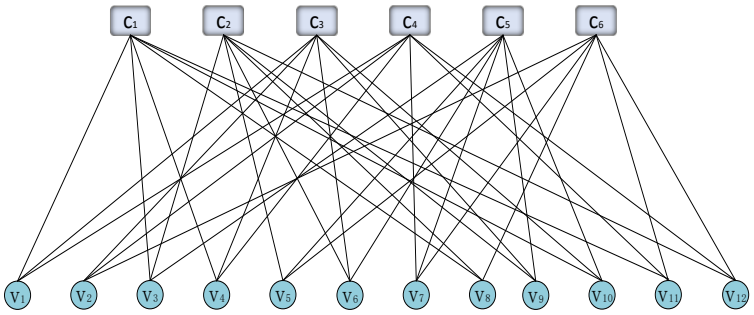

Figure 1. Tanner graph corresponding to Matrix H.

Initially, implementing the Belief Propagation (BP) algorithm in the probability domain required extensive use of multiplication, division, and logarithmic operations, making it not only challenging to implement on hardware but also prone to numerical instability with longer code lengths. The introduction of the Log-Likelihood Ratio Belief Propagation (LLR-BP) algorithm, which transforms multiplication operations into addition operations through the use of log-likelihood ratios, significantly reduced the complexity of hardware implementation.

The log-likelihood ratio is first defined as Formula (3).

$$L(Q_i) = \log(\Pr(x_i = 0 \mid y_i)/\Pr(x_i = 1 \mid y_i)) \tag{3}$$

Under the condition of Gaussian white noise with a variance of σ^2, Formula (4) can be simplified to the following:

$$L(Q_i) = \log(\frac{\Pr(x_i = 1 \mid y_i)}{\Pr(x_i = -1 \mid y_i)}) = \log(\frac{1 + e^{2y_i/\sigma^2}}{1 + e^{-2y_i/\sigma^2}}) = 2y_i/\sigma^2 \tag{4}$$

The update for the check nodes is as follows:

$$L(c_{ji}) = \prod_{i' \in M(j) \setminus i} sign(Lv_{i'j}) \phi(\sum_{i' \in M(j) \setminus i} \phi(|Lv_{i'j}|)) \tag{5}$$

In Formula (5), $\phi(x) = -\log(\tanh(x/2)) = \log\left(\frac{e^x+1}{e^x-1}\right)$

For the update of variable nodes

$$L(v_{ij}) = L(Q_i) + \sum_{j' \in N(i)/j} L^{(l)}(c_{j'i}) \tag{6}$$

Regarding the posterior probability

$$L(q_i) = L(Q_i) + \sum_{j \in N(i)} L^{(l)}(c_{ji}) \tag{7}$$

In the LLRBP algorithm, the hyperbolic tangent operations within the check nodes remain overly complex for hardware implementation. Taking advantage of the property that the slope of the hyperbolic tangent function decreases as the input x increases, the Min-Sum decoding algorithm was proposed. Its main improvement involves selecting the

minimum values from the variable nodes to replace the hyperbolic tangent operations in the check nodes.

Thus, Formula (5) is modified to Formula (8):

$$L(c_{ji}) = \prod_{i' \in M(j) \setminus i} \operatorname{sgn}(L^{(l-1)}(v_{i'j})) \times \min_{i' \in M(j) \setminus i}(|L^{(l-1)}(v_{i'j})|) \tag{8}$$

The following is a brief description of the decoding steps for the NMS algorithm:

Compared to earlier decoding algorithms, the Min-Sum decoding algorithm significantly reduces computational complexity, but it also considerably impacts performance. Consequently, the Normalized Min-Sum (NMS) decoding algorithm was introduced, which incorporates a normalization correction factor into Formula (8). This modification transforms Formula (8) into Formula (10), effectively mitigating the performance degradation incurred by simplifying the LLRBP algorithm to the MS algorithm.

Initialization: Set the maximum number of iterations as $l_{\max}$ and the current iteration count as $l = 0$. Initialize the channel information as the variable node information.

$$L^{(0)}(v_{ij}) = L(Q_i) = x \tag{9}$$

C—Check Node Update: Update the variable node information to the check node information row-wise according to the Formula (10).

$$L^{(l)}(c_{ji}) = \prod_{i' \in M(j) \setminus i} \operatorname{sgn}(L^{(l-1)}(v_{i'j})) \times \min_{i' \in M(j) \setminus i}(|L^{(l-1)}(v_{i'j})|) \times \alpha \tag{10}$$

Var—Variable Node Update: Update the check node information to the variable node information column-wise according to Formula (11).

$$L^{(l)}(v_{ij}) = L(Q_i) + \sum_{j' \in N(i)/j} L^{(l)}(c_{j'i}) \tag{11}$$

Pos—Posterior Probability Calculation and Decision: Calculate the posterior probability for each variable node and make a decision.

$$L^{(l)}(q_i) = L(Q_i) + \sum_{j \in N(i)} L^{(l)}(c_{ji}) \tag{12}$$

Dec—Decoding Check:

If condition $L^{(l)}(q_i) > 0$ is met, choose option $\hat{x}_i = 0$. Otherwise, choose option $\hat{x}_i = 1$.

Check whether $\left[\hat{x}_1, \hat{x}_2, \hat{x}_3, \cdots, \hat{x}_n\right] * \mathbf{H}^{\mathrm{T}} = 0$ is satisfied. If so, decoding is successful and the decoded output is provided. Otherwise, perform action $l = l + 1$ and return to step 1 until the maximum iteration count $l_{\max}$ is reached, then finish decoding.

The meanings of the symbols used in the formulas are shown in the Table 1 below:

Table 1. Symbol meanings.

Name	Symbol Meanings
$L(Q_i)$	Sequence of information waiting to be decoded
v_i	Variable nodes
c_j	Check nodes
$L^{(l)}(c_{ij})$	Edge information passed from variable node v_i to check node c_j at the l-th iteration
$L^{(l)}(v_{ij})$	Edge information passed from check node c_j to variable node v_i at the l-th iteration
$M(j)$	Set of all variable nodes connected to check node j
$N(i)$	Set of all check nodes connected to variable node i
$M(j) \setminus i$	Set of all variable nodes connected to check node j, excluding variable node i
$N(i) \setminus j$	Set of all check nodes connected to variable node i, excluding check node j

2.2. Structure in the IEEE 802.16e Standard

The IEEE 802.16e standard specifies a maximum code length of 672. In this paper, the focus is on the (672,336) LDPC code. This code starts with 336 bits of original information and generates 336 parity bits through encoding. The code is transmitted over the channel and then decoded iteratively. The IEEE 802.16e protocol includes six different base parity-check matrices, with the base matrix $H_{\mathbf{b}}$ (at a code rate of 1/2) illustrated in the following Figure 2.

$$\begin{bmatrix}
-1 & 94 & 73 & -1 & -1 & -1 & -1 & -1 & 55 & 83 & -1 & -1 & 7 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & 27 & -1 & -1 & -1 & 22 & 79 & 9 & -1 & -1 & -1 & 12 & -1 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & 24 & 22 & 81 & -1 & 33 & -1 & -1 & -1 & 0 & -1 & -1 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
61 & -1 & 47 & -1 & -1 & -1 & -1 & -1 & 65 & 25 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & 39 & -1 & -1 & -1 & 84 & -1 & -1 & 41 & 72 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & 46 & 40 & -1 & 82 & -1 & -1 & -1 & 79 & 0 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & 95 & 53 & -1 & -1 & -1 & -1 & -1 & 14 & 18 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & -1 & -1 & -1 \\
-1 & 11 & 73 & -1 & -1 & -1 & 2 & -1 & -1 & 47 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & -1 & -1 & -1 \\
12 & -1 & -1 & -1 & 83 & 24 & -1 & 43 & -1 & -1 & -1 & 51 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & -1 \\
-1 & -1 & -1 & -1 & -1 & 94 & -1 & 59 & -1 & -1 & 70 & 72 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 \\
-1 & -1 & 7 & 65 & -1 & -1 & -1 & -1 & 39 & 49 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 \\
43 & -1 & -1 & -1 & -1 & 66 & -1 & 41 & -1 & -1 & -1 & 26 & 7 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0
\end{bmatrix}$$

Figure 2. Base parity-check matrix for IEEE 802.16e standard LDPC code at 1/2 code rate.

The parity-check matrix H is derived by expanding the base matrix $N(i)\backslash j$. Different code lengths correspond to different expansion factors z. Depending on these factors, the base parity-check matrix $H_{\mathbf{b}}$ requires updating. Let $q(\mathrm{i},\mathrm{j})$ denote the element at the i-th row and j-th column in the base parity-check matrix. The update to $q(\mathrm{i},\mathrm{j})$ is as follows:

$$q(i,j) = \begin{cases} q(i,j), q(i,j) \leq 0 \\ \lfloor q(i,j) * z/96 \rfloor, q(i,j) > 0 \end{cases} \tag{13}$$

Here, $\lfloor x \rfloor$ represents the floor operation on x. According to the IEEE 802.16e standard, the expansion factor z corresponding to the 672-length LDPC code is 28. Substituting this value, the base parity-check matrix for the 672-length LDPC code remains as depicted in Figure 3.

$$\begin{bmatrix}
-1 & 23 & 18 & -1 & -1 & -1 & -1 & -1 & 13 & 20 & -1 & -1 & 1 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & 6 & -1 & -1 & -1 & 5 & 19 & 2 & -1 & -1 & -1 & 3 & -1 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & 6 & 5 & 20 & -1 & 8 & -1 & -1 & -1 & 0 & -1 & -1 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
15 & -1 & 11 & -1 & -1 & -1 & -1 & -1 & 16 & 6 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & 9 & -1 & -1 & -1 & 21 & -1 & -1 & 10 & 18 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & 11 & 10 & -1 & 20 & -1 & -1 & -1 & 19 & 0 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & 23 & 13 & -1 & -1 & -1 & -1 & -1 & 3 & 4 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & -1 & -1 & -1 \\
-1 & 2 & 18 & -1 & -1 & -1 & 0 & -1 & -1 & 11 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & -1 & -1 \\
3 & -1 & -1 & -1 & 20 & 6 & -1 & 10 & -1 & -1 & -1 & 12 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & -1 \\
-1 & -1 & -1 & -1 & -1 & 23 & -1 & 14 & -1 & -1 & 17 & 18 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 \\
-1 & -1 & 1 & 16 & -1 & -1 & -1 & -1 & 9 & 12 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 \\
10 & -1 & -1 & -1 & -1 & 16 & -1 & 10 & -1 & -1 & -1 & 6 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0
\end{bmatrix}$$

Figure 3. Base parity-check matrix for IEEE 802.16e standard LDPC code with a code length of 672.

The matrix H, expanded from the base parity-check matrix $H_{\mathbf{b}}$, is defined as follows:

$$H = \begin{bmatrix}
I_{0,0} & I_{0,1} & I_{0,2} & \cdots & I_{0,n_g-2} & I_{0,n_g-1} \\
I_{1,0} & I_{1,1} & I_{1,2} & \cdots & I_{1,n_g-2} & I_{1,n_g-1} \\
I_{2,0} & I_{2,1} & I_{2,2} & \cdots & I_{2,n_g-2} & I_{2,n_g-1} \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
I_{m_g-1,0} & I_{m_g-1,1} & I_{m_g-1,2} & \cdots & I_{m_g-1,n_g-2} & I_{m_g-1,n_g-1}
\end{bmatrix} \tag{14}$$

In this structure, $I_{i,j}$ represents a z-by-z zero matrix or a permutation matrix. If the value $q(\mathrm{i},\mathrm{j})$ at the corresponding position in the base parity-check matrix $H_{\mathbf{b}}$ is less than zero, the corresponding position $I_{i,j}$ in the parity-check matrix H will be a zero matrix. Otherwise, $I_{i,j}$ represents a permutation matrix derived by cyclically right-shifting an identity matrix through $q(\mathrm{i},\mathrm{j})$ positions.

3. Improved Decoding Algorithm

3.1. Quantization-Based Correction Factors

When implementing LDPC decoding algorithms on FPGA hardware, it is necessary to convert floating-point data to fixed-point data using quantization. Let the quantization module have a quantization range of $[\theta_{\min}, \theta_{\max}]$, a quantization bit-width of q, and a step size of Δ. The relationship among these parameters can be expressed by the following formula:

$$\theta_{\max} - \theta_{\min} = 2^q \times \Delta \tag{15}$$

After processing through the quantization module, the quantized output sequence can be represented as follows:

$$y_i' = \mathrm{sgn}(y_i)\Delta \left\lfloor \frac{|y_i|}{\Delta} + \frac{1}{2} \right\rfloor \tag{16}$$

Here, y_i' denotes the quantized output sequence, $\lfloor x \rfloor$ represents the floor operation, y_i indicates the output sequence before quantization, and $\mathrm{sgn}(y_i)$ represents the sign of y_i.

Under the influence of AWGN channel noise, the signal range at the input of the decoder is $[-4, 4]$. With 8-bit quantization, the step size is $1/32$ according to Formula (15), resulting in an amplification factor of 32. In this 8-bit quantization, 1 bit is for the sign, 2 bits represent the integer value, and 5 bits represent the fraction.

FPGA excels at performing parallel additions due to its parallel architecture, but more complex operations like look-up tables or successive approximations require significant logic resources. For this reason, in the NMS algorithm's check node information update (Formula (10)), the multiplication by the correction factor α can be approximated using shift-add operations for hardware implementation.

Here is an illustrative example:

Assume the smallest value $L^{(l)}(c_{ji})$ processed by a check node has a binary representation of 00010110. The first bit is the sign bit, while the remaining bits 0010110 represent the absolute value 22. If the bits are cyclically right-shifted by one position, the result is 0001011, representing 11. Thus, right-shifting by one position is equivalent to $L^{(l)}(c_{ji})/2$. Similarly, right-shifting by two positions approximates $L^{(l)}(c_{ji})/4$, and so forth. By using shift-add operations, the multiplication by the correction factor α can be approximated.

For example, to achieve $L^{(l)}(c_{ji})' = L^{(l)}(c_{ji}) \times 0.8$, use $L^{(l)}(c_{ji})' = L^{(l)}(c_{ji}) \times (1/2 + 1/4 + 1/32 + 1/64)$ as an approximation by right-shifting the data bits by one, two, five, and six positions before summing them up. Due to the limited number of quantization bits, each multiplication results in some loss of precision.

3.2. TNMS Decoding Algorithm

Quantization essentially amplifies the channel information of all codewords to be decoded by a consistent factor. Subsequent information derived from the channel data is similarly amplified. Consequently, both the initial channel information $L(Q_i)$ and the variable node edge information $\sum_{j \in N(i)} L^{(l)}(c_{ii})$ in the final codeword decision formula are amplified by the same factor, ensuring that the final decision information $L^{(l)}(q_i)$ remains accurate when determining whether the decoded bit is 0 or 1.

Drawing on this concept, the traditional NMS decoding algorithm has been improved.

After the traditional NMS algorithm initializes the channel information as the variable node information (Formula (9)), a self-update step is added using the correction factor β, following Formula (17):

$$L(Q_i)' = \beta \times L(Q_i) \tag{17}$$

The correction factor α used in the check node update step is removed, replacing Formula (10) with Formula (18):

$$L^{(l)}(c_{ij}) = \prod_{i' \in M(j) \setminus i} \operatorname{sgn}(L^{(l-1)}(v_{i'j})) \times \min_{i' \in M(j) \setminus i}(|L^{(l-1)}(v_{i'j})|) \tag{18}$$

In the decision step, $L(Q_i)$ in Formula (11) is replaced with $L(Q_i)'$ in Formula (19), and consequently, Formula (11) is replaced with Formula (19):

$$L^{(l)}(v_{ij}) = L(Q_i)' + \sum_{j' \in N(i)/j} L^{(l)}(c_{j'i}) \tag{19}$$

In the decision step, $L(Q_i)$ in Formula (12) is replaced with $L(Q_i)'$ in Formula (20), and consequently, Formula (12) is replaced with Formula (20):

$$L^{(l)}(q_i) = L(Q_i)' + \sum_{j \in N(i)} L^{(l)}(c_{ji}) \tag{20}$$

In the original NMS algorithm, the optimal correction factor for the check node formula is α. In the improved decoding algorithm, the optimal correction factor β is set to $1/\alpha$, which yields decoding performance equivalent to using the correction factor α in the original NMS algorithm. For the NMS algorithm, the optimal correction factor selection is influenced by factors like code length, code rate, and signal-to-noise ratio (SNR). Generally, the value ranges from 0.7 to 0.85.

In certain cases, approximating α using shift-add operations may require numerous addition steps, while switching to the correction factor in the improved algorithm can reduce the number of additions required, lowering computation overhead and minimizing logic resource usage.

For example, consider LDPC decoding for a code with length n, check bit length m, and a code rate R = (n − m)/n. In the original NMS algorithm (Formula (10)), if the optimal correction factor α is 0.8, it can be approximated by $L^{(l)}(c_{ji})' = L^{(l)}(c_{ji}) \times (1/2 + 1/4 + 1/32 + 1/64)$. During each iteration, the second minimum and minimum values among the m check nodes are corrected once each, requiring m × 2 × 3 additions and m × 4 × 2 shifts.

With the improved decoding algorithm, in Formula (17), if β is set to $1/\alpha$, i.e., 1.25, it can be approximated using $L(Q_i)' = L(Q_i) \times (1 + 0.25)$. In each iteration, this requires correcting the n channel information values once, which involves n additions and n shifts.

Additionally, due to the quantization bit-width limitation in hardware implementation, the channel information cannot be indefinitely expanded. Therefore, after every three iterations, both the channel information and check node information need to be right-shifted by one bit. This results in an average of (n + 2 × m)/3 shifts per iteration. This process keeps the channel information stable within the quantization range while converting the data loss from correction factors in each iteration of the original NMS algorithm into a single bit-shift of both channel and check node information after every three iterations.

In summary, the improved decoding algorithm requires n additions and (4 × n + 2 × m)/3 shifts per iteration to achieve the correction factor β of 1.25.

As shown in Table 2, for LDPC decoding with a code length n, check bit length m and a code rate R = (n − m)/n:

When the code rate R = 2/3, the optimal α values for the NMS algorithm are 0.73 and 0.8. Transitioning to the TNMS algorithm can reduce computational overhead.

When the code rate R = 1/2, the optimal α values for the NMS algorithm are 0.73, 0.74, 0.79, 0.8, 0.84, and 0.85. Transitioning to the TNMS algorithm can reduce computational overhead.

When the code rate R = 1/3, the optimal α values for the NMS algorithm are 0.72, 0.73, 0.74, 0.78, 0.79, 0.8, 0.83, 0.84, and 0.85. Transitioning to the TNMS algorithm can reduce computational overhead.

When the code rate R = 1/4, the optimal α values for the NMS algorithm are 0.7, 0.71, 0.72, 0.73, 0.74, 0.78, 0.79, 0.8, 0.83, 0.84, and 0.85. Transitioning to the TNMS algorithm can reduce computational overhead.

When the code rate R = 1/5, the optimal α values for the NMS algorithm are 0.7, 0.71, 0.72, 0.73, 0.74, 0.77, 0.78, 0.79, 0.8, 0.82, 0.83, 0.84, and 0.85. Transitioning to the TNMS algorithm can reduce computational overhead.

From this data, it is evident that the lower the code rate, the broader the applicability of the improved decoding algorithm.

Table 2. Difference in operations between NMS and TNMS algorithms.

Optimal α Value for NMS Algorithm	Number of Addition and Shift Operations Required per Iteration for NMS with a Given α Value		Number of Addition and Shift Operations Required per Iteration for TNMS with a Given $\beta = 1/\alpha$ Value		Difference in the Number of Addition and Shift Operations between the Two Algorithms	
0.7	6m	8m	4n	4n + (n + 2m)/3	6m − 4n	(22m − 13n)/3
0.71	6m	8m	4n	4n + (n + 2m)/3	6m − 4n	(22m − 13n)/3
0.72	6m	8m	3n	3n + (n + 2m)/3	6m − 3n	(22m − 10n)/3
0.73	8m	10m	2n	2n + (n + 2m)/3	8m − 2n	(28m − 7n)/3
0.74	8m	10m	3n	3n + (n + 2m)/3	8m − 3n	(28m − 10n)/3
0.75	2m	4m	2n	2n + (n + 2m)/3	2m − 2n	(10m − 7n)/3
0.76	2m	4m	2n	2n + (n + 2m)/3	2m − 2n	(10m − 7n)/3
0.77	4m	6m	3n	3n + (n + 2m)/3	4m − 3n	(16m − 10n)/3
0.78	4m	6m	2n	2n + (n + 2m)/3	4m − 2n	(16m − 7n)/3
0.79	6m	8m	2n	2n + (n + 2m)/3	6m − 2n	(22m − 7n)/3
0.8	6m	8m	n	n + (n + 2m)/3	6m − n	(22m − 4n)/3
0.81	4m	6m	4n	4n + (n + 2m)/3	4m − 4n	(16m − 13n)/3
0.82	4m	6m	3n	3n + (n + 2m)/3	4m − 3n	(16m − 10n)/3
0.83	6m	8m	3n	3n + (n + 2m)/3	6m − 3n	(16m − 10n)/3
0.84	6m	8m	2n	2n + (n + 2m)/3	6m − 2n	(22m − 7n)/3
0.85	6m	8m	2n	2n + (n + 2m)/3	6m − 2n	(22m − 7n)/3

4. Improved LDPC Decoder Hardware Design and Implementation

4.1. Control Module and Overall Architecture

As shown in Figure 4, the overall architecture of the LDPC decoder utilizes the control module to coordinate all other modules. This is achieved in conjunction with the shift address update section, which synchronizes operations across the entire decoder structure. The control module ensures that all modules interact seamlessly, enabling efficient decoding.

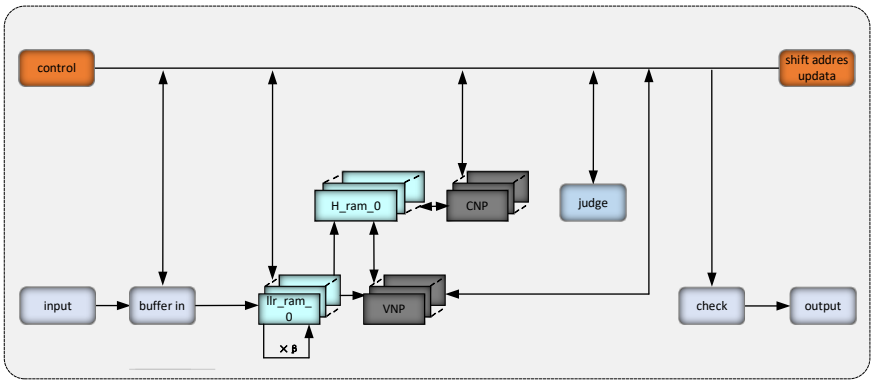

Figure 4. Control module and overall architecture.

First, once the input buffer module receives the start signal, it begins working, reading the 672 initial channel information values into 24 dual-port RAM blocks, known as llr_ram. After all the signals have been read in, the buffer input module sends a completion signal to the control module. At the same time, the control module sends the initial start signal to llr_ram, which then uses the shift address update section to load the channel information into the H_ram group.

After receiving the completion signal from llr_ram, the control module sends start signals to the 12 CNP modules and llr_ram. The 12 CNP modules then begin updating the 336 rows of check node information and return them to the H_ram group. Meanwhile, under read–write signal control, llr_ram completes the self-update of its information using correction factor β.

After receiving the completion signals from the CNP modules, the control module sends operation signals to the 24 VNP modules and the decision module. The 24 VNP modules update the 672 columns of variable node information and return them to the H_ram group. Simultaneously, the decision module reads the initial channel information from llr_ram and the check node information from H_ram. After accumulation, it uses the sign bit to output the final decision.

Once the control module receives the completion signals from the 24 VNP modules and the decision module, it sends a start signal to the verification module. If the verification module's XOR operation yields zero, decoding is successful, and the decoded output is provided. Otherwise, decoding continues to the next iteration.

Every third iteration, the control signal simultaneously sends a signal to shift the channel information in the llr_ram and the check node information in the CNP module to the right by one position.

4.2. Data Storage Section

In this design, the storage modules include llr_ram, which stores the initial channel messages, and H_ram, which stores the confidence message matrix. The quantized channel information is received by the decoder and entered into the 24 llr_ram blocks through the buffer input module.

Based on the structure of the H matrix in the IEEE 802.16e standard for the 672-length LDPC code, the 672 8-bit quantized data values are distributed across 24 dual-port RAMs (each llr_ram). Each RAM block has an 8-bit width and a depth of 32. Every dual-port RAM stores 28 8-bit quantized data values, and the storage structure is illustrated in Figure 5.

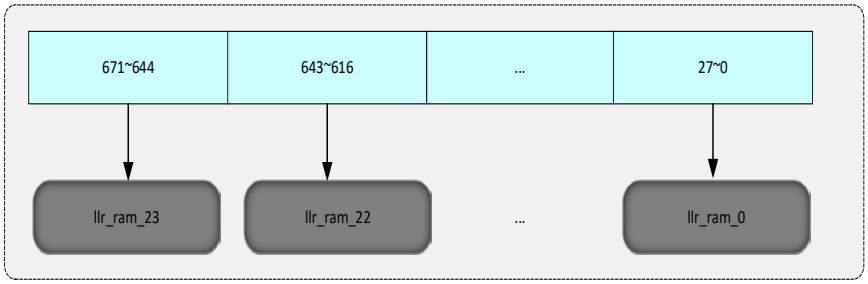

Figure 5. Channel information storage.

To implement the correction factor β, a 1-bit-wide read–write signal is used. With the enable signal, this read–write signal increments by one on each clock cycle. When the enable signal is active, the read–write signal cycles between 0 and 1.

When the read–write signal is 0, a signal at one of the depths in each llr_ram is read. After performing a self-update using the shift-add operation under correction factor β, the updated data is written back into the corresponding llr_ram depth when the read–write signal switches to 1.

All channel information undergoes a self-update within 28 clock cycles for the current iteration. To prevent signal overflow, both channel information and check node information are right-shifted once during the first, fourth, seventh, and every third subsequent iteration.

Each row and column of the sub-matrices in the parity-check matrix H contains at least one non-zero element. Each sub-matrix is a cyclic right-shifted identity matrix, where the amount of shift is determined by the value of the corresponding element in the base parity-check matrix.

This structure allows storing a one-dimensional array in RAM and, in conjunction with the shift address update module, writing/reading it into the corresponding position in the sub-matrix of H as a two-dimensional array.

For instance, if the value at row i and column j in the base matrix is p, then the corresponding sub-matrix in H is a 28 × 28 identity matrix cyclically right-shifted by p positions per row. The H_ram at this position stores 28 data values (from 0 to 27). Each depth s in H_ram represents the value of the non-zero element in row s and column s + p in the sub-matrix, as illustrated in Figure 6.

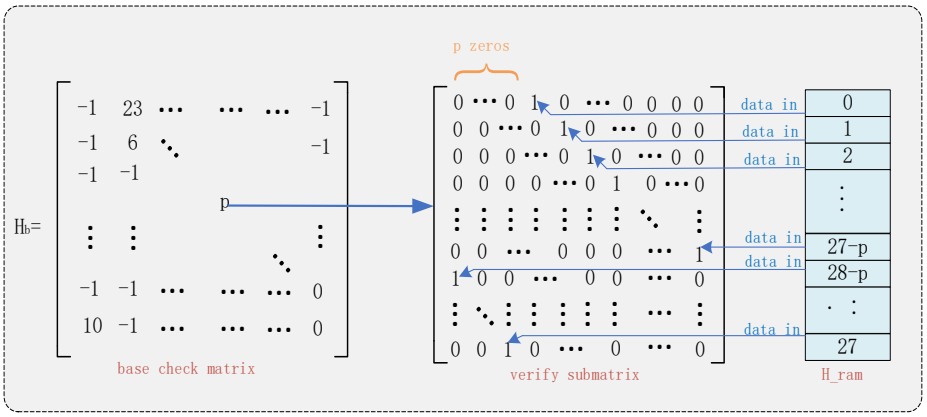

Figure 6. Confidence message matrix storage.

In the RAM array composed of multiple H_ram modules, each H_ram corresponds to an element in the base parity-check matrix H_b. Under the control of the main module and with the assistance of the check node processing (CNP) and variable node processing (VNP) modules, information is continuously exchanged, as illustrated in the Figure 7. If an element in the base matrix equals −1, the corresponding sub-matrix in the H matrix will be a zero matrix and does not need to be stored.

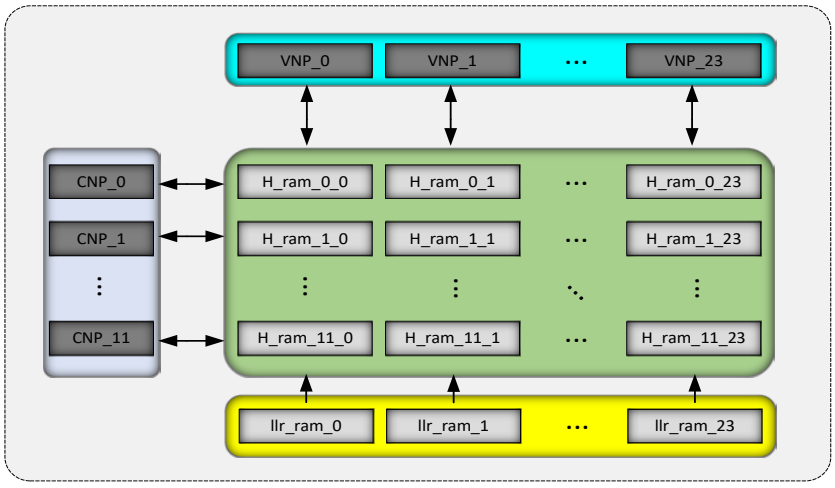

Figure 7. Information exchange between nodes.

According to the base parity-check matrix H_b shown in Figure 3, there are 76 non-negative elements. Thus, 76 H_ram modules are required, each being a dual-port RAM with an 8-bit width and a depth of 31. These will store the information for the non-zero elements in the parity-check matrix.

4.3. Check Node Information Processing Module

As shown in Figure 3, the base parity-check matrix H_b contains eight rows with six non-negative elements each and four rows with seven non-negative elements each. Thus, eight six-input, six-output check node processing units and four seven-input, seven-output check node processing units are required.

After receiving the enable signal from the control module, each check node processing unit simultaneously reads the variable node messages stored at the same address in the H_ram matrix, corresponding to their respective row in the base matrix. Once processing is complete, the outputs are updated as check node information. Every two clock cycles, the dual-port RAM completes a read operation and a write operation, and the storage address increments by one. In 56 clock cycles, data in all 28 depths of H_ram is updated.

After all 12 check node processing units have updated their 336 check node messages, the check node module sends a completion signal to the control module.

The design of the check node processing units is shown in Figure 8. Each input is divided into a sign bit and data bits. The sign bits from all inputs undergo an XOR operation to determine the sign bit of each output. Simultaneously, the data bits of each input are compared using a divide-and-conquer approach to determine the least and second-smallest values.

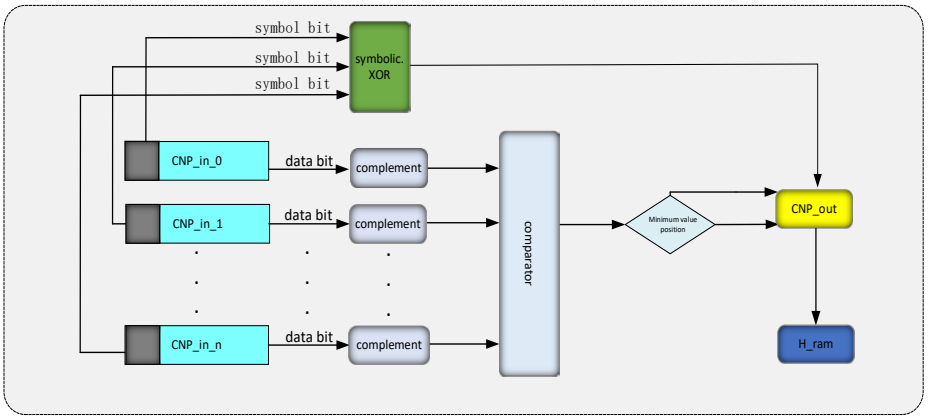

Figure 8. Check node processing unit.

Every third iteration, the least and second-smallest values output by the check node processing units are right-shifted once before being output. The data bits of each input are compared with the smallest value. If they match, the second-smallest value is concatenated with the computed sign bit and output as the check node information at the corresponding depth in H_ram. Otherwise, the smallest value is concatenated with the sign bit and output.

4.4. Variable Node Information Processing Module and Decision Module

According to Figure 3, the base parity-check matrix H_b includes 8 columns with three non-negative elements each, 11 columns with two non-negative elements each, and 5 columns with six non-negative elements each. Each variable node processing unit handles not only the messages passed through the confidence message matrix but also includes one updated channel data input and one decision signal output. Therefore, the design requires 8 four-input, four-output variable node processing units, 11 three-input, three-output units, and 5 seven-input, seven-output units.

After receiving the enable signal from the control module and in conjunction with the shift address update module, each variable node processing unit reads the check node

messages stored at the same address in H_ram for their respective columns in the base matrix, along with the channel information from llr_ram at the corresponding position.

After processing, the updated variable node messages are returned to the corresponding column in H_ram, and the processed decision information is output. Every two clock cycles, the dual-port RAM completes a read-and-write operation, incrementing the storage address. Within 56 clock cycles, all decision information is output, and the 28-depth data in all H_ram blocks is updated.

After all 24 variable node processing units have updated the 672 rows of variable node information, the decision information is also fully output, and both the variable node and decision modules simultaneously send completion signals to the control module.

The design of the variable node processing unit is depicted in Figure 9. The updated channel data and check node information read from the column are summed up using an addition tree. To prevent overflow due to excessively high input amplitudes during addition, a limiter module prevents result overflow. The sign bit of the sum result is used as the decision output. The sum result is subtracted from each check node message input using two's complement addition, and the limited output is returned to the corresponding H_ram depth as the variable node information.

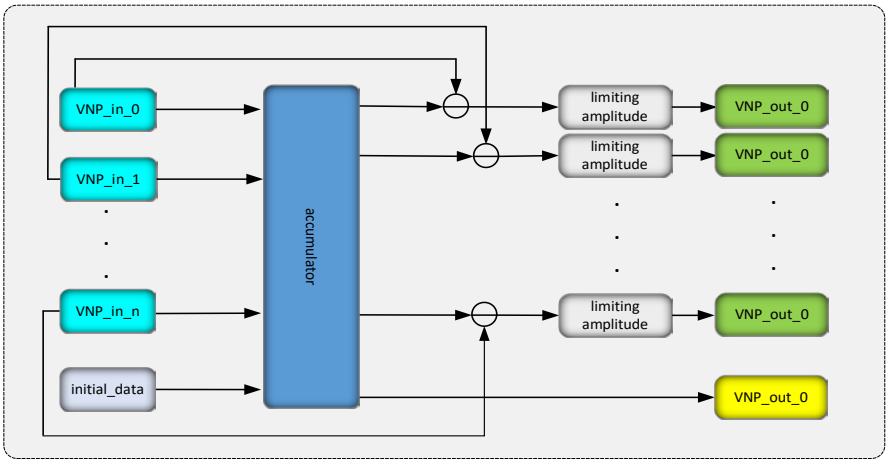

Figure 9. Variable node processing unit.

4.5. Verification Module

After the decision module completes its work, it outputs 672 codewords that are stored in 24 shift registers. When the control module sends an enable signal to the verification module, it retrieves the codeword positions corresponding to the non-zero elements of each row in the parity-check matrix. Using XOR operations, the module multiplies the decision codewords with the row vectors of the parity-check matrix.

Each clock cycle performs row vector multiplication for 12 rows, and all row vector multiplications are completed within 28 clock cycles. The outcomes of the XOR operations are documented. If all results are zero, decoding is successful. A decoding success signal is sent, and the first 336 bits are extracted as the decoded output.

If the sum of XOR results is not zero, a decoding failure signal is sent, the iteration count is incremented, and the next decoding iteration is initiated.

4.6. Comparison before and after Improvement

Originally, the NMS decoder required each check node's minimum and second minimum values to undergo correction using factor $\alpha = 0.8$ in the check node information processing module. This involved shifting the minimum values right by 1, 2, 5, and 6 bits, respectively, and then summing them to achieve correction, with the second minimum values processed similarly. In each iteration, for the 336 check nodes, this resulted in a total of 2688 shift operations and 2016 addition operations.

The improved decoder, however, eliminates the correction of the minimum and second minimum values using factor $\alpha = 0.8$ in each check node. Instead, it applies correction factor $\beta = 1.25$ in the variable node information processing by shifting each input channel information right by one bit and adding it back to the original pre-shift information before updating the variable nodes. To prevent signal data overflow due to limited data width, every first, fourth, seventh, etc., iteration—every third iteration—the channel information and check node information are both shifted right by one bit together.

In each iteration, the improved decoder requires a total of 1120 shift operations and 672 addition operations for the correction factor $\beta = 1.25$. Compared to the original 2688 shifts and 2016 additions, this undoubtedly reduces logic resource usage and power consumption. Since both correction factor processes before and after improvement involve shift-add combination logic, they do not impact timing, and therefore, the decoding operation speed remains unaffected.

5. Testing and Validation

This paper employs LDPC decoding based on the IEEE 802.16e standard, using BPSK modulation and transmission through an AWGN channel, with a maximum of 30 iterations. Simulations are performed for LDPC codes with a code rate of 1/2 and lengths of 576, 672, and 1440, as well as a code rate of two-thirds for a length of 1248. Frame error statistics are collected, and the simulation terminates when the number of erroneous frames reaches a preset value, at which point the bit error rate is calculated.

With a target of 1000 erroneous frames, simulations were conducted on the traditional NMS algorithm with correction factor α ranging from 0.7 to 0.85, accurate to two decimal places, for four different code lengths and rates of LDPC codes. Specifically, simulations for LDPC codes of lengths 576 and 672 were performed at a signal-to-noise ratio (SNR) of 1.8, for a code length of 1248 at an SNR of 1.6, and for a code length of 1440 at an SNR of 1.5. The simulation results, as depicted in Figure 10, confirmed that the optimal α value is 0.78 for a code length of 576 at a 1/2 code rate, 0.8 for code lengths of 672 and 1440 at a 1/2 code rate, and 0.76 for a code length of 1248 at a two-thirds code rate.

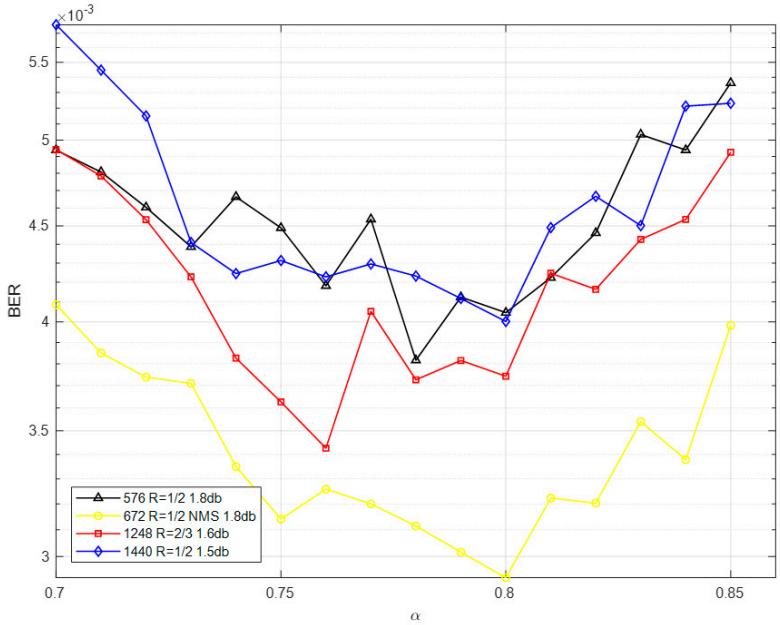

Figure 10. Optimal α value measurement for four different LDPC codes.

For LDPC codes with lengths of 576, 672, 1248, and 1440, comparative simulations were conducted between the traditional NMS algorithm with optimal α values and the TNMS algorithm with $\beta = 1/\alpha$, under various code lengths, rates, and optimal α values.

As illustrated in Figure 11, it was observed that under all these differing conditions, the decoding performance of both algorithms is fundamentally similar.

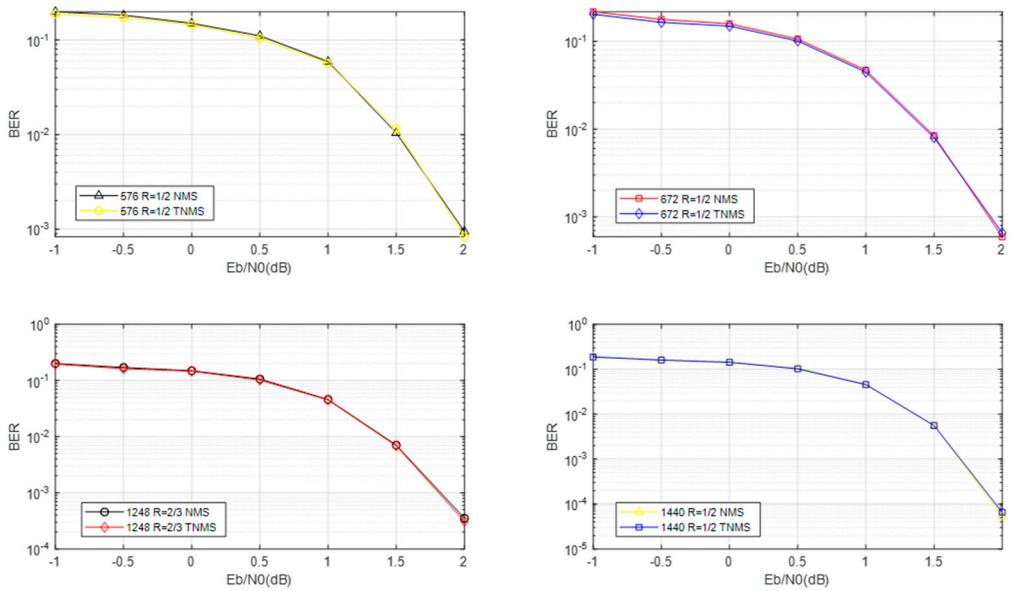

Figure 11. Performance comparison of NMS and TNMS algorithms across four different code lengths and rates.

The LDPC decoder design was completed using Verilog HDL on the Vivado software platform, implementing an improved algorithm based on the IEEE 802.16e 1/2-rate base matrix with a code length of 672. The simulation results, shown in Figure 12, indicate successful decoding after 10 iterations. Comparing the decoded codeword sequence with the pre-encoded sequence shows no errors in the decoded output.

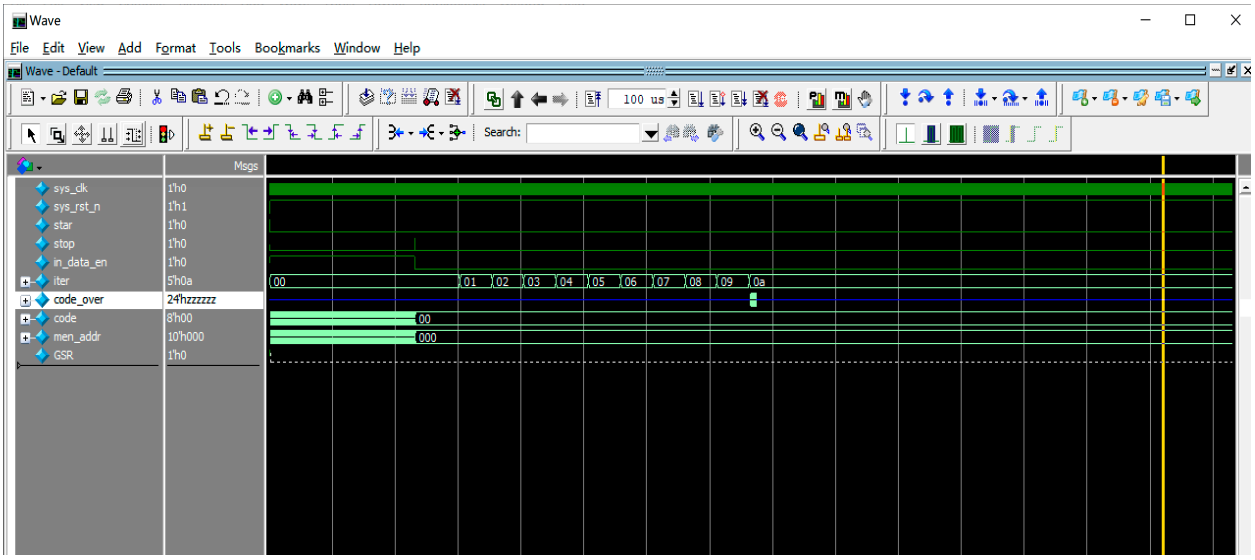

Figure 12. FPGA decoder simulation diagram for the improved algorithm.

The TNMS decoder and the original NMS decoder were compiled using the Vivado software platform, and the comparison of their logic resource utilization is shown in Figure 13. The comparison of these resource utilization reports demonstrates that the improved decoder has reduced logic resource consumption relative to the original.

Primitive type	Count	Primitive type	Count
FLOP_LATCH	1943	FLOP_LATCH	1943
LUT	20118	LUT	19835
MUXFX	230	MUXFX	230
CARRY	2004	CARRY	2004
IO	49	IO	49
BMEM	50	BMEM	50
CLK	1	CLK	1

Figure 13. Comparison of logic resource utilization between NMS decoder and improved TNMS decoder.

6. Conclusions

This paper introduces a transfer correction factor-based normalized Min-Sum algorithm as an improvement on the traditional NMS algorithm, aiming to reduce computational overhead without compromising performance. The proposed algorithm modifies the correction factor α used in the traditional NMS decoding algorithm for check node information to a new correction factor β for channel information.

The principle is based on quantization, and the MATLAB simulation platform confirmed that the two algorithms perform almost identically at $\beta = 1/\alpha$. However, under different code lengths, rates, and other conditions, the optimal correction factor, α, varies for the NMS algorithm. In cases where the best α value requires too many shifts and additions but $\beta = 1/\alpha$ requires only a few, it can be converted into the improved decoding algorithm, thereby achieving the goal of reducing computational overhead without compromising performance.

Based on the IEEE 802.16e standard, an FPGA decoder for an LDPC code with a code rate of $1/2$ and a length of 672 was designed using the Vivado software simulation platform. This design validated the feasibility of the algorithm and confirmed the reduction in logic resource consumption through resource utilization reports.

The improved decoding algorithm offers a broader range of applications at lower code rates. Taking the 3GPP standards for 5G LDPC encoding as an example, which includes two base matrices, BG1 and BG2, BG1 is designed for information bits ranging from 500 to 8448, with code rates from one-third to eight-ninths, while BG2 is tailored for information bits ranging from 40 to 2560, with code rates from one-fifth to two-thirds. The enhanced decoding algorithm is particularly advantageous for the latter, which is used in scenarios requiring the transmission of text and images at low code rates and short lengths, where there is a greater demand for minimal logic resource usage, such as in digital watermarking and route planning applications.

Author Contributions: Writing—original draft preparation, H.-Y.W.; supervision, Z.-X.W.; methodology, S.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors on request.

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

References

1. Gallager, R.G. Loiv-Density Parity-Check Codes. *Inf. Theory IRE Trans.* **1962**, *8*, 21–28. [CrossRef]
2. MacKay, D.J.; Neal, R.M. Near Shannon limit performance of low density parity check codes. *Electron. Lett.* **1996**, *33*, 457–458. [CrossRef]
3. Davey, M.C. Error-Correction using Low Density Parity Check Codes. Ph.D. Thesis, University of Cambridge, Cambridge, UK, 1999.
4. Davey, M.C.; Mackay, D. Low-density parity check codes over GF(q). *Commun. Lett. IEEE* **1998**, *2*, 165–167. [CrossRef]
5. Richardson, T.J.; Urbanke, R.L. The Capacity of Low-Density Parity-Check Codes Under Message-Passing Decoding. *IEEE Trans. Inf. Theory* **2001**, *47*, 599–618. [CrossRef]
6. Kschischang, F.R.; Frey, B.J.; Loeliger, H.A. Factor graphs and the sum-product algorithm. *IEEE Trans. Inf. Theory* **2001**, *47*, 498–519. [CrossRef]
7. Eleftheriou, E.; Mittelholzer, T.; Dholakia, A. Reduced-complexity decoding algorithm for low-density parity-check codes. *Electron. Lett.* **2001**, *37*, 102–104. [CrossRef]
8. Zhou, W.; Lentmaier, M. Generalized Two-Magnitude Check Node Updating with Self Correction for 5G LDPC Codes Decoding. In Proceedings of the 12th International ITG Conference on Systems, Communications and Coding, Rostock, Germany, 11–14 February 2019. [CrossRef]
9. Cui, H.; Ghaffari, F.; Le, K.; Declercq, D.; Lin, J.; Wang, Z. Design of High-Performance and Area-Efficient Decoder for 5G LDPC Codes. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2021**, *68*, 879–891. [CrossRef]
10. Wu, X.; Song, Y.; Jiang, M.; Zhao, C. Adaptive-Normalized/Offset Min-Sum Algorithm. *IEEE Commun. Lett.* **2010**, *14*, 667–669. [CrossRef]
11. Le Trung, K.; Ghaffari, F.; Declercq, D. An Adaptation of Min-Sum Decoder for 5G Low-Density Parity-Check Codes. In Proceedings of the 2019 IEEE International Symposium on Circuits and Systems (ISCAS), Sapporo, Japan, 26–29 May 2019. [CrossRef]
12. Savin, V. Self-Corrected Min-Sum decoding of LDPC codes. In Proceedings of the 2008 IEEE International Symposium on Information Theory, Toronto, ON, Canada, 6–11 July 2008. [CrossRef]
13. Ren, Y.; Harb, H.; Shen, Y.; Balatsoukas-Stimming, A.; Burg, A. A Generalized Adjusted Min-Sum Decoder for 5G LDPC Codes: Algorithm and Implementation. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2024**, *71*, 2911–2924. [CrossRef]
14. Marchand, C.; Boutillon, E. LDPC decoder architecture for DVB-S2 and DVB-S2X standards. In Proceedings of the 2015 IEEE Workshop on Signal Processing Systems (SiPS), Hangzhou, China, 14–16 October 2015. [CrossRef]
15. Degardin, V.; Lienard, M.; Zeddam, A.; Gauthier, F.; Degauquel, P. Classification and characterization of impulsive noise on indoor powerline used for data communications. *IEEE Trans. Consum. Electron.* **2002**, *48*, 913–918. [CrossRef]
16. Lu, Q.; Sham, C.W.; Lau, F.C. Rapid prototyping of multi-mode QC-LDPC decoder for 802.11n/ac standard. In Proceedings of the 2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC), Macao, China, 25–28 January 2016. [CrossRef]