Column-Weighted Probabilistic GDBF Decoder for Irregular LDPC Codes

Changfu He, Keyue Deng, Suwen Song and Zhongfeng Wang

August 1, 2023

Column-Weighted Probabilistic GDBF Decoder for Irregular LDPC Codes

Changfu He, Keyue Deng, Suwen Song, and Zhongfeng Wang
School of Electronic Science and Engineering, Nanjing University, Nanjing, China
Email: 191180045@smail.nju.edu.cn, kydeng@smail.nju.edu.cn, suwsong@sina.com, zfwang@nju.edu.cn

***Abstract*—Existing bit-flipping algorithms, when used for irregular low-density parity-check (LDPC) codes, often suffer from performance degradation due to the imbalance caused by the irregular codeword structure. To alleviate this problem, this paper presents a column-weighted probabilistic gradient-descent bit-flipping (CW-PGDBF) decoder for irregular LDPC codes. Different weighting factors are allocated to variable nodes with different column weights to solve the imbalance. Furthermore, a modified probabilistic flipping rule is employed to reduce the hardware complexity while maintaining the error-correction performance. Simulation results demonstrate that the proposed algorithm can achieve significantly improved performance compared with other BF-based algorithms for irregular codes. Additionally, a hardware architecture for the CW-PGDBF decoder is proposed with acceptable hardware overhead compared to the original PGDBF decoder.**

***Index Terms*—Low-density parity-check codes, irregular codes, bit-flipping, column-wise weighting factor, hardware implementation.**

I. Introduction

Low-density parity-check codes have become increasingly popular since the introduction by Gallager in 1962 [1]. Featuring near Shannon limit performance and low hardware overhead under iterative decoding, LDPC codes have been widely used in the field of digital communication and storage applications, such as IEEE802.16e, IEEE802.11n, and DVB-S2 standards. The decoding algorithms can generally be divided into two categories, soft-decision and hard-decision ones. Soft-decision decoding shows great decoding performance, including Belief Propagation (BP) algorithm [1] and its simplified algorithms such as Min-Sum (MS) algorithm [2] and Offset MS (OMS) algorithm [3]. Nevertheless, soft-decision decoding algorithms are far more resource-consuming than hard-decision decoding algorithms, such as Bit-Flipping (BF) algorithms [4]–[11], which only utilize binary messages instead of floating-point/quantized messages.

The BF algorithms lead to less hardware overhead but obvious performance degradation. To mitigate this problem, the gradient descent BF (GDBF) algorithm [7] was proposed by adding an extra term to involve the similarity between the received bit and the current decoded bit into the inversion function. In [8], the probabilistic GDBF (PGDBF) decoder was proposed, in which the bits with the satisfactory inversion value are flipped with a probability, playing a certain role in helping the decoding escape from the trapping sets of LDPC codes. To decrease the critical path while keeping the performance, the probabilistic parallel BF (PPBF) algorithm [9] was proposed by flipping each bit with different probabilities depending on their reliabilities. Recently, Cui *et. al.* have proposed the Tabu-list random-penalty GDBF (TRGDBF) algorithm [10], which adds a random penalty to help step out of the trapping sets and a tabu-list to avoid flipping the bits repeatedly, greatly improving the error-correction performance. Furthermore, to shorten the critical path induced by the maximum-finding operation, an information storage BF (ISBF) algorithm [11] was proposed, storing the information of the previous iteration and using them to perform maximum-finding operations in parallel with other operations.

However, the algorithms mentioned above are all designed for regular codes, instead of irregular codes. The irregular codes were proposed by Luby *et. al.* and shown the capability of yielding much better performance than regular codes [12], [13]. If the algorithms dedicated to regular codes are directly applied to irregular codes without any changes, there will be a huge performance loss. In [14], the adaptive GDBF diversity decoder (AD-GDBF) was proposed, combining the GDBF-based decoders with different parameters into a unified diversity decoder by using genetic algorithm optimization and outperforming all existing probabilistic bit-flipping decoders. Although the AD-GDBF algorithm can be applied to irregular codes, its complex framework demands significant computational resources and time for training and parameter optimization. To design a more lightweight hard-decision decoding algorithm for irregular codes, a new GDBF decoder is proposed, which combines column-wise weighting factors and a modified probabilistic flipping rule with the GDBF decoder. The main characteristic of the CW-PGDBF decoder is that a column-degree-based weighting factor is introduced when calculating the inversion values, to make their maximum magnitudes consistent for different codeword bits, so that the regular decoding algorithms can be applied. A modified probabilistic flipping rule is also employed to reduce the hardware overhead while maintaining the error-correction performance. Meanwhile, a well-optimized architecture is implemented. The experiment results show that compared with other hard-decision decoders, our proposed decoder performs better in error-correction performance with acceptable hardware overhead.

The rest of this paper is organized as follows. Section II introduces the notations and the existing PGDBF algorithm. In Section III, we detailedly describe the proposed CW-PGDBF algorithm. Section IV displays the simulation results for our algorithm, including the comparison with other algorithms and the selection of parameters. In Section V, the hardware architecture and synthesis results are presented. Finally, we draw a conclusion in Section VI.

II. Preliminaries

A. Notations

The $M \times N (N > M)$ parity-check matrix is denoted as $\boldsymbol{H}$, where M rows correspond to M parity-check equations and N columns correspond to N channel bits. A codeword is a binary vector $v_i \in \{0, 1\}, i = 0, 1, \cdots, N-1$, that satisfies all parity-check equation, i.e. $\boldsymbol{H}\boldsymbol{v}^T = 0$. Suppose that the decoder is specially designed for Binary Symmetric Channel (BSC), and the output of the channel is $y_i \in \{0, 1\}, i = 0, 1, \cdots, N-1$. A common representation of LDPC codes is the Tanner graph, which is composed of M check nodes (CNs) and N variable nodes (VNs). $\mathcal{N}(i)$ is the set of indices of VNs that is connected to the i-th CN, and $\mathcal{M}(j)$ is the set of indices of CNs that is connected to the j-th VN. For example, $\mathcal{N}(i) = \{n \mid h_{i,n} = 1, n \in [1, N]\}$, and $\mathcal{M}(j) = \{m \mid h_{m,j} = 1, m \in [1, M]\}$. $|\mathcal{N}(i)|$ denotes the degree of the i-th row and $|\mathcal{M}(j)|$ denotes the degree of the j-th column. For clarity, we also use d_v to represent the degree of each column in this paper.

B. The PGDBF Algorithm

A BF decoder updates the variable nodes iteratively. Assume that the n-th bit in the t-th iteration is $v_n^{(t)}$, and the m-th parity-check value in the t-th iteration is $s_m^{(t)}$, which is calculated by:

$$s_m^{(t)} = \bigoplus_{n \in \mathcal{N}(i)} v_n^{(t)}, \tag{1}$$

where $\oplus$ means exclusive-OR (XOR) operation on all of the codeword bits. If $\boldsymbol{s}^{(t)}$ satisfies the parity checks, the decoding will declare success. Otherwise, the next iteration of decoding will be carried out. The decoding will be terminated when t reaches the pre-defined maximum number of iterations T_{max}, and at this time if the parity checks are still not satisfied by $\boldsymbol{s}^{(t)}$, a decoding failure will be declared.

At the t-th iteration, each VN first calculates the inversion value $E_{v_i}^{(t)}$ using Eq. (2):

$$S_{v_i}^{(t)} = \sum_{m \in \mathcal{M}(i)} s_m^{(t)}, \tag{2a}$$

$$E_{v_i}^{(t)} = v_i^{(t)} \oplus y_i + S_{v_i}^{(t)}, \tag{2b}$$

where the first term in $E_{v_i}^{(t)}$ means the correlation between the updated codeword and the received word and the latter one is the sum of parity-check values which are connected to the corresponding VN, regarded as a penalty term that forces $\mathbf{v}$ to be a correct codeword. A large $E_{v_i}^{(t)}$ could mean that the correlation is weak, or there exist many unsatisfied parity-check equations, so the bits with the maximum $E_{v_i}^{(t)}$, i.e. $E_{max}^{(t)} = \max(E_{v_i}^{(t)})$, should be flipped to correct the code.

In PGDBF, instead of flipping all the bits with inversion value $E_{max}^{(t)}$, only a random fraction of them are flipped, depending on the probability p, which is selected before decoding. After that, the updated values of all VNs are sent to the next iteration until they satisfy the termination conditions mentioned above.

However, compared with regular LDPC codes, the structure of irregular LDPC codes is more complex. For variable nodes with different degrees in irregular LDPC codes, the PGDBF algorithm treats them equally, leading to unfair inversion values. Moreover, the decoding of irregular codes requires far more iterations to achieve better decoding performance. Therefore, a huge performance loss will be caused if continuing to use the PGDBF algorithm or other variants to decode irregular LDPC codes.

III. Proposed CW-PGDBF Algorithm

To minimize the performance loss as much as possible, the CW-PGDBF algorithm is proposed for the decoding of irregular LDPC codes in this section. We focus our attention on the different parts of the proposed algorithm from the PGDBF algorithm, including the newly introduced column-wise weighting factor and the modified probabilistic flipping rule.

A. Column-Wise Weighting Factor

Suppose that we are using an irregular $\boldsymbol{H}$ containing only two column degrees, i.e. 4986.93xb.329 [15], which has $d_{v1} = 3$ in the first 9141 columns and $d_{v2} = 9$ in the other 831 columns. If we use a regular decoding algorithm such as PGDBF, the $E_{max}^{(t)}$ of d_{v1} part can only reach a maximum of 4, and the d_{v2} part reach 10. Thus, as long as the $E_{max}^{(t)}$ is larger than 4, the d_{v1} part can never be flipped. In other words, the d_{v1} part has less chance of being flipped than the d_{v2} part, which will cause a huge loss in decoding performance.

Therefore, we consider multiplying the second term of each inversion value by a column-wise weighting factor so that the new ones are in the same range. For example, through multiplying the d_{v1} part by w_1 and the d_{v2} part by w_2, the range of the d_{v1} part is $0 \sim w_1 \times d_{v1}$ and the d_{v2} part is $0 \sim w_2 \times d_{v2}$. Accordingly, the weighted inversion values turn into Eq. (3):

$$E_{v_i}^{(t)} = \begin{cases} v_i^{(t)} \oplus y_i + w_1 \times S_{v_i}^{(t)}, & \text{for } d_v = d_{v1}, \quad (3a) \\ v_i^{(t)} \oplus y_i + w_2 \times S_{v_i}^{(t)}, & \text{for } d_v = d_{v2}. \quad (3b) \end{cases}$$

If we select w_1 and w_2 appropriately to achieve the equation $w_1 \times d_{v1} = w_2 \times d_{v2}$, the weighted variable nodes will have a fairer chance to be flipped. For our chosen $\boldsymbol{H}$ 4986.93xb.329, we have tried different combinations of w_1 and w_2, and finally, we select $w_1 = 3$ and $w_2 = 1$ to be the weighting factors. For other $\boldsymbol{H}$s consisting of two degrees, w_1 and w_2 can be constructed in the same way.

B. Modified Probabilistic Flipping Rule

Obviously, after column-weighting, the bit-width of each inversion value may increase, which requires additional hardware overhead when executing the maximum-finding operation. To avoid this problem, we improve our algorithm by finding the maximum value before column-weighting. Moreover, instead of finding the maximum value of $E_{vi}^{(t)}$, we just find the maximum value of its second term for d_{v1} and d_{v2} part respectively, as in Eqs. (4-5). This can also save hardware resources to some extent, which will be mentioned in Section V. Then we can find the entire maximum by column-weighting the two maximum values and making comparisons between them, as in Eq. (6).

$$S_{max1}^{(t)} = \max\{S_{v_i}^{(t)}\},\ i \in \{i \mid degree_i = d_{v1}\}, \tag{4}$$

$$S_{max2}^{(t)} = \max\{S_{v_j}^{(t)}\},\ j \in \{j \mid degree_j = d_{v2}\}, \tag{5}$$

$$E_{max}^{(t)} = \max\{\mathbf{w_1} S_{max1}^{(t)}, \mathbf{w_2} S_{max2}^{(t)}\}. \tag{6}$$

It should be noted that the $E_{max}^{(t)}$ we just attained may not be the exact maximum inversion value, on account that we actually ignore the term $v_i^{(t)} \oplus y_i$ when finding the maximum values, so the exact maximum inversion value may be equal to or one more than $E_{max}^{(t)}$. Given this, we assign two probabilities to control the flipping.

- If the inversion value is equal to $E_{max}^{(t)}$, then flip the corresponding bit with probability p_1.
- If the inversion value is one more than $E_{max}^{(t)}$, then flip the corresponding bit with probability p_2.
- If the inversion value is smaller than $E_{max}^{(t)}$, then no flipping operation is needed.

To get the optimal p_1 and p_2, we can try different combinations of (p_1, p_2) and find the best one.

C. Other Irregular Codes

Up till now, we have considered the $\boldsymbol{H}$ with only two degrees. However, there exist many irregular codes with more than two degrees, such as PEGirReg504x1008 [15], which has degrees of $2, 3, 4, 5, 7, 14, 15$. In this case, the weighting factor has to be the degrees' least common multiple (LCM) divided by each degree. Take the PEGirReg504x1008 code for example, the degrees $2, 3, 4, 5, 7, 14, 15$ have LCM $= 420$, so they are relatively multiplied by $210, 140, 105, 84, 60, 30, 24$. The weighting factors are so large that the hardware overhead will be unimaginably huge.

We solve this problem by dividing the degrees into two groups and treating each degree in the same group equally. For example, we can divide the degrees $2, 3, 4, 5$ into the first group and $7, 14, 15$ into the other one, accordingly the weighting factors can be 3 and 1 to make the range as close as possible. Similarly, we can also divide the degrees $2, 3$ into the first group and $4, 5, 7, 14, 15$ into the other one, and the weighting factors are 5 and 1. We finally choose the former one, which shows better decoding performance. For other irregular codes containing more than two degrees, the same idea can also be adopted.

Finally, the proposed CW-PGDBF algorithm is detailedly described in Alg.1.

Algorithm 1 The CW-PGDBF Algorithm.

```
Input       $\mathbf{y} = (y_0, y_1, \cdots, y_{N-1})$
Parameter   $(w_1, w_2)$, $(p_1, p_2)$, $group_1$, $group_2$
Initialize  $v_n^{(0)} \leftarrow y_n,\ n = 0, \cdots, N-1$
for $t = 0$ to $T_{max} - 1$ do
    for $i = 0$ to $M - 1$ do
        $s_i = \bigoplus_{k \in \mathcal{N}(i)} v_k^{(t)}$
    end for
    if s = 0 then break
    Generate $R_n^{(t)} \sim \mathcal{B}(p_1),\ n = 1, \cdots, N$
    Generate $Q_n^{(t)} \sim \mathcal{B}(p_2),\ n = 1, \cdots, N$

    for $i \in \{i \mid degree_i \in group_1\}$ do
        $S_{v_i}^{(t)} = \sum_{m \in \mathcal{M}(i)} s_m^{(t)}$
        $E_{v_i}^{(t)} = v_i^{(t)} \oplus y_i + w_1 S_{v_i}^{(t)}$
    end for
    for $j \in \{j \mid degree_j \in group_2\}$ do
        $S_{v_j}^{(t)} = \sum_{m \in \mathcal{M}(j)} s_m^{(t)}$
        $E_{v_j}^{(t)} = v_j^{(t)} \oplus y_j + w_2 S_{v_j}^{(t)}$
    end for

    $S_{max1}^{(t)} = \max\{S_{v_i}^{(t)}\},\ i \in \{i \mid degree_i \in group_1\}$
    $S_{max2}^{(t)} = \max\{S_{v_j}^{(t)}\},\ j \in \{j \mid degree_j \in group_2\}$
    $E_{max}^{(t)} = \max\{w_1 S_{max1}^{(t)}, w_2 S_{max2}^{(t)}\}$

    for $i = 0$ to $N - 1$ do
        if $E_{v_i}^{(t)} = E_{max}^{(t)}$ then
            $v_i^{(t+1)} = v_i^{(t)} \oplus R_i^{(t)}$
        else if $E_{v_i}^{(t)} = E_{max}^{(t)} + 1$ then
            $v_i^{(t+1)} = v_i^{(t)} \oplus Q_i^{(t)}$
        end if
    end for
end for
output $\mathbf{v}^{(k)}$
```

IV. Simulation Results

This section presents the decoding performance of the CW-PGDBF algorithm and compares it with PGDBF, TRGDBF, and ISBF. The considered irregular LDPC codes are the 4986.93xb.329 and PEGirReg504x1008. The T_{max} is set to 300, and the optimal p_1 and p_2 values are found through numerical search. For the 4986.93xb.329 code, we vary both p_1 and p_2 from 0.1 to 1.0 with a step size of 0.1, and test the error-correction performance under $\alpha = 0.035$. We mark the total of 100 cases of (p_1, p_2) with an index i $(0 \leq i \leq 99)$, plotted in Fig. 1.

For the code 4986.93xb.329, the optimal values of the parameters are selected to be $p_1 = 0.50$ and $p_2 = 1.00$. It is worth noting that $p_2 = 1.00$ is also optimal for other codewords, which is not surprising since the maximum inversion bits are expected to be flipped. It eliminates the need for additional hardware resources such as a random generator module. The error-correction performance of the proposed algorithm on the code 4986.93xb.329, using the chosen (p_1, p_2), is compared with that of the PGDBF, TRGDBF, and ISBF algorithms in Fig. 2. As α decreases, both the TRGDBF and ISBF algorithms experience an error floor phenomenon. In contrast, the CW-PGDBF algorithm shows better error-correction performance than other algorithms and does not

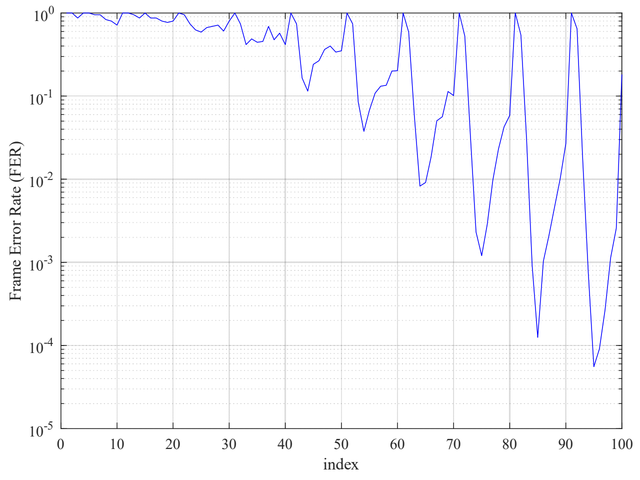

Fig. 1. Numerical search of (p_1, p_2) for 4986.93xb.329 under $\alpha = 0.035$.

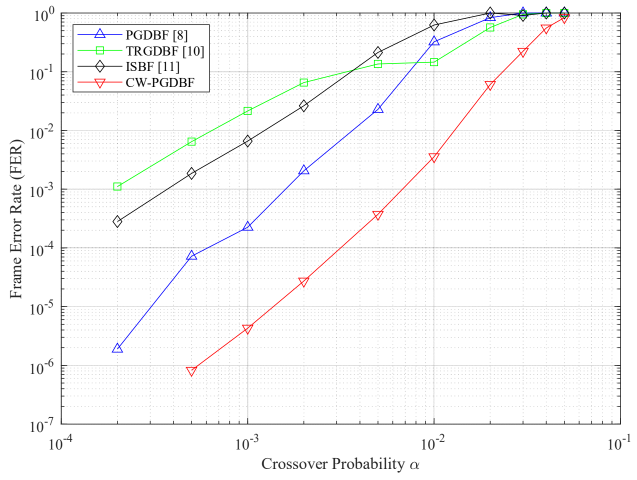

Fig. 3. FER of CW-PGDBF over BSC channel using PEGirReg504x1008.

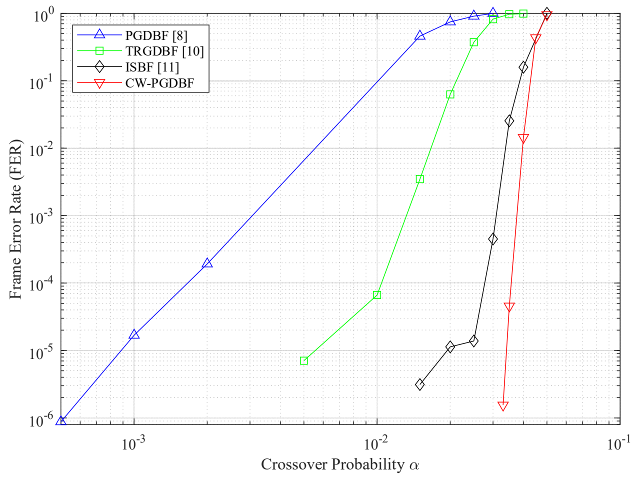

Fig. 2. FER of CW-PGDBF over BSC channel using 4986.93xb.329.

experience an error floor, which increases the robustness of the decoding process.

When decoding irregular LDPC codes with more than two degrees, we select PEGirReg504x1008 as an example. The degrees in this code are $2, 3, 4, 5, 7, 14, 15$, and we divided them into two groups: $2, 3, 4, 5$ in one and $7, 14, 15$ in the other. The optimal (p_1, p_2) was selected to be $(1.00, 1.00)$. We then compared the performance of CW-PGDBF with PGDBF, TRGDBF, and ISBF on PEGirReg504x1008, as shown in Fig. 3, and found that CW-PGDBF outperformed the other algorithms. As a result, we can say that the CW-PGDBF algorithm is a good choice when decoding irregular LDPC codes.

V. Hardware Decoder Architecture

A. Top-Level Architecture

The architecture for our proposed CW-PGDBF decoder is shown in Fig. 4, where $\mathbf{y}$ is the received noisy codeword and $\mathbf{v}^{(t)}$ is the estimated codeword at the current iteration t. The decoding process proceeds as follows. In the initialization stage, the signal *init* selects the noisy vector as the initial estimate $\mathbf{v}^{(0)}$. The CNUs then calculate the parity-checksum for each row's connected column through an XOR operation and send the results to the VNUs. In each VNU, the ECU$_{dv1}$ and ECU$_{dv2}$ modules calculate the sum of the parity-check results of the connected rows for the corresponding column, which is denoted as $\mathbf{S}^{(t)}$. After that is the column-weighting operation, where each $S_i^{(t)}$ is multiplied by its weighting factor and added to the term $v_i^{(t)} \oplus y_i$, as shown in Eq.(3). The maximum-finding operation is in parallel with the column-weighting operation. For each degree, the $S_i^{(t)}$ values are sent to the corresponding MF (maximum-finding) module to find the temporary maximum value, which is then sent to the MF_all module to find the maximum value $E_{max}^{(t)}$, as described in Eq. (6). The last step in each iteration is to compare each bit of $E_{v_i}^{(t)}$ with $E_{max}^{(t)}$'s. If $E_{v_i}^{(t)} = E_{max}^{(t)}$, then $v_i^{(t)}$ will be flipped with probability p_1 and if $E_{v_i}^{(t)} = E_{max}^{(t)} + 1$, then $v_i^{(t)}$ will be flipped with probability p_2. To generate random bits with a distribution of $\mathcal{B}(p_1)$ and $\mathcal{B}(p_2)$, we implement the random generator (RG) module using a cyclical shifting method during initialization. As p_2 is selected to be 1.00 for 4986.93xb.329 and PEGirReg504x1008, we can directly flip the corresponding bits instead of using an RG module, which reduces the hardware overhead. Finally, the updated $\mathbf{v}^{(t)}$ is sent to the next iteration until T_{max} iterations are spent or all of the checksums from CNUs are equal to zero.

The ECU (Energy-Value Calculation Unit) modules utilize the Leading Zero Counting Topology, described in [16], to add inputs and convert the result into a modified one-hot code. For instance, an ECU$_3$ has three inputs: s_0, s_1 and s_2. The resulting sum, S, is represented as S_3, S_2, S_1, where $S_j = 1$ if j is less than or equal to the sum of s_0, s_1 and s_2. To multiply S by a factor of n, we expand each bit of S by n times, resulting in $\{n\{S_3\}, n\{S_2\}, n\{S_1\}\}$. For addition operations, we need to calculate the energy value E by adding $v_i^{(t)} \oplus y_i$. If $v_i^{(t)} \oplus y_i$ is equal to 1, E is $\{S, 1\}$ and otherwise,E is $\{0, S\}$.

B. Synthesis Results

In this section, we present the synthesis results for the CW-PGDBF decoder, which has been implemented using register

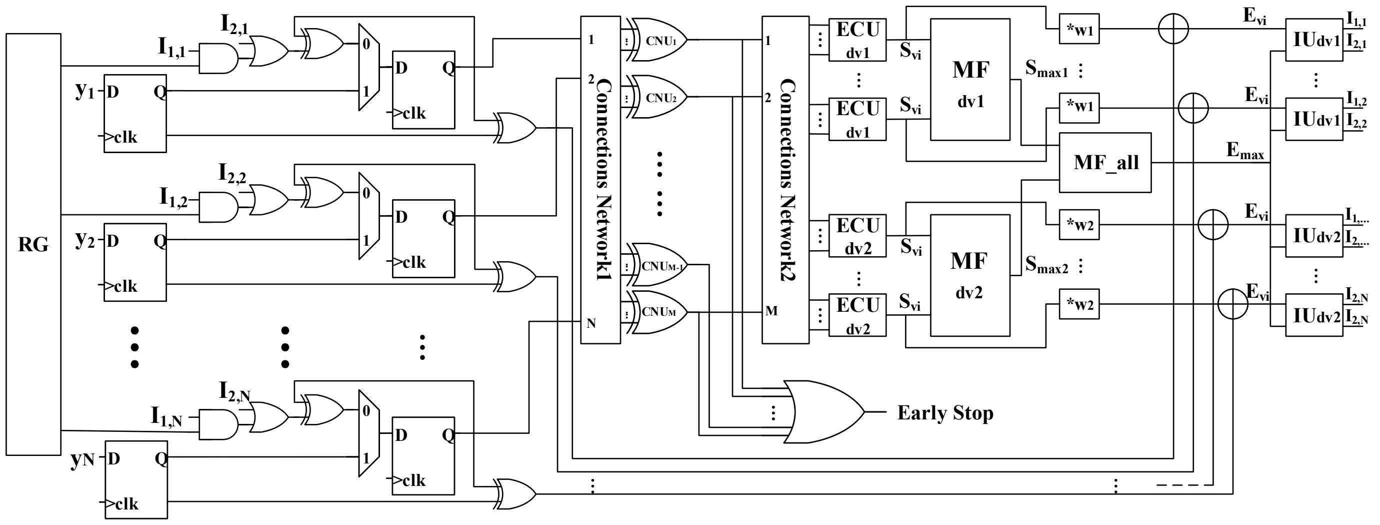

Fig. 4. The top-level architecture for CW-PGDBF decoder.

transfer level (RTL) design and synthesized under both 90nm and 28nm CMOS technologies with DC compiling tools. Since the algorithm is based on PGDBF, our analysis focuses on comparing the hardware resources and timing of CW-PGDBF with that of PGDBF. We evaluated the performance of the CW-PGDBF decoder on two benchmarks, 4986.93xb.329 and PEGirReg504x1008, and summarize the results in Table I.

TABLE I
THE SYNTHESIS RESULTS OF CW-PGDBF DECODERS

(a) 4986.93xb.329

	CW-PGDBF		PGDBF	
Technology	28nm	90nm	28nm	90nm
f_{max}(MHz)	1000	202	1000	233
Area(mm^2)	0.411	2.45	0.264	1.61
t_{ave}	12.6	12.6	9.1	9.1
θ(Gbps)	791.4	159.9	1096	255.3
AE(Gbps/mm^2)	1953	65.3	4151	158.6

(b) PEGirReg504x1008

	CW-PGDBF		PGDBF	
Technology	28nm	90nm	28nm	90nm
f_{max}(MHz)	1000	234	1000	229
Area(mm^2)	0.137	0.460	0.157	0.564
t_{ave}	3.2	3.2	19.1	19.1
θ(Gbps)	315	73.7	52.8	12.1
AE(Gbps/mm^2)	2299	160.2	336.1	21.4

In Table I, the parameter f_{max} represents the maximum frequency achievable in the decoding process. The average number of iterations when α is set to 0.005 is denoted by t_{ave}. The parameters θ and AE are respectively calculated as $\theta = \frac{f_{max}\times N}{t_{ave}}$ and $AE = \frac{f_{max}\times N}{t_{ave}\times \text{Area}}$.

As shown in Table I, the CW-PGDBF algorithm demonstrates an obvious performance improvement of 4986.93xb.329. It is worth noting that it also incurs a noticeable increase in hardware overhead. In contrast, such a phenomenon is not observed in the case of PEGirReg504x1008 using CW-PGDBF. This design incurs even less hardware overhead than PGDBF while achieving a substantially better throughput. The discrepancy in hardware overhead between CW-PGDBF and PGDBF can be attributed to the difference in the area of the IU and ECU modules in the respective decoders, which were synthesized using the 28nm process, as illustrated in Table II. The table shows the hardware resources consumed by one IU module and one ECU in each of the two decoders, as well as the number of such modules in each decoder.

TABLE II
THE HARDWARE AREA OF IUS AND ECUS IN CW-PGDBF AND PGDBF UNDER 28NM CMOS TECHNOLOGY

(a) 4986.93xb.329

	CW-PGDBF	Number	PGDBF	Number
IU_3	21.042	9141	3.402	9141
IU_9	29.61	831	10.08	831
ECU_3	3.402	9141	5.922	9141
ECU_9	23.94	831	26.082	831

(b) PEGirReg504x1008

	CW-PGDBF	Number	PGDBF	Number
IU	48.74	1008	17.05	1008
ECU_2	1.26	481	2.65	481
ECU_3	2.65	283	4.03	283
ECU_4	4.03	35	7.43	35
ECU_5	7.46	98	11.34	98
ECU_7	12.85	9	17.16	9
ECU_14	380.65	1	632.39	1
ECU_15	688.65	101	1157.70	101

On the one hand, Table II demonstrates that the IUs in CW-PGDBF require significantly more hardware resources than those in PGDBF for the same d_v. The IUs in PGDBF make comparisons between two one-hot inputs using OR gates on each bit, but CW-PGDBF needs two comparison operations per block, leading to a total of N more modules being needed and causing a substantial overhead. On the other hand, for the same d_v, ECUs in CW-PGDBF require fewer hardware resources than those in PGDBF. As explained earlier, the ECUs add up the inputs and convert the sum into the one-hot format, which leads to an exponential increase as the number of inputs grows. However, in CW-PGDBF, the input number of one ECU is one less than that in PGDBF, which results in a more significant reduction in hardware overhead compared with the increased hardware overhead of the IU module.

Table II clearly illustrates the impact of the IU and ECU modules on the hardware overhead of the CW-PGDBF and PGDBF decoders for different codewords. This can explain why the two codewords exhibit completely different results in terms of hardware overhead. For instance, for the codeword 4986.93xb.329, the negative impact of the IU module far outweighs the positive impact of the ECU module, resulting in a significantly higher overall hardware overhead for the CW-PGDBF decoder than the PGDBF decoder. Conversely, for the codeword PEGirReg504x1008, the increase in overhead caused by the IU module is smaller than the decrease in overhead caused by the ECU module, leading to a surprising reduction in overall hardware overhead for the CW-PGDBF decoder compared with the PGDBF decoder.

The above analysis indicates that compared with the PGDBF algorithm, the CW-PGDBF algorithm is more suitable for codewords with short code lengths and high column weights. This is due to the fact that under high column weights, the optimized overhead of the ECU modules may exceed the increased overhead of the IU modules that occurs when the code length is short. In such a scenario, the CW-PGDBF algorithm not only improves decoding performance but also reduces hardware complexity, making it a very promising and competitive algorithm.

VI. Conclusion

This work introduces a new decoder designed specially for irregular LDPC codes, in which a column-wise weighting factor is used for improved performance and a modified probabilistic flipping rule is applied for reduced complexity. Simulation results show that the proposed CW-PGDBF algorithm can achieve a far better performance compared with almost all existing algorithms for irregular LDPC codes. Besides, a hardware implementation is developed for the CW-PGDBF algorithm. The synthesis results demonstrate the effectiveness of the CW-PGDBF decoder, and future work may investigate improvements and optimizations for the CW-PGDBF algorithm.

Acknowledgment

This work was supported by the National Key R&D Program of China under Grant 2022YFB4400604. (Corresponding authors: Suwen Song; Zhongfeng Wang.)

References

[1] R. Gallager, "Low-density parity-check codes," *IRE Transactions on information theory*, vol. 8, no. 1, pp. 21–28, Jan. 1962.

[2] J. Chen and M. P. Fossorier, "Near optimum universal belief propagation based decoding of low-density parity check codes," *IEEE Transactions on communications*, vol. 50, no. 3, pp. 406–414, Mar. 2002.

[3] J. Chen, A. Dholakia, E. Eleftheriou, M. P. Fossorier, and X.-Y. Hu, "Reduced-complexity decoding of LDPC codes," *IEEE Transactions on Communications*, vol. 53, no. 8, pp. 1288–1299, Aug. 2005.

[4] V. V. Zyablov and M. S. Pinsker, "Estimation of the error-correction complexity for Gallager low-density codes," *Problemy Peredachi Informatsii*, vol. 11, no. 1, pp. 23–36, Jan. 1975.

[5] Y. Chen, H. Cui, J. Lin, and Z. Wang, "Fine-grained bit-flipping decoding for LDPC codes," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 5, pp. 896–900, May 2020.

[6] K. Deng, H. Cui, J. Lin, and Z. Wang, "Counter random gradient descent bit-flipping decoder for LDPC codes," in *2021 IEEE Computer Society Annual Symposium on VLSI.* IEEE, Jul. 2021, pp. 55–60.

[7] T. Wadayama, K. Nakamura, M. Yagita, Y. Funahashi, S. Usami, and I. Takumi, "Gradient descent bit flipping algorithms for decoding LDPC codes," *IEEE Transactions on Communications*, vol. 58, no. 6, pp. 1610–1614, Jun. 2010.

[8] O. A. Rasheed, P. Ivaniš, and B. Vasić, "Fault-tolerant probabilistic gradient-descent bit flipping decoder," *IEEE Communications Letters*, vol. 18, no. 9, pp. 1487–1490, Sep. 2014.

[9] K. Le, F. Ghaffari, L. Kessal, D. Declercq, E. Boutillon, C. Winstead, and B. Vasić, "A probabilistic parallel bit-flipping decoder for low-density parity-check codes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 1, pp. 403–416, Jan. 2019.

[10] H. Cui, J. Lin, and Z. Wang, "An improved gradient descent bit-flipping decoder for LDPC codes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 8, pp. 3188–3200, May 2019.

[11] H. Cui, J. Lin, and Z. Wang, "Information storage bit-flipping decoder for LDPC codes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 11, pp. 2464–2468, Nov. 2020.

[12] M. Luby, M. Amin Shokrolloahi, M. Mizenmacher, and D. Spielman, "Improved low-density parity-check codes using irregular graphs and belief propagation," in *Proceedings. 1998 IEEE International Symposium on Information Theory*, Aug. 1998.

[13] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, "Improved low-density parity-check codes using irregular graphs," *IEEE Transactions on information Theory*, vol. 47, no. 2, pp. 585–598, Feb. 2001.

[14] S. Brkic, P. Ivaniš, and B. Vasić, "Adaptive gradient descent bit-flipping diversity decoding," *IEEE Communications Letters*, vol. 26, no. 10, pp. 2257–2261, Oct. 2022.

[15] D. MacKay, "David mackay's Gallager code resources," 2008, http://www.inference.org.uk/mackay/CodesFiles.html.

[16] B. Yuce, H. F. Ugurdag, S. Gören, and G. Dündar, "Fast and efficient circuit topologies for finding the maximum of n k-bit numbers," *IEEE Transactions on Computers*, vol. 63, no. 8, pp. 1868–1881, Aug. 2014.