An Improved Gradient Descent Bit-Flipping Decoder for LDPC Codes

Hangxuan Cui, Jun Lin, *Senior Member, IEEE*, and Zhongfeng Wang, *Fellow, IEEE*

***Abstract*— Low-complexity and high-performance low-density parity-check (LDPC) decoders are highly demanded in various modern communication and storage systems. In this paper, a novel hard-decision decoding algorithm, called tabu-list random-penalty gradient descent bit-flipping (TRGDBF) algorithm, is proposed. Compared to the state-of-the-art hard-decision algorithms, the TRGDBF algorithm has much better error-correction performance due to several algorithmic improvements. First, a random-penalty term is introduced to the inversion function to help the decoder escape from trapping sets, which are the main causes of the error-floor phenomenon. Second, a tabu-list is employed to improve the decoding efficiency. Numerical results show that the TRGDBF algorithm can achieve up to two orders of magnitude better error-correction performance than the probabilistic gradient descent bit-flipping (PGDBF) algorithm and reduce the average iteration count by about 15%. In addition, a well-optimized hardware architecture is developed to implement the TRGDBF algorithm. Algorithmic transformation and architecture optimization are well explored to reduce the hardware complexity and latency. Synthesis results show that the TRGDBF decoder can work at a higher frequency and offer a larger throughput than the PGDBF decoder.**

***Index Terms*— Gradient descent bit-flipping, low-density parity-check codes, tabu-list, random perturbation, hard-decision decoding, high-performance decoder.**

I. Introduction

LOW-DENSITY Parity-Check (LDPC) codes [1] have received increasing attention in the past decades. Due to the capacity-approaching performance, LDPC codes have been adopted in 3GPP New Radio for 5G and many communication standards, such as IEEE802.16e, IEEE802.11ad, and IEEE802.11n/ac/ax. LDPC decoding algorithms usually follow an iterative manner. The decoding messages are iteratively exchanged between Variable Nodes (VNs) and Check Nodes (CNs). Soft-decision and hard-decision algorithms are two main classes of decoding algorithms for LDPC codes. When paired with soft-decision algorithms, such as the Belief Propagation (BP) algorithm [1], Min-Sum (MS) algorithm [2], and Offset Min-Sum (OMS) algorithm [3], LDPC codes have excellent performance at the cost of high computation complexity, which leads to a highly complex decoder. Instead of processing soft messages in soft-decision algorithms, only binary messages are passed and calculated for hard-decision algorithms, such as the Bit-Flipping (BF) algorithm [4] and Gallager-B algorithm [5]. Therefore, they have much less computation complexity but usually suffer from a poor decoding performance. These characteristics of hard-decision algorithms make them suitable for low-complexity hardware implementations but with limited error-correction performance.

Many research efforts have been made in improving the error-correction capability of hard-decision decoding algorithms and the BF algorithm has been investigated extensively. The decoding of the BF algorithm is based on the inversion function, which indicates the reliability of each bit. Parity check results and received messages from the noised channel are considered in this function. In each iteration, only the least reliable bit is flipped in the so-called single BF algorithm and all bits with reliability values falling below a specific threshold are flipped in the multi BF algorithm. Due to the very poor performance of the BF algorithm, many variants of it have been proposed in the past several years, such as the weighted BF (WBF) algorithm [6] and the improved BF (IBF) algorithm [7]. In the WBF algorithm, each parity check result is weighted by a factor that indicates the reliability of the corresponding CN, where the value of the factor is the smallest magnitude of the received messages sent to this CN. In the IBF algorithm, the bits with reliability values below a predetermined threshold are flipped with probability p rather than 1, where $0 < p < 1$, and the optimal p is decided empirically. These two algorithms have better decoding performance and less average iteration number when compared to the original BF algorithm. However, compared to soft-decision algorithms, these two algorithms still have much worse error-correction performance.

Recently, the Gradient Descent Bit-Flipping (GDBF) algorithm was proposed in [8], which demonstrates better error-correction performance than the algorithms mentioned above. For the GDBF algorithm, a non-linear objective function is defined and the decoding process is considered as the maximization process of this function. The inversion function is derived from a gradient descent formulation and the flipping rule is closely related to the steepest descent algorithm based on ℓ_1-norm [9]. To further improve the performance, the authors of [8] introduced a Hybrid GDBF (H-GDBF) algorithm to help the decoder escape from the undesirable

Manuscript received November 6, 2018; revised January 25, 2019 and March 10, 2019; accepted March 31, 2019. Date of publication April 17, 2019; date of current version July 3, 2019. This work was supported in part by the National Natural Science Foundation of China under Grants 61774082 and 61604068 and in part by the Fundamental Research Funds for the Central Universities under Grants 021014380065 and 021014380087. This paper was recommended by Associate Editor M. Martina. *(Corresponding authors: Jun Lin; Zhongfeng Wang.)*

The authors are with the School of Electronic Science and Engineering, Nanjing University, Nanjing 210008, China (e-mail: haxcui@163.com; jlin@nju.edu.cn; zfwang@nju.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSI.2019.2909653

1549-8328 © 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

OKTA-1029

local maximum. The flipping mode is switched from the single-bit mode to the multi-bit mode when a trapped search point is detected. This detection process requires evaluating the objective function across all bits in each iteration. The H-GDBF algorithm performs better than most known BF variants in term of error-correction performance. However, the global operation costs too much computation resource, especially for LDPC codes with large codeword length.

The Noisy GDBF (NGDBF) algorithm [10] offers similar error-correction performance as the H-GDBF algorithm with significantly reduced decoding complexity under the Additive White Gaussian Noise (AWGN) channel. More recently, the adjustment factor aided NGDBF (A-NGDBF) algorithm was proposed where an adjustment factor was introduced to the syndrome, leading to a slightly better error-correction performance. However, the soft information is required in the above algorithms which is hard to be obtained in some practical applications, such as NAND storage systems. For this reason, we restrict the discussion to the Binary Symmetric Channel (BSC) where only hard information can be used. In this case, the Probabilistic GDBF (PGDBF) [11] is one of the best hard-decision decoders in term of error-correction. The idea proposed in the IBF algorithm is adopted in the PGDBF algorithm to improve its decoding performance. Different from the GDBF algorithm, only a randomly selected subset of bits which have the minimum reliability value is flipped in the PGDBF algorithm. Many works have been done to implement the PGDBF algorithm [12]–[14]. As stated in [14], the complexity of the PGDBF decoder can be lower than that of the GDBF decoder. More recently, the Probabilistic Parallel Bit Flipping (PPBF) decoder [15] and the Flash Memory adapted PGDBF (FM-PGDBF) decoder [16] were proposed to reduce the latency of the PGDBF decoder. In the PPBF decoder, the flipping operation is performed in a probabilistic manner. In the FM-PGDBF decoder, a bit is flipped when the value of its inversion function is greater than or equal to a predefined threshold. Instead of the FM-PGDBF decoder with performance degeneration, the PPBF decoder has a similar error-correction performance as that of the PGDBF decoder [17]. However, there still exists an obviously performance gap between the PGDBF and soft-decision algorithms.

In this work, we propose an algorithm called Tabu-list Random-penalty GDBF (TRGDBF) algorithm for decoding LDPC codes using only hard information, aiming to narrow the performance gap between hard-decision and soft-decision decoding algorithms without sacrificing the superiority of the former with respect to hardware implementation. First, a random-penalty term is introduced to the inversion function to help the decoder escape from trapping sets (TSs). The value of the random-penalty term can only be 0 or 1, which avoids involving floating-point numbers and thus introducing less additional complexity. Second, in order to improve the decoding efficiency, a tabu-list is defined and employed in the flipping rule. Different from involving global operations in both the flipping-restriction mode proposed in [18] and the excluding-set proposed in [19], no detection operation is applied in the proposed tabu-list. Based on these modifications, the proposed TRGDBF algorithm reaches a comparable performance to that of the OMS algorithm. Compared to the PGDBF algorithm, the TRGDBF algorithm has a much larger coding gain while maintaining its low-complexity property.

A well-optimized hardware architecture for the TRGDBF algorithm is also developed in this paper. The random-penalty term is pre-generated to avoid using complex random number generators (RNGs). To mitigate the severe performance loss caused by the fixed penalty term, the estimated codeword is cyclically shifted by one position in each iteration. Simulation results show that only a negligible performance loss is observed with the aid of this solution. Moreover, the required hardware resource to find the global maximum value of the inversion function is reduced by nearly half. The proposed decoder architecture is coded with RTL and synthesized under the 90nm CMOS technology. The area of the proposed TRGDBF decoder is 0.299 mm^2, which is 93% of the PGDBF decoder [14] and 22% of the MS decoder [20] with the same codeword length. Moreover, the TRGDBF decoder can operate at a higher frequency (435MHz) which is 2.4 times and 1.2 times that of the PGDBF and MS decoders, respectively. When compared at the same channel condition, TRGDBF decoder has a much higher average throughput than that of the MS decoder. The average throughput of the TRGDBF decoder is 142Gbps, which is 1.43 times that of the PGDBF decoder.

The rest of this paper is organized as follows. Section II introduces the notations related to the LDPC codes and the GDBF decoding algorithm. In Section III, the proposed TRGDBF algorithm is described in detail. Section IV presents the simulation results including performance comparison and parameter selection. The hardware architecture is developed in Section V. In Section VI, we provide the synthesis results of the TRGDBF decoder. At last, the conclusion is drawn in Section VII.

II. Preliminaries

A. Notations

In this paper, we consider a linear LDPC code $\mathcal{C}$ over $GF(2)$ of length N, which is defined by an $M \times N$ parity check matrix $\boldsymbol{H}$, where $N > M$. The code rate $R = 1 - M/N$. M parity check equations correspond to M rows of the parity check matrix, indexed by $\mathcal{J} = \{0, 1, \cdots, M-1\}$. Each of the N codeword symbols, indexed by $\mathcal{I} = \{0, 1, \cdots, N-1\}$, corresponds to a column in $\boldsymbol{H}$. An LDPC code can also be represented by a Tanner graph which has N VNs and M CNs. $\mathcal{N}_c(j)$ is the set of indices of VNs that connected to the j-th CN, *i.e.*, $\mathcal{N}_c(j) = \{\, i \mid h_{j,i} = 1\}$. Similarly, $\mathcal{N}_v(i)$ denotes the neighborhoods of the i-th VN, *i.e.*, $\mathcal{N}_v(i) = \{\, j \mid h_{j,i} = 1\}$. The degree of the i-th VN $d_v^i = |\mathcal{N}_v(i)|$ and the degree of the j-th CN $d_c^j = |\mathcal{N}_c(j)|$. A length-$N$ binary vector $\boldsymbol{u}$ is a codeword if all parity checks are satisfied, *i.e.*, $\boldsymbol{u}\boldsymbol{H}^T = \boldsymbol{0}$. For a regular (J, K) QC-LDPC code, $\boldsymbol{H}$ consists of $J \times K$ submatrices. $d_c^j = d_c$ and $d_v^i = d_v$ for arbitrary $i \in \mathcal{I}$, $j \in \mathcal{J}$. The submatrix $\boldsymbol{H}_{m,n}$ is a $z \times z$ zero square matrix if the shifting factor $c_{m,n} = -1$. Otherwise, $\boldsymbol{H}_{m,n}$ is obtained by circularly shifting a $z \times z$ identity matrix $c_{m,n}$ times.

B. The GDBF Algorithm

Assume $\boldsymbol{y}$ is the received log-likelihood ratio vector from AWGN channel and the Binary Phase Shift Keying (BPSK) modulation is applied. The maximum likelihood (ML) decoding [21] is equivalent to the problem of finding an estimated codeword $\boldsymbol{x}_{ML}$, which satisfies the equation

$$\boldsymbol{x}_{ML} = \arg\max_{\boldsymbol{x}\in\mathcal{C}} \sum_{i=1}^{N} (1-2x_i)y_i. \tag{1}$$

In the GDBF algorithm, the maximum likelihood problem is transformed to the optimization process of an objective function. The syndromes are included in this function as a penalty term, as shown in Eq. (2).

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} (1-2x_i)y_i + \sum_{j=1}^{M} \prod_{i\in\mathcal{N}_c(j)} (1-2x_i). \tag{2}$$

The gradient descent method is used to solve the optimization problem. The flipping rule consists of selecting the position at which the absolute value of the partial derivative of the objective function is largest (steepest descent direction). Eq. (3) is the inversion function (energy function) of the GDBF algorithm and the value of this function is called energy value.

$$\triangle_k(\boldsymbol{x}) = (2x_k - 1)y_k + \sum_{j\in\mathcal{N}_v(k)} (2s_j - 1). \tag{3}$$

$\boldsymbol{s}$ is the syndrome vector and $s_j = \oplus_{i\in\mathcal{N}_c(j)} x_i$. $\oplus$ represents the bit-wise Exclusive-OR (XOR) operation. Two versions of the GDBF algorithm were presented in [8], named Single GDBF algorithm (S-GDBF) and Multi GDBF (M-GDBF) algorithm, respectively. In the S-GDBF algorithm, only the bit with the maximum energy value is flipped. In the M-GDBF algorithm, all the bits with energy values greater than a predetermined threshold are flipped. The M-GDBF algorithm converges faster and S-GDBF algorithm has more accurate convergence behavior.

In [11], the objective and inversion functions are modified to adapt to BSC. Assume an LDPC codeword $\boldsymbol{u}$ is transmitted over a BSC with crossover probability α and $\boldsymbol{r}$ is the received vector, *i.e.*, $\Pr(r_i = u_i) = 1-\alpha$ and $\Pr(r_i = 1-u_i) = \alpha$. $\boldsymbol{c}$ is the estimated codeword. The modified objective and inversion functions are expressed as Eq. (4) and Eq. (5), respectively.

$$O(\boldsymbol{x}) = \sum_{i=1}^{N} (1 - c_i \oplus r_i) + \sum_{j=1}^{M} (1 - s_j). \tag{4}$$

$$\Lambda_k(\boldsymbol{c}) = c_k \oplus r_k + \sum_{j\in\mathcal{N}_v(k)} s_j. \tag{5}$$

In each iteration, the bits with maximum energy value are flipped. Since the range of values of $\Lambda_k(\boldsymbol{c})$ is 0 to $d_v + 1$, there are only $d_v + 2$ available numbers for the energy values of all bits. In most cases, the length of an LDPC codeword is much larger than $d_v + 2$. Therefore, there are usually many bits sharing the same energy value and the number of bits with the maximum value is very large.

The GDBF algorithm for BSC is described in Algorithm 1, where $\boldsymbol{c}^{(t)}$ denotes the estimated vector in the t-th iteration and $t \in [0, T_{max}[$. T_{max} is the maximum number of iterations. Specifically, $\boldsymbol{c}^{(0)} = \boldsymbol{r}$. First, each CN updates its syndrome value through the bit-wise XOR operation. Second, the energy value of each bit is calculated using Eq. (5) and the maximum energy value is obtained. Finally, the bits with the maximum energy value are flipped. In each iteration, a comparison operation is executed to determine whether the current estimated codeword $\boldsymbol{c}^{(t)}$ is valid or not. If $\boldsymbol{c}^{(t)}$ satisfies the parity checks, a decoding success will be claimed. Otherwise the decoding will go to the next iteration. The decoding will be terminated when T_{max} is reached and declare a decoding failure if the correct codeword is still not obtained.

Algorithm 1 The GDBF Algorithm for BSC

input : $\boldsymbol{r} = (r_0, r_1, \cdots, r_{N-1})$
initialize: $\boldsymbol{c}^{(0)} = \boldsymbol{r}$, $t = 0$
for $t = 0$ **to** $T_{max} - 1$ **do**
  for $j = 0$ **to** $M - 1$ **do**
    $s_j = \oplus_{i\in\mathcal{N}_c(j)} c_i^{(t)}$
  if $\boldsymbol{s} = \boldsymbol{0}$ **then**
    break
  $\Lambda_{max} = 0$
  for $i = 0$ **to** $N - 1$ **do**
    $\Lambda_i = c_i^{(t)} \oplus r_i + \sum_{j\in\mathcal{N}_v(i)} s_j$
    if $\Lambda_{max} < \Lambda_i$ **then**
      $\Lambda_{max} = \Lambda_i$
  for $i = 0$ **to** $N - 1$ **do**
    if $\Lambda_i = \Lambda_{max}$ **then**
      $c_i^{(t+1)} = 1 - c_i^{(t)}$
output : $\boldsymbol{c}^{(t)}$

III. Proposed Tabu-List Random-Penalty GDBF Algorithm

A. Random-Penalty Term

It has been proved that small TSs are the main cause of the error-floor phenomenon of LDPC codes [22] and many methods have been proposed to solve this problem [23], [24]. To help the decoder escape from TSs, a modified version of the GDBF algorithm, named RGDBF algorithm, is proposed. The proposed RGDBF algorithm is mainly derived through simulation, based on observing the structure of TS and the decoding process of the GDBF algorithm.

Fig. 1 shows two states of a TS which has five VNs and nine CNs, where Fig. 1(a) is the original state and Fig. 1(b) is the intermediate state. Each VN is marked with a circle and each CN is marked with a square. Assume an all-zero codeword is transmitted over BSC channel. The black ones denote erroneous bits and unsatisfied checks. The received values of the five VNs are 1, 1, 1, 1, and 0, respectively. In Fig. 1(a), since there is an unsatisfied check node (c_3) connected to v_1 and $v_1 \oplus r_1 = 0$, according to Eq. (5), the energy value of v_1 is 1. Similarly, the energy values of the latter four VNs are 1, 1, 1, and 2, respectively. Hence, v_5

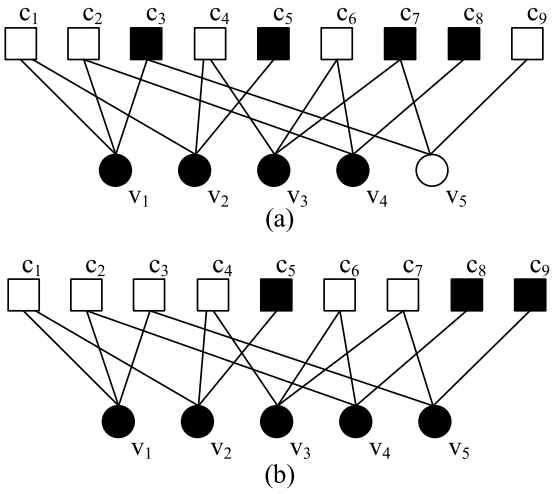


Fig. 1. Two states of a trapping set.

should be flipped. After a flip, the energy values of the five VNs are transformed to 0, 1, 0, 1, and 2, respectively. v_5 should be flipped again. It can be seen that the TS is recovered to the original state after being flipped twice. Therefore, the decoding will be locked in this TS so that this error pattern cannot be corrected by the GDBF decoder. This kind of TSs cannot be corrected by the PGDBF decoder either. When decoded by the PGDBF decoder, only v_5 may be flipped in each iteration because only the bits with the maximum energy value can be flipped with probability p. Hence, the PGDBF decoder cannot solve this error pattern.

The RGDBF algorithm introduces a random-penalty term to the inversion function, as shown in Eq. (6).

$$\Lambda_k^R(\boldsymbol{c}) = c_k \oplus r_k + \sum_{j \in \mathcal{N}_v(k)} s_j + \lambda, \tag{6}$$

where λ equals 0 or 1. $\Pr(\lambda = 1) = \sigma$ and $\Pr(\lambda = 0) = 1 - \sigma$. σ is a predetermined probability value and $0 < \sigma < 1$. Through this operation, the regularity of the decoding is broken so that the probability of the decoder captured by a TS is reduced. Moreover, since the gradient steepest descent direction is not always the optimal direction to obtain a valid codeword, only flipping the bits in this direction limits the decoding performance. The random-penalty term gives chances for the bits which are not in the gradient steepest descent direction to be flipped. As a result, the decoding performance is improved. For the TS shown in Fig. 1, the RGDBF algorithm can help the decoder escape from it. When $\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = 1$ and $\lambda_5 = 0$, v_1 to v_5 will be flipped at the same time. As a result, only v_5 is incorrect in the next iteration, which is very easy to be corrected. Simulation results demonstrate that the RGDBF has better error-correction performance than the GDBF and PGDBF algorithms for selected codes, which will be introduced in Section IV-B.

B. Tabu-List

To further improve the decoding performance, a tabu-list is designed to overcome the repeated flip issue, *i.e.*, some bits

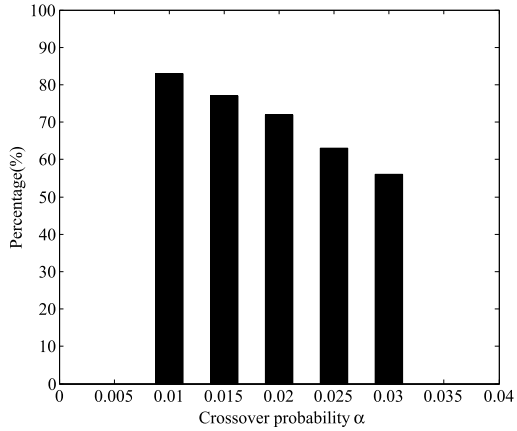


Fig. 2. The percentage of the decoding failures caused by the repeated flip issue in all decoding failures for the GDBF decoder.

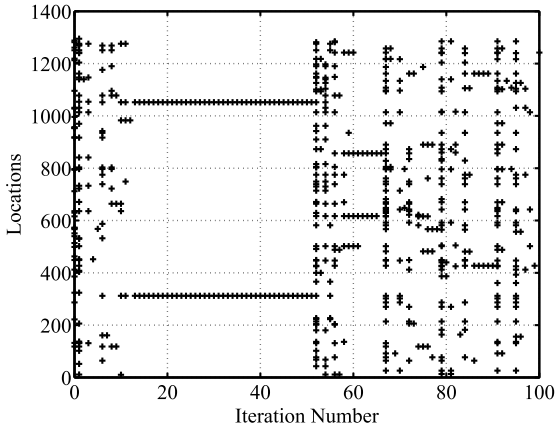


Fig. 3. Locations of the flipped bits in each iteration.

are flipped repeatedly in several consecutive iterations. The percentage of the decoding failures caused by this problem in all decoding failures for the GDBF decoder is shown in Fig. 2, where the Tanner code is used. It can be concluded that the repeated flip issue is the main cause of the decoding failure, especially in the low-error region. When $\alpha = 0.01$, the percentage equals 83%. Hence, designing a specific solution for this problem is very meaningful.

As an example, we monitor the decoding process of the RGDBF algorithm in a failed decoding and the locations of flipped bits in each iteration are recorded and presented in Fig. 3. A $d_v = 4$, $d_c = 8$, $R = 0.5$, $N = 1296$ (dv4R050N1296) QC-LDPC code is used and T_{max} is set to 300. Only the results in the first 100 iterations are shown for clarity. As can be seen, there are some bits being flipped repeatedly in several iterations, which means that the decoding efficiency of the RGDBF algorithm is damaged by the repeated flip issue. Hence, the RGDBF algorithm is still not effective enough for this problem. To solve this problem, it is crucial to avoid sticking to flip certain bits. Therefore, we put the bits which are flipped in the current iteration to the tabu-list and free them in the next iteration. In each iteration, only the bits which are not in the tabu-list and have the maximum energy value are flipped. Moreover, the bits in the tabu-list will not participate in the process of finding the maximum energy value. This algorithm is called the TRGDBF algorithm.

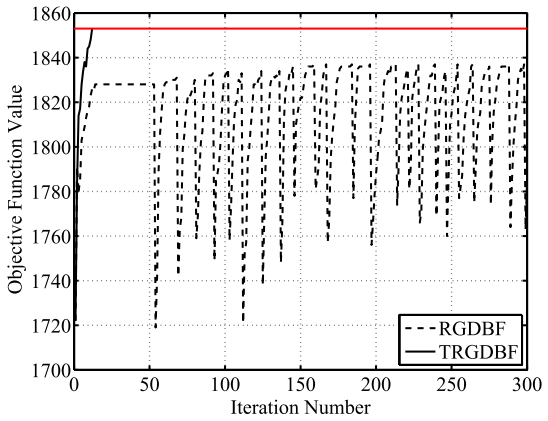


Fig. 4. Values of the objective function in each iteration.

Since the decoding process of GDBF variants can be regarded as the maximization process of the objective function, the value of this function can be used to monitor the decoding behavior. Fig. 4 shows the values of the objective function under the RGDBF and TRGDBF algorithms, where the codeword which causes the decoding failure appearing in Fig. 3 is applied.

Algorithm 2 The Proposed TRGDBF Algorithm

input : $\boldsymbol{r} = (r_0, r_1, \cdots, r_{N-1})$
initialize: $\boldsymbol{c}^{(0)} = \boldsymbol{r}$, $t = 0$, $(l_0, l_1, \cdots, l_{N-1}) = \mathbf{0}$
for $t = 0$ **to** $T_{max} - 1$ **do**
  for $j = 0$ **to** $M - 1$ **do**
    $s_j = \oplus_{i \in \mathcal{N}_c(j)} c_i^{(t)}$
  if $\boldsymbol{s} = \mathbf{0}$ **then**
    break
  $\Lambda_{max}^R = 0$
  for $i = 0$ **to** $N - 1$ **do**
    Generate the penalty term λ_i
    $\triangle_i^R(\boldsymbol{c}) = c_i \oplus r_i + \sum_{j \in \mathcal{N}_v(i)} s_j + \lambda_i$
    if $\triangle_{max}^R < \triangle_i^R$ *and* $l_i = 0$ **then**
      $\triangle_{max}^R = \triangle_i^R$
  for $i = 0$ **to** $N - 1$ **do**
    if $\triangle_i^R = \triangle_{max}^R$ *and* $l_i = 0$ **then**
      $c_i^{(t+1)} = 1 - c_i^{(t)}$
      $l_i = 1$
    else $l_i = 0$;
output : $\boldsymbol{c}^{(t)}$

As shown in Fig. 4, when paired with the GDBF algorithm, the global maximum value of the objective function, marked as the red line, cannot be reached within 300 iterations so a decoding failure is observed. However, the TRGDBF algorithm only needs 12 iterations to reach the global maximum value and declares a decoding success. Hence, the effectiveness of the tabu-list is illustrated in the example of Fig. 4. Algorithm 2 describes the proposed TRGDBF algorithm, where l_i is used to indicate whether the i-th bit is in the tabu-list or not.

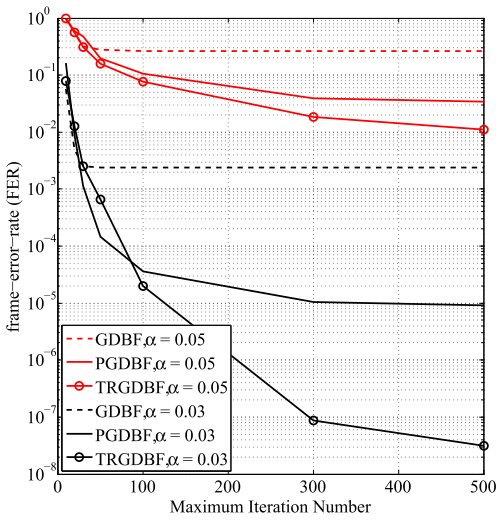


Fig. 5. The FER performances of several algorithms under different maximum iteration numbers.

IV. Simulation Results

The decoding performance of the TRGDBF algorithm is illustrated in this section and compared with the GDBF, OMS, and PGDBF algorithms. Moreover, the performance of the OMS algorithm with a layered schedule is also included for reference, which is marked as OMS-layered. All algorithms are simulated on BSC and the numerical results are obtained based on Monte Carlo methods. We consider four regular LDPC codes with different lengths and rates. The dv4R050N1296 QC-LDPC code which is used in [14] for the PGDBF algorithm is employed to evaluate the performance of the TRGDBF algorithm. A $d_v = 3$, $d_c = 6$, $R = 0.5$, $N = 1296$ (dv3R050N1296) QC-LDPC code [14], a rate-1/2 regular (4,8) LDPC code of length 4000 (dv4R050N4000), and a rate-0.87 regular (4,31) LDPC code of length 16383 (dv4R087N16383) are also used to verify the performance. The latter two LDPC codes are selected from MacKay's online encyclopedia [25]. For the OMS based algorithms, the maximum number of iterations is set to 20. In all simulations, the decoding will be terminated when 300 failed codewords are collected.

A. Performance Analysis

In order to better understand the decoding behaviors of the GDBF, PGDBF, and TRGDBF algorithms and find a proper maximum number of iterations for each algorithm, the FER performances of them under different T_{max} are presented in Fig. 5, where the dv4R050N1296 code is applied. Two channel conditions are considered to obtain a complete analysis result, which are $\alpha = 0.05$ (waterfall region) and $\alpha = 0.03$ (error-floor region), respectively. In both situations, the GDBF algorithm only needs about 50 iterations to reach its performance limit. Also, it has the worst performance limit. In the waterfall region, the TRGDBF performs better than the PGDBF under all values of T_{max} for the selected code. However, in the error-floor region, the PGDBF algorithm is superior to the TRGDBF algorithm when T_{max} is less than 100. When T_{max} is set to a larger value, the performance

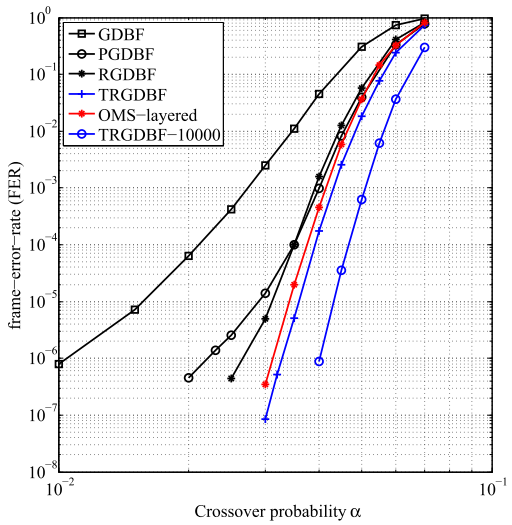


Fig. 6. The simulation results for the dv4R050N1296 LDPC code.

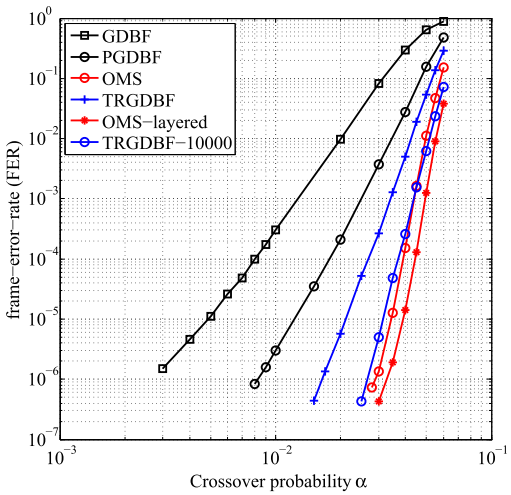


Fig. 7. The simulation results for the dv3R050N1296 LDPC code.

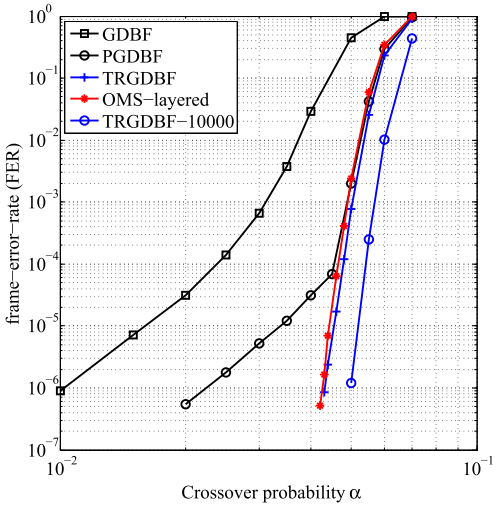


Fig. 8. The simulation results for the dv4R050N4000 LDPC code.

curve of the PGDBF begins to level off while that of the TRGDBF continues to be better. Based on these results, the following conclusions can be drawn. When compared to the current GDBF variants, the TRGDBF converges slower but has a much better performance limit. The performance gain continues to increase along with the maximum iteration number. When the number of iterations is sufficient large, e.g., $T_{max} = 500$, the TRGDBF can perform an impressive decoding performance. Hence, the TRGDBF is very suitable in applications requiring very low error-rate and where the decoding time is not the major limitation.

Based on the above analyses, the maximum numbers of iterations are set to 50 for GDBF and 300 for PGDBF and TRGDBF to make a fair comparison. Moreover, the simulation results of the TRGDBF algorithm when $T_{max} = 10000$ are also presented to illustrate its performance limit, which are marked as TRGDBF-10000.

B. Decoding Performance of the TRGDBF Algorithm

Fig. 6 shows the simulation results for the dv4R050N1296 code, where the OMS decoder takes layered schedule to obtain a better performance. Parameter σ is set to 0.9. The selection of this parameter will be introduced in Section IV-C. As can be seen, the TRGDBF algorithm offers the best performance among all algorithms. The performance of it is more than four orders of magnitude better than that of the GDBF algorithm when the crossover probability $\alpha = 0.03$. The TRGDBF algorithm provides a much larger performance gain than the PGDBF algorithm, which is one of the algorithms that have the best decoding performance among the current GDBF variants. Moreover, since the TRGDBF algorithm is designed by targeting the TSs, the error-floor phenomenon associated with the PGDBF algorithm has not been observed for the TRGDBF algorithm within the simulated frame error rate range. Hence, the modifications proposed in this paper can also be used to improve the error-floor problem. In addition, the performance of the TRGDBF algorithm is slightly better than that of the OMS-layered algorithm, which is a commonly used soft-decision algorithm. To the best of our knowledge, no existing hard-decision algorithm can achieve this standard while keeping the low-complexity property. When T_{max} is set to 10000, the performance of the TRGDBF algorithm can be further improved to a much higher level, which demonstrates it still has a great potential in error-correction when more iterations are used.

In order to verify the effectiveness of the random-penalty term, the performance curve of the RGDBF algorithm is also included in Fig. 6. It can be seen that the performance of the RGDBF is superior to that of the PGDBF algorithm, especially in the low frame-error-rate (FER) region. In the PGDBF algorithm, a random perturbation is introduced to the decoding to select a random subset of bits with the maximum energy value [11]. Due to the better performance, our random-penalty term is more effective than the random perturbation introduced in [11]. Moreover, the effectiveness of the tabu-list is further verified since the TRGDBF algorithm offers a much better performance than the RGDBF algorithm.

Figs. 7, 8, and 9 show the simulation results for the dv3R050N1296 code, dv4R050N4000 code, and the

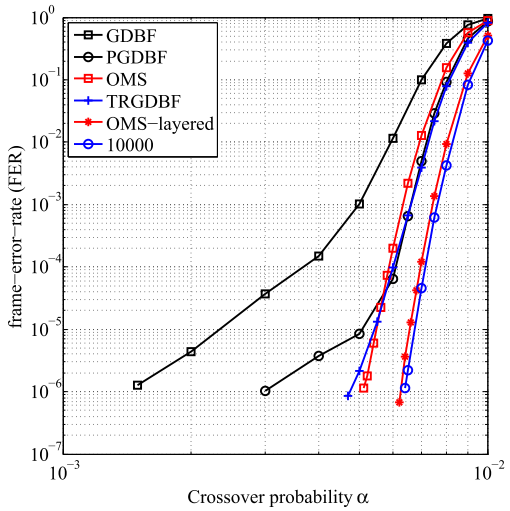


Fig. 9. The simulation results for the dv4R087N16383 LDPC code.

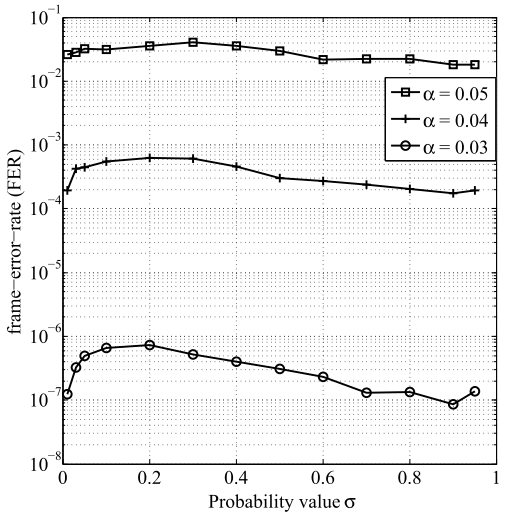


Fig. 10. The FER performances versus the probability value σ of the TRGDBF algorithm.

dv4R087N16383 code, respectively. Parameters σ for these codes are set to 0.9 uniformly. For all selected codes, the TRGDBF algorithm has a much better FER performance than that of the PGDBF algorithm. For the dv3R050N1296 and dv4R087N16383 codes, the performance of the TRGDBF algorithm is worse than that of the OMS algorithm. However, when T_{max} is increased to 10000, the TRGDBF algorithm can achieve a performance which is close to that of the OMS algorithm with layered schedule. Moreover, the error-floor phenomenon is still not observed yet for the TRGDBF algorithm. As a result, we can say that the TRGDBF algorithm significantly narrows the performance gap between hard-decision and soft-decision decoding algorithms. When the maximum number of iterations is not limited, it can achieve a surprising performance which is comparable to that of the OMS algorithm with layered schedule.

C. Parameter Selection

In this section, we will study the choice of parameter σ. The optimal value of the probability value σ is found through

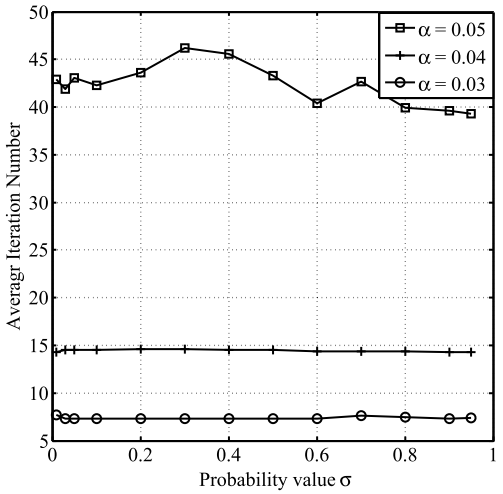


Fig. 11. The average iteration numbers versus the probability value σ of the TRGDBF algorithm.

a numerical search method. Fig. 10 shows FER versus the probability value σ of the TRGDBF algorithm under three different channel conditions, where the dv4R050N1296 code is used. It can be seen that the decoding performance is not very sensitive to parameter σ, which is suitable for hardware implementations and practical applications. The best performance can be obtained when $\sigma = 0.9$. On the premise of guaranteeing the decoding performance, the optimal probability value should also be chosen to minimize the average iteration number per frame. Fig. 11 shows the average iteration numbers as a function of σ. As can be seen, average iteration numbers are stable in all points, especially when $\alpha = 0.03$ and $\alpha = 0.04$. This result shows that the choice of σ has little impact on the average iteration number in medium to low FER regions. The minimum average iteration number also appears when $\sigma = 0.9$. Hence, the parameter σ is set to 0.9 in all simulations of the dv4R050N1296 code. The optimal values for other codes can also be obtained in this way. For all selected codes, the average number of iterations is insensitive to σ and 0.9 is a generally good choice. However, for the dv4R087N16383 codes, the FER performance is sensitive to σ. This is because the performance of the dv4R050N4000 code is very sensitive to the simulation settings, such as channel condition and parameter selection. For example, the FER of the dv4R050N4000 code decreases four orders of magnitude when α decreases from 0.05 to 0.04. Hence, a small deviation from the optimal value of σ will lead to a big performance loss, especially in the waterfall region.

D. Average Iteration Number

Fig. 12 shows the average iteration numbers of three algorithms, where the dv4R050N1296 code is adopted. When compared at the same crossover probability point, the average iteration number of the PGDBF algorithm is the largest, while the TRGDBF algorithm has the intermediate number and the GDBF algorithm has the smallest number. This is because the GDBF converges faster and the maximum iteration number of it (50) is far less than that of other two algorithms (300).

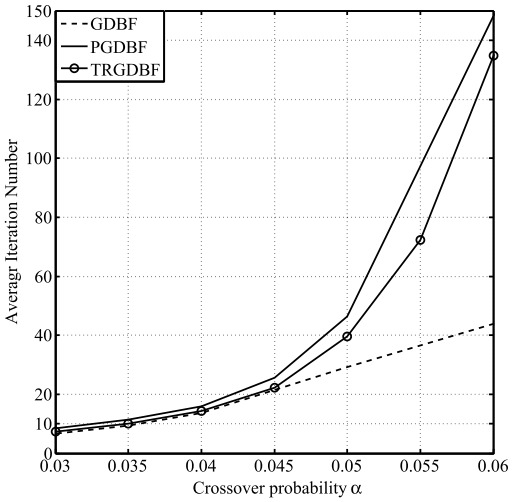


Fig. 12. The simulation results of the average iteration number at a given crossover probability.

Moreover, compared to the PGDBF algorithm, the TRGDBF algorithm needs smaller average number of iterations to correct the same amount of erroneous bits. It reduces the average iteration number by about 15% for the dv4R050N1296 code. When $\alpha = 0.03$, the average iteration number is 8.5 for PGDBF and 7.3 for TRGDBF. Thus, it can be concluded that the TRGDBF algorithm not only has a better error-correction ability, but also provides a faster convergence speed when compared to the PGDBF algorithm.

V. Efficient Hardware Decoder Architecture

Compared to the GDBF algorithm, the random-penalty term and tabu-list are added in the TRGDBF algorithm. Therefore, a well-optimized hardware architecture is developed in this section to implement the TRGDBF algorithm. First, since RNGs cost too much hardware resource, the random-penalty term is predetermined to avoid using RNGs. By referring to the idea proposed in [14], the regularity of QC-LDPC codes is employed to reduce the performance loss caused by the fixed penalty term. Second, the hardware resource required to determine the bits which should be flipped is reduced by nearly half, especially for codes with high column weight.

A. Top-Level Architecture

The top-level architecture of the TRGDBF decoder for QC-LDPC codes is shown in Fig. 13, where $\boldsymbol{r}$ is the received vector and $\boldsymbol{v}$ is the estimated codeword. The decoding process is described as follows. At the initialization of the decoding, $\boldsymbol{v} = \boldsymbol{r}$. Then, M CN units (CNUs) are used to calculate the syndrome vector $\boldsymbol{s}$, which will be sent to the inversion indicator (InvIn). The energy value of each bit is calculated in the inversion indicator and it also determines whether the bit should be flipped or not. The bits, which are not in the tabu-list and have the maximum energy value, correspond to $I_i = 1$. Otherwise $I_i = 0$. All bits with $I_i = 1$ are flipped in VN units (VNUs). There are K VNUs in the proposed architecture. Each VNU corresponds to z rows (a base row)

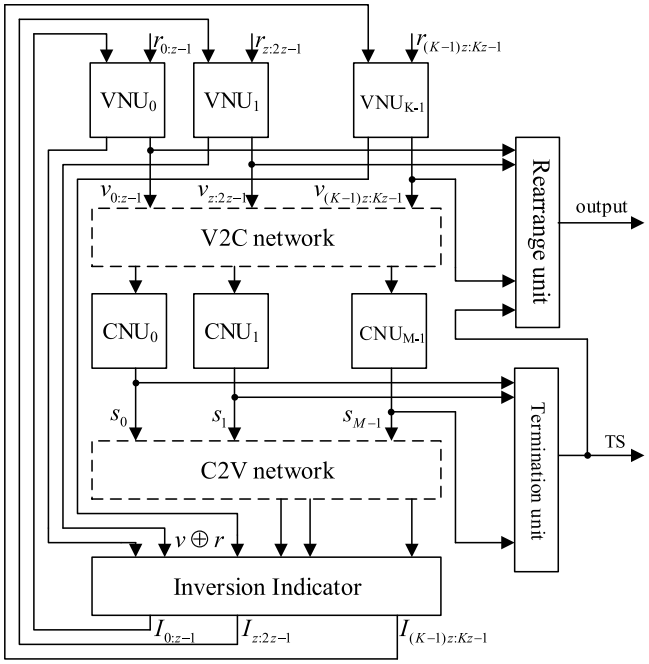


Fig. 13. The top-level architecture of the TRGDBF decoder.

of the parity check matrix. In each iteration, the termination unit will verify whether the currently estimated codeword is valid or not. By employing the tree structure, the latency of this unit is reduced from $M - 1$ to $\lceil \log_2 M \rceil$. When the valid codeword is obtained or the maximum number of iterations is reached, the output of this unit TS will be transformed to 1. Finally, vector $\boldsymbol{v}$ will be re-sorted to the correct order in the rearrange unit (ReU) and the decoder will be stopped.

B. Shift Structure

As mentioned earlier, since RNGs cost too much hardware resource, it is very necessary to avoid using it. Hence, the random-penalty term is pre-generated and fixed in this work. However, after being fixed, the randomness of the penalty term will not exist so it will cause great performance loss. In order to preserve the performance gain of the TRGDBF decoder, the penalty term sequence is cyclically shifted by one position in each iteration. Simulation results show that this modification only causes a negligible performance loss, which will be presented in Section VI. Since the penalty term λ_i may be 0 or 1, N additional registers are required to store them. Because N is a large number in most cases, it is obviously not the optimal solution.

Cyclic shift structure is commonly used in IC design for many different applications, especially for the encoder and decoder design of quasi-cyclic LDPC code. The following VNU architecture is employed in this work. Through shifting the received and estimated codeword vectors instead, we can achieve the same function as shifting the random bits and meanwhile the N registers used to store the random bits can be eliminated. Compared to the VNU proposed in [14], the VNU in our paper undertakes fewer tasks and is only used to rotate the received codeword $\boldsymbol{r}$ and estimated codeword $\boldsymbol{v}$ without involving other calculations.

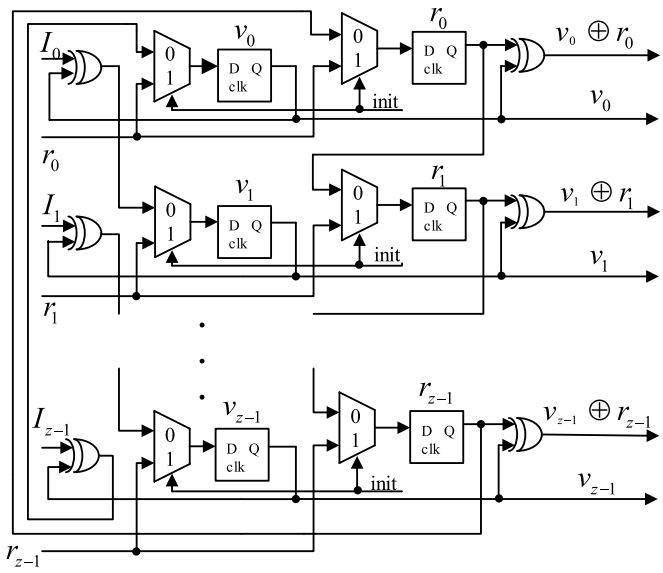


Fig. 14. The architecture of VNU.

	λ_0	λ_1	...	λ_{z-1}	λ_z	λ_{z+1}	...	λ_{2z-1}	... λ_{Kz-1}
iteration 1:	v_0	v_1	...	v_{z-1}	v_z	v_{z+1}	...	$v_{2z\text{-}1}$	... $v_{Kz\text{-}1}$
iteration 2:	v_{z-1}	v_0	...	v_{z-2}	$v_{2z\text{-}1}$	v_z	...	v_{2z-2}	... v_{Kz-2}

Fig. 15. The shift operation of QC-LDPC codes.

Fig. 14 shows the proposed VNU architecture, where the $init$ signal is valid only at the beginning of decoding to initialize the registers. The random sequence is fixed and the shift operation is realized by changing the message storage locations of vector $\boldsymbol{r}$ and vector $\boldsymbol{v}$. In each iteration, the elements in these two vectors are cyclically shifted one position to the right. Through this operation, the value of the random penalty term in each location is fixed so there is no need to use registers to store the random bits. Since the V2C network is fixed in the hardware implementation, changing the message storage locations may cause the mismatch problem. To handle this problem, the regularity of QC-LDPC codes is employed.

For QC-LDPC codes, the submatrix is a $z \times z$ zero square matrix or a matrix which is obtained by circularly shifting a $z \times z$ identity matrix. Therefore, the codeword vector is divided into K sections. Each section is a length-z sub-vector and processed by a VNU. As shown in Fig. 15, λ_0 is combined with v_0 in the first iteration. Since we take a length-z sub-vector as a unit, λ_0 is combined with v_{z-1} rather than v_{Kz-1} in the next iteration. As a result, the arrangement of the items in syndrome vector $\boldsymbol{s}$ is also changed as it does in vector $\boldsymbol{v}$. Therefore, we do not have to change the V2C network after each shift operation, which makes this modification feasible.

C. Inversion Indicator

The inversion indicator occupies a large part of the area of the decoder and has large critical path delay. Therefore, it is necessary to optimize this unit. The proposed inversion indicator is shown in Fig. 16, which is designed based on the Leading-zero Counting Topology (LCT) presented in [26]. There are N Energy-value Calculation Units (ECUs) and each ECU is used to calculate the energy value of a bit. The bits which have the maximum energy value and are not in the tabu-list will be identified by the Maximum-value Selection Unit (MSU). As shown in Fig. 16, this unit contains two parts. The maximum value finder is used to calculate the maximum energy value and Indicator Units (IUs) produce the indicator value I_i of each bit.

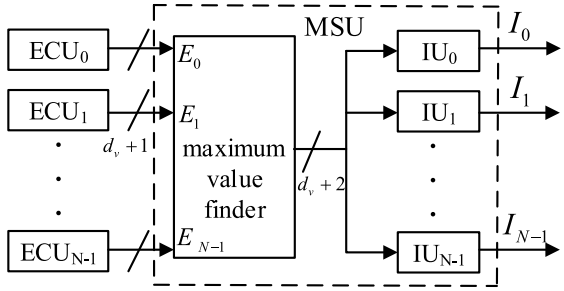


Fig. 16. The architecture of the inversion indicator.

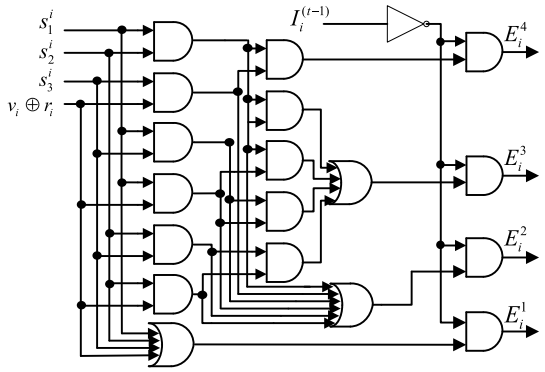


Fig. 17. The architecture of ECU.

In this paper, we take the code with $d_v = 3$ as an example to introduce the detailed design for clarity. The architecture for other codes can be easily concluded. Fig. 17 shows the ECU used for the i-th bit, where s_1^i, s_2^i, and s_3^i are the syndromes connected to the i-th VN. The energy value is expressed in a modified one-hot format, *i.e.*, E_i will be represented by four bits $E_i^4 ... E_i^2 E_i^1$ and $E_i^k = 1$ if $E_i \geq k$. Note that the bits which are in the tabu-list ($I_{i-1} = 1$) should not participate in the process of finding the maximum energy value. Therefore, the energy values of these bits are set to zero through AND operations.

From Eq. (6) we know that for a code with $d_v = 3$, the inputs of ECU should include five items, *i.e.*, $v_i \oplus r_i$, three syndromes, and the penalty term λ_i. However, the random penalty term is not included in the inputs of the proposed ECU. As stated in Section V-B, by applying the shift structure, the value of λ_i in each location is fixed. Therefore, the random penalty term can be introduced by specially treating the energy values in positions with $\lambda_i = 1$ without entering the penalty item into the ECU, which will be introduced later. In this work, we take the delay and area of an OR gate as a time-unit and an area-unit, respectively. We can calculate that for

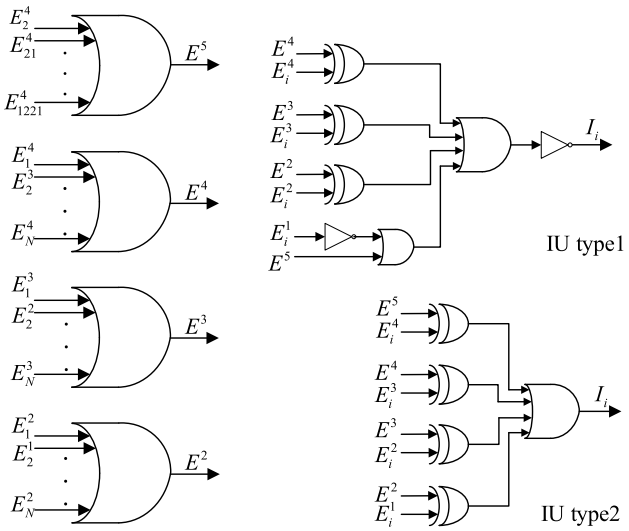


Fig. 18. The architecture of MSU.

N ECUs, the total area is $(2^{n+1} - n - 2)N$ and the delay is $\max_i\{\lceil \log_2 i \rceil + \lceil \log_2 C_n^i \rceil + 1\}$, where n is the number of inputs and $0 \le i \le n$. When the general design is adopted, the number of inputs $n = d_v + 2$ while in our design $n = d_v + 1$. Therefore, for a code with $d_v = 3$, the total area of N ECUs is reduced from $57N$ to $26N$ and the delay of the ECU is reduced from 7 to 5.

The architecture of MSU is shown in Fig. 18. Since the maximum energy value is larger than 1, E^1 must be 1 so the circuit used to calculate its value can be omitted. In this work, the locations with penalty term $\lambda_i = 1$ are called penalized locations and the bits in penalized locations are called penalized bits. As shown in Fig. 18, the second location, the 21-th location,..., and the 1221-th location are the penalized locations. Since $\lambda = 1$, the energy values of penalized bits should be increased by 1. Therefore, we combine the j-th term of these values with the $(j + 1)$-th term of others. For instance, E_2^3, E_{21}^3,..., and E_{1221}^3 are combined with E_1^4,..., E_N^4, which are the inputs of the N-input OR gate that generates E^4. Through this operation, the function of the penalty term is achieved while no additional operation is involved. Moreover, an N-input OR gate is replaced by an L-input OR gate, where L is the number of penalty locations and $L < N$. In the previous works [12], [14], the energy value is expressed in one-hot format. If the maximum energy value equals 4, it will be expressed by vector 01000. Therefore, the outputs of the N-input OR gates have to be entered to an additional circuit to calculate the maximum energy value. This will not only increase the hardware complexity but also increase the decoder delay because this part of the circuit is in the critical path of the whole decoder. By employing the modified one-hot format in this work, for the maximum energy value equals 4, it will be expressed by vector 01111 rather than 01000. Therefore, the outputs of the OR gates are the required maximum energy value so the delay of the whole decoder can be reduced.

The circuit used to calculate the indicator value of penalized bits is slightly different from the circuit for other bits. Type2 is the IU for the penalized bits and Type1 is the IU for others.

TABLE I

COMPARISON RESULTS OF THE REQUIRED HARDWARE RESOURCE

unit		ECU	MSU	InvIn
area	general InvIn	$57N$	$12N + 4$	$69N + 4$
	proposed InvIn	$26N$	$10N + L - 4$	$36N + L - 4$
delay	general InvIn	7	$6 + \log_2 N$	$13 + \log_2 N$
	proposed InvIn	5	$3 + \log_2 N$	$8 + \log_2 N$

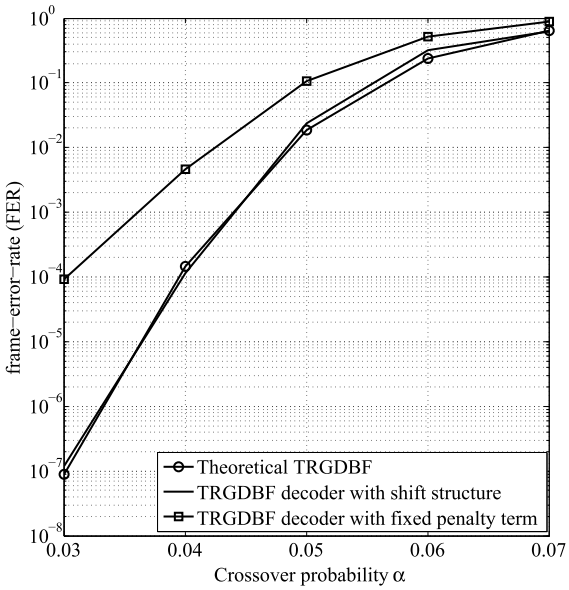


Fig. 19. Decoding performance of the TRGDBF decoder.

TABLE II

THE COMPARISON RESULT OF AVERAGE ITERATION NUMBERS IN EACH CROSSOVER PROBABILITY POINT

α	0.07	0.06	0.05	0.04	0.03
Theoretical TRGDBF	251.4	134.7	39.6	14.3	7.3
TRGDBF decoder	238.7	144.1	38.5	14.3	7.4

The total area of the proposed MSU is $(3d_v + 1)N + L - 4$ and the delay is $\lceil \log_2 N \rceil + \lceil \log_2(d_v + 1) \rceil + 1$.

Detailed comparison results of the required hardware resource between the proposed InvIn and the general InvIn are shown in Table I, in which a $d_v = 3$ code is applied. It is shown that the area of the modified InvIn is almost half of the original one. Moreover, the delay of the proposed InvIn is smaller than that of the general one. Therefore, our design is obviously superior to the general design. It is worth mentioning that since the area of the ECUs increases exponentially with d_v, the area reduction will be larger if a code with bigger d_v is adopted.

VI. DECODING PERFORMANCE AND IMPLEMENTATION RESULTS OF THE TRGDBF DECODER

The performance of the proposed TRGDBF decoder will be presented in this section. Moreover, the ASIC results of the proposed TRGDBF decoder are also reported. The hardware architecture is coded by RTL and synthesized under the TSMC 90nm CMOS technology.

TABLE III
THE SYNTHESIS AND COMPARISON RESULTS

Decoder	TRGDBF	VNSA-PGDBF [14]	PPBF [17]	GDBF [14]	MS [20]
(N,M)	(1296,648)	(1296,648)	(1296,648)	(1296,648)	(1296,648)
Technology	90nm	90nm	90nm	90nm	65nm
Frequency (MHz)	435(+17.6%) 435	370(+0.0%) 370	476(+28.6%) 476	385(+4.1%) 385	250* 181†
Area (mm^2)	0.299(-6.6%) 0.299	0.320(+0.0%) 0.320	0.367(+14.7%) 0.367	0.360(+12.5%) 0.360	0.72* 1.38†
N_c	1	1	1	1	6
T_{max}	300	300	300	300	20
Worst case Latency (ns)	690(-14.9%)	811(+0.0%)	630(-22.3%)	779(-3.9%)	663 †
Throughput @ T_{max} (Gbps)	1.88(+17.5%) 1.88	1.60(+0.0%) 1.60	2.06(+28.75%) 2.06	1.66(+3.75%) 1.66	2.7* 1.95 †
Area Efficiency (Gbps/mm^2)	6.29(+25.8%)	5.00(+0.0%)	5.61(+12.2%)	4.61(-7.8%)	1.41 †
T_{ave} @ α = 0.01	3.97 @ FER = $2e^{-8}$	4.83 @ FER = $8e^{-6}$	7.09 @ FER = $2e^{-6}$	2.95 @ FER = $3e^{-4}$	1.29 @ FER = $1e^{-7}$
Average case Latency (ns)	9.13(-30.0%)	13.05(+0.0%)	14.89(+14.1%)	7.66(-41.3%)	42.8 †
Throughput @ T_{ave} (Gbps)	142.0(+43.0%) 142.0	99.3(+0.0%) 99.3	87(-12.4%) 87	169(+70.2%) 169	41.9* 30.3†
Area Efficiency (Gbps/mm^2)	474.9(+54.5%)	310(+0.0%)	237(-23.5%)	470(+51.6%)	22.0†

* These are the original implementation results based on a 65nm CMOS technology. †These are the scaled results under the TSMC 90nm CMOS technology.

A. The Performances of the TRGDBF Decoder

Fig. 19 shows the decoding performance of the TRGDBF decoder as a function of crossover probability α. The dv4R050N1296 code is used for simulations without loss of generality. As mentioned in Section V-B, the TRGDBF decoder with fixed penalty suffers from severe performance degradation, proving the necessity of employing the shift-structure proposed in Section V-B. Moreover, there is only a negligible performance loss by employing the shift structure, which proves the feasibility of this modification. The average iteration numbers of the TRGDBF decoder and theoretical TRGDBF algorithm are also presented in Table II. It is shown that the average iteration number of the TRGDBF decoder is almost the same as the theoretical result. Hence, the performance of the TRGDBF decoder is reliable.

B. Synthesis Results

The synthesis results of the proposed TRGDBF decoder for the dv3R050N1296 QC-LDPC code are shown in Table III. From this table we know that the area of the TRGDBF decoder is 0.299 mm^2 and the frequency is 435MHz. As a soft-decision decoder which has a closer performance to the TRGDBF decoder, the area of the MS decoder [20] is 0.72 mm^2 and the frequency of it is 250MHz when synthesized under the 65nm technology. After being scaled to the 90nm technology, the area and frequency of it are 1.38 mm^2 and 181MHz, respectively. Therefore, the area of the TRGDBF decoder is only 21.7% of the MS decoder and the frequency is about 2.4 times that of the MS decoder. The proposed TRGDBF decoder can be easily reconfigured to suit other codes with the same block length. However, since the proposed TRGDBF decoder is specifically designed for the dv3R050N1296 code to obtain better synthesis results, it has the fixed V2C and C2V networks and cannot be adapted to support other codes at run-time like the MS decoder.

When compared to the PGDBF decoder, the TRGDBF decoder requires less area and works at a higher frequency. Note that our TRGDBF decoder also offers a far better error-correction performance than the PGDBF decoder. Though the PPBF decoder can operate at a higher frequency than the proposed TRGDBF decoder, it only provides a similar performance as that of the PGDBF decoder and requires more area.

The latency of the decoders in the worst case and in the average case are computed as $L_{worst} = f_{max} * T_{max} * N_c$ and $L_{ave} = f_{max} * T_{ave} * N_c$, respectively. We also present the throughput in these two cases, which are computed as $\theta_{worst} = \frac{f_{max}*N}{T_{max}*N_c}$ and $\theta_{ave} = \frac{f_{max}*N}{T_{ave}*N_c}$, respectively. f_{max} is the maximum working frequency of the decoder, N_c is the number of clock cycles needed for one decoding iteration, and T_{ave} is the average iteration number. Since the MS decoder presented in [20] takes the layered schedule, it needs 6 clock cycles to finish one iteration while the others only need one cycle.

In the worst case, the TRGDBF decoder offers a slightly less throughput than the MS decoder, which is higher than those of the PGDBF and GDBF decoders and less than that of the PPBF decoder. The PPBF decoder has the smallest latency due to the largest frequency. Moreover, the latency of the TRGDBF decoder is 690ns, which is higher than that of the MS decoder and smaller than those of the PGDBF and GDBF decoders. Due to less area, the proposed TRGDBF offers the best area efficiency among all decoders.

In the average case, the GDBF decoder has the largest average throughput and smallest latency. However, it suffers from a bad decoding performance so it is seldom used in practical applications. The MS decoder is far worse than other decoders in all throughput performance indexes and latency due to relatively high complexity. Moreover, since the PPBF decoder requires larger average iteration number, it has a worse throughput behavior and larger latency when compared to the TRGDBF and PGDBF decoders. Compared to the PGDBF decoder, the TRGDBF decoder increases the throughput by 43.0% and reduces the latency by 30.0%. Moreover, it has a much better area efficiency, which is 1.55 times that of the PGDBF decoder.

TABLE IV
THE AREA OF EACH UNIT OF THE TRGDBF DECODER

Unit	Total	VNUs	CNUs	InvIn	ReU
Area(mm^2)	0.299	0.079	0.004	0.116	0.100
Percentage	100%	26.4%	1.3%	38.8%	33.6%

Table IV lists the area of each unit of the TRGDBF decoder. As can be seen, the area of VNUs is the biggest. It is worth noting that the area of InvIn is only $0.116\ mm^2$, which proves that the modifications for this unit are very effective.

In conclusion, the proposed TRGDBF decoder offers a comparable error-correction performance to that of the soft-decision decoders while maintaining the high-throughput and low-complexity property of the hard-decision decoders. Hence, it has great potential in practical applications.

VII. CONCLUSION

In this work, we introduce a much-improved hard-decision decoding algorithm for LDPC codes, called Tabu-list Random-penalty GDBF algorithm. The random-penalty term is introduced to the inversion function to break TSs. Moreover, a tabu-list is employed to handle the repeated flip issue. These improvements are intelligently combined to achieve a significant performance gain when compared to the state-of-the-art hard-decision decoding algorithms. Simulation results demonstrate that the performance of the proposed TRGDBF algorithm is comparable to that of the offset min-sum algorithm with layered schedule when the number of iterations is sufficient. Besides, an efficient hardware architecture is developed for the TRGDBF algorithm. In order to avoid using complex random number generators, the random-penalty term is pre-generated. The received and estimated codeword vectors are cyclically shifted in each iteration to eliminate the performance loss caused by the fixed penalty term. The overall hardware complexity is further reduced by employing a well-optimized inversion indicator. Compared to the PGDBF decoder, the proposed decoder offers a higher throughput and requires less hardware resource. The high-performance and low-complexity property make the proposed TRGDBF decoder become a very competitive solution for LDPC decoding.

REFERENCES

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

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

[3] J. Chen, A. Dholakia, E. Eleftheriou, M. P. C. Fossorier, and X.-Y. Hu, "Reduced-complexity decoding of LDPC codes," *IEEE Trans. Commun.*, 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," *Problems Inf. Transmiss.*, vol. 11, no. 1, pp. 23–26, Jan. 1975.

[5] B. Unal, A. Akoglu, F. Ghaffari, and B. Vasić, "Hardware implementation and performance analysis of resource efficient probabilistic hard decision LDPC decoders," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 9, pp. 3074–3084, Sep. 2018.

[6] Y. Kou, S. Lin, and M. P. C. Fossorier, "Low-density parity-check codes based on finite geometries: A rediscovery and new results," *IEEE Trans. Inf. Theory*, vol. 47, no. 7, pp. 2711–2736, Nov. 2001.

[7] N. Miladinovic and M. P. C. Fossorier, "Improved bit-flipping decoding of low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 51, no. 4, pp. 1594–1606, Apr. 2005.

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

[9] P. O. Vontobel and R. Koetter, "Towards low-complexity linear-programming decoding," in *Proc. 4th Int. Symp. Turbo Codes Rel. Topics*, Munich, Germany, Apr. 2006, pp. 1–9.

[10] G. Sundararajan, C. Winstead, and E. Boutillon, "Noisy gradient descent bit-flip decoding for LDPC codes," *IEEE Trans. Commun.*, vol. 62, no. 10, pp. 3385–3400, Oct. 2014.

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

[12] K. Le, F. Ghaffari, D. Declercq, and B. Vasić, "Efficient hardware implementation of probabilistic gradient descent bit-flipping," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 64, no. 4, pp. 906–917, Apr. 2017.

[13] K. Le, F. Ghaffari, L. Kessal, D. Declercq, V. Savin, and O. Boncalo, "Lightweight hardware architecture for probabilistic gradient descent bit flipping on QC-LDPC codes," in *Proc. IEEE Int. Symp. Circuits Syst.*, Florence, Italy, May 2018, pp. 1–5.

[14] K. Le, D. Declercq, F. Ghaffari, L. Kessal, O. Boncalo, and V. Savin, "Variable-node-shift based architecture for probabilistic gradient descent bit flipping on QC-LDPC codes," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 7, pp. 2183–2195, Jul. 2018.

[15] K. Le, F. Ghaffari, D. Declercq, B. Vasić, and C. Winstead, "A novel high-throughput, low-complexity bit-flipping decoder for LDPC codes," in *Proc. Int. Conf. Adv. Technol. Commun.*, Quy Nhon, Vietnam, Oct. 2017, pp. 126–131.

[16] F. Ghaffari and B. Vasić, "Probabilistic gradient descent bit-flipping decoders for flash memory channels," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Florence, Italy, May 2018, pp. 1–5.

[17] K. Le *et al.*, "A probabilistic parallel bit-flipping decoder for low-density parity-check codes," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 1, pp. 403–416, Jan. 2019.

[18] M.-R. Li, L.-M. Jhuang, and Y.-L. Ueng, "A modified gradient descent bit flipping decoding scheme for LDPC codes," in *Proc. IEEE Int. Workshop Signal Process. Syst.*, Lorient, France, Oct. 2017, pp. 1–6.

[19] H. Li, H. Ding, and L. Zheng, "An escaping scheme for gradient descent bit-flipping decoding of LDPC codes," in *Proc. 9th Int. Congr. Image Signal Process.*, Datong, China, Oct. 2016, pp. 2026–2030.

[20] T. T. Nguyen-Ly, T. Gupta, M. Pezzin, V. Savin, D. Declercq, and S. Cotofana, "Flexible, cost-efficient, high-throughput architecture for layered LDPC decoders with fully-parallel processing units," in *Proc. Euromicro Conf. Digit. Syst. Design*, Limassol, Cyprus, Aug./Sep. 2016, pp. 230–237.

[21] C.-H. Hsu and A. Anastasopoulos, "Capacity-achieving codes with bounded graphical complexity and maximum likelihood decoding," *IEEE Trans. Inf. Theory*, vol. 56, no. 3, pp. 992–1006, Mar. 2010.

[22] T. Richardson, "Error floors of LDPC codes," in *Proc. 41st Annu. Allerton Conf. Commun., Control, Comput.*, Oct. 2003, pp. 1426–1435.

[23] V. L. Orozco and S. Yousefi, "Trapping sets of fountain codes," *IEEE Commun. Lett.*, vol. 14, no. 8, pp. 755–757, Aug. 2010.

[24] H. Cui, J. Lin, and Z. Wang, "An efficient post-processor for lowering the error floor of LDPC codes," *IEEE Trans. Circuits Syst. II, Express Briefs*, vol. 66, no. 3, pp. 397–401, Mar. 2019.

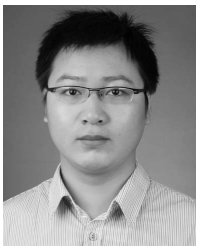
[25] D. J. C. MacKay. (Accessed: 20, 2014). *Encyclopedia Sparse Graph Codes*. [Online]. Available: http://www.inference.phy.cam.ac.uk/mackay/codes/data.html

[26] B. Yuce, H. F. Ugurdag, S. Gören, and G. Dündar, "Fast and efficient circuit topologies forfinding the maximum of n k-bit numbers," *IEEE Trans. Comput.*, vol. 63, no. 8, pp. 1868–1881, Aug. 2014.



Hangxuan Cui received the B.S. degree in signal processing from Northwestern Polytechnical University, Xi'an, China, in 2017. He is currently pursuing the Ph.D. degree from Nanjing University.

His research interests include error-correction code algorithms and their efficient very large-scale integration (VLSI) implementations, focusing on concatenated codes and low-density parity-check (LDPC) decoders.



Jun Lin (S'14–SM'18) received the B.S. degree in physics and the M.S. degree in microelectronics from Nanjing University, Nanjing, China, in 2007 and 2010, respectively, and the Ph.D. degree in electrical engineering from Lehigh University, Bethlehem, PA, USA, in 2015.

From 2010 to 2011, he was an Application Specified Integrated Circuit (ASIC) Design Engineer with AMD. In 2013, he was an Intern with Qualcomm Research, Bridgewater, NJ, USA. In 2015, he joined the School of Electronic Science and Engineering, Nanjing University, where he is currently an Associate Professor. His current research interests include low-power high-speed very large-scale integration (VLSI) design, specifically VLSI design for digital signal processing and cryptography. He is a member of the Design and Implementation of Signal Processing Systems (DISPS) Technical Committee of the IEEE Signal Processing Society. He was a co-recipient of the Merit Student Paper Award at the IEEE Asia Pacific Conference on Circuits and Systems in 2008. He was a recipient of the 2014 IEEE Circuits and Systems (CAS) Society Student Travel Award.



Zhongfeng Wang (M'00–SM'05–F'16) received the B.E. and M.S. degrees from Tsinghua University, Beijing, China, and the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN, USA, in 2000.

He was a leading Very Large-Scale Integration (VLSI) Architect with Broadcom Corporation for nearly nine years, where he has contributed significantly on 10 Gb/s and beyond high-speed networking products. In addition, he has made critical contributions in defining FEC coding schemes for 100- and 400-Gb/s Ethernet standards. So far, his technical proposals have been adopted by many networking standard specs. He joined Nanjing University as a Distinguished Professor. From 2003 to 2007, he was an Assistant Professor with the School of EECS, Oregon State University, Corvallis. He was with National Semiconductor Corporation. He has published over 100 technical papers, edited one book (VLSI), and filed tens of U.S. patent applications and disclosures. His current research interests are in the area of VLSI for high-speed signal processing systems.

Dr. Wang was a world-recognized expert on VLSI for forward error correction codes. He was a recipient of the IEEE Circuits and Systems (CAS) Society VLSI Transactions Best Paper Award in 2007. In the current record, he has had five papers ranked among top twenty most downloaded manuscripts in the IEEE TRANSACTIONS ON VLSI SYSTEMS since 2007. Since 2004, he served as an Associate Editor for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS I (TCAS-I), TCAS-II, and the IEEE TRANSACTIONS ON VLSI SYSTEMS for many terms. He also served as a technical program committee member, the session chair, the track chair, and a review committee member for many IEEE and ACM conferences. In 2013, he served for the Best Paper Award Selection Committee of the IEEE CAS Society.