Fault-Tolerant Probabilistic Gradient-Descent Bit Flipping Decoder

Fault-Tolerant Probabilistic Gradient-Descent Bit Flipping Decoder

Omran Al Rasheed, *Student Member, IEEE,* Predrag Ivaniš, *Member, IEEE,* and Bane Vasić, *Fellow, IEEE*

***Abstract*—We propose a gradient descent type bit flipping algorithm for decoding low density parity check codes on the binary symmetric channel. Randomness introduced in the bit flipping rule makes this class of decoders not only superior to other decoding algorithms of this type, but also robust to logic-gate failures. We report a surprising discovery that for a broad range of gate failure probability our decoders actually benefit from faults in logic gates which serve as an inherent source of randomness and help the decoding algorithm to escape from local minima associated with trapping sets.**

***Index Terms*—Bit-flipping algorithm, low-density parity check codes, decoding by unreliable hardware, fault-tolerance.**

I. Introduction

It is now widely accepted that design of low-energy consumption Very Large Scale Integration (VLSI) systems must incorporate the fact that due to lower supply voltages and variations in the technological process, emerging nano-scale devices are inherently unreliable [1]. The reliable storage of data in a memory built of unreliable logic gates with transient failures can be achieved by employing low density parity check (LDPC) codes and simple bit-flipping (BF) decoding [2]. Density evolution of the sum-product algorithm (SPA) by Varshney [3], min-sum algorithm [4], FAID algorithm by Huang *et al.* [5] and Gallager B algorithm [6], [7] demonstrate robustness of these more complex decoding algorithms.

However, when only bit hard decisions are available, the decoder must rely on the BF or Gallager A/B algorithms. Since the performance of the BF decoder is typically inferior when compared to the Gallager-B algorithm it is of importance to try to close this gap by using more powerful variants of the BF decoder yet simpler than Gallager A/B algorithms. Although the previous results suggest a possibility that these decoders might be robust in the presence of gate failures, no analysis exists in the literature to prove or disprove this belief.

In this letter we consider fault-tolerant BF on the binary symmetric channel (BSC). The decoder we present is inspired by two decoders: Wadayama's Gradient Descent Bit Flipping (GDBF) [8] and Miladinovic and Fossorier's Probabilistic Bit Flipping (PBF) [9]. The GDBF was designed for the Additive White Gaussian Noise (AWGN) channel and shown to provide good balance between steady convergence and decoding speed. Inserting noise in all variable nodes in every particular iteration, as a means of improving the GDBF on the AWGN channel was proposed by Sundararajan *et al.* [10], and termed the noisy GDBF (NGDBF).

In PBF, code bits with a number of unsatisfied check sums larger than a fixed threshold are flipped with some probability, which is adapted throughout the iterations. By combining the ideas of [8] and [9] with some critical improvements following from our intuition on fault-tolerant decoders, we design a hard decision decoder, which we call the Probabilistic GDBF (PGDBF) decoder, resilient to logic gate failures.

Manuscript received June 25, 2014. This work was supported by the Seventh Framework Programme of the European Union, under Grant Agreement number 309129 (i-RISC project).

The authors I especially thank the editor and two anonymous reviewers for their constructive comments, which helped us to improve the manuscript..

O. Al Rasheed and P. Ivaniš are with the Faculty of Electrical Engineering, University of Belgrade, Serbia (e-mail: omrano84@hotmail.com, predrag.ivanis@etf.rs), B. Vasić is with the Department of Electrical and Computer Engineering, University of Arizona, Tucson, AZ, 85721 USA.

Digital Object Identifier

II. Preliminaries

Let G denote the Tanner graph of an (N, K) binary LDPC code $\mathcal{C}$ of rate $R = K/N$, which consists of the set of N variable nodes V and the set of M check nodes C. The parity check matrix H is the bi-adjacency matrix of G. Two nodes in G are neighbors if there is an edge between them. The degree of a node v is the number of its neighbors and is denoted as γ_v. Graph G is said to be γ-variable-regular if all variable nodes in V have the same degree, $\gamma_v = \gamma$. The degree of a check node c, ρ_c, and ρ-check-regular codes are defined analogously. The sets of neighbors of nodes v and c are denoted as $\mathcal{N}_v$ and $\mathcal{N}_c$, respectively.

Let $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ denote a codeword of $\mathcal{C}$ that is transmitted over a BSC with crossover probability α, where x_v denotes the value of the bit associated with variable node v and let the vector received by a decoder from the BSC be $\mathbf{y} = \{y_1, y_2, \ldots, y_N\}$. $\mathbf{e} = (e_1, e_2, \ldots, e_N)$ denotes the *error pattern* introduced by the BSC such that $\mathbf{y} = \mathbf{x} \oplus \mathbf{e}$, and $\oplus$ is the component-wise modulo-two sum.

We consider iterative decoders which at the l-th iteration ($l \in [0, L]$, where L is the maximal number of iterations) produce the estimated codeword $\hat{\mathbf{x}}^{(l)}$ as an output. The GDBF algorithm is based on the calculation of the inverse function, defined as [8, Eqn. (6)]

$$\Delta_v^{(l)}(\chi, \eta) = \chi_v^{(l)} \eta_v + \sum_{c \in \mathcal{N}_v} \prod_{u \in \mathcal{N}_c} \chi_u^{(l)}, \tag{1}$$

where $\boldsymbol{\eta} = (-1)^{\mathbf{y}}$ and $\boldsymbol{\chi}^{(l)} = (-1)^{\hat{\mathbf{x}}^{(l)}}$ denote the "bipolar" versions of $\mathbf{y}$ and $\hat{\mathbf{x}}^{(l)}$. The estimate of a variable node v is initialized as $\chi_v^{(0)} = \eta_v$, and in the l-th iteration, the values $\Delta_v^{(l)}$ are calculated for all variable nodes v, and (in single-bit flipping mode of GDBF [8]) only the symbols with minimum value of the inverse function are inverted to obtain $\chi_v^{(l+1)}$.

III. GDBF Decoding Algorithm For BSC

The original GDBF was designed for the additive white Gaussian noise (AWGN) channel, where η are real valued vectors. To adapt it to the BSC, we first rewrite the polar-based inverse function in binary form. Since $\chi_v^{(l)} = 1 - 2\hat{x}_v^{(l)}$ and $\eta_v = 1 - 2y_v$, by using modulo-2 arithmetic, the inverse function can be simplified into

$$\Delta_v^{(l)}(\hat{\mathbf{x}}, \mathbf{y}) = 2 - 2(\hat{x}_v^{(l)} \oplus y_v) + \gamma_v - 2 \sum_{c \in \mathcal{N}_v} \bigoplus_{u \in \mathcal{N}_c} \hat{x}_u^{(l)}. \quad (2)$$

For γ-variable-regular codes, the above expression is minimized by maximization of the following modified inverse function

$$\Lambda_v^{(l)}(\hat{\mathbf{x}}, \mathbf{y}) = \hat{x}_v^{(l)} \oplus y_v + \sum_{c \in \mathcal{N}_v} \bigoplus_{u \in \mathcal{N}_c} \hat{x}_u^{(l)}. \quad (3)$$

Let $b^{(l)}$ be the largest value of the modified inverse function at the l-th iteration, i.e., $b^{(l)} = \max_v(\Lambda_v^{(l)}(\hat{\mathbf{x}}, \mathbf{y}))$. If the flipping decision was wrong, the "flip messages" would propagate through the short cycles in G. To minimize this effect, it is reasonable to flip a small number of bits per iteration. In the case of the AWGN channel, this typically results in flipping of one bit per iteration (with minimum value of $\Delta_v^{(l)}(\hat{\chi}, \eta)$). In the BSC the range of the modified inverse function is restricted to the set of integer values $[0, \gamma + 1]$, and usually more than one variable node satisfies the relation $\Lambda_v^{(l)}(\hat{\mathbf{x}}, \mathbf{y}) = b^{(l)}$, which has a negative impact on the algorithm convergence. On the other hand, as we show in the next section, the restrictions of the range of $\Lambda_v^{(l)}$ makes BSC version of GDBF decoder much less sensitive to logic gate failures in the decoder.

IV. Fault-Tolerant PGDBF Decoder For BSC

In this section, we define formally the PGDBF decoder, and suggest a possible implementation in faulty hardware. According to Eqn. (3), for a (γ, ρ)-regular code it is necessary to calculate the parities in the neighboring check nodes, by using ρ-input exclusive or (XOR) gates. An additional two-input XOR gate is required to check if the v-th bit of the current estimate is the same as the bit initially estimated from the channel. The value of $\Lambda_v^{(l)}(\hat{\mathbf{x}}, \mathbf{y})$ is equal to the number of non-zero outputs of the XOR gates. Combinational logic at the variable nodes is based on a set of majority logic (MAJ) gates, each having $\gamma + 1$ inputs and adaptable threshold. The output of the v-th MAJ gate in the l-th iteration is non-zero only if the modified inverse function value is equal to the threshold value $b^{(l)}$. This threshold is the same for every variable node.

The new decoder is given in Algorithm 1, and Fig. 1 shows the hardware structure of a corresponding variable node processor. The main novelty we propose is the following. In the GDBF decoder for BSC $\Lambda_v^{(l)} = b^{(l)}$ was a sufficient condition for flipping, and the refreshed estimation is calculated as $\hat{x}_v^{(l+1)} = 1 \oplus \hat{x}_v^{(l)}$ if $\Lambda_v^{(l)}(\hat{\mathbf{x}}, \mathbf{y}) = b^{(l)}$ and $\hat{x}_v^{(l+1)} = \hat{x}_v^{(l)}$ if $\Lambda_v^{(l)}(\hat{\mathbf{x}}, \mathbf{y}) < b^{(l)}$. It is stored in the register inside the decoder, and used for calculation of $\Lambda_v^{(l+1)}(\hat{\mathbf{x}}, \mathbf{y})$. In our algorithm, even if $\Lambda_v^{(l)} = b^{(l)}$, the observed bit $\hat{x}_v$ will not be flipped automatically - instead it will be flipped with a predefined probability p. In Algorithm 1 this is done by multiplying the flipping decision with the Bernoulli $B(1, p)$ random variable a_v. As we show in the next section, this modification is critical for ensuring the resilience to gate failures. In hardware, it can be realized by adding to each variable node processor one AND gate and a generator of Bernoulli random variables a_v with $\Pr(a_v = 1) = p$.

Algorithm 1 Probabilistic GDBF Algorithm

Input: $\mathbf{y}$
$\forall v \in V: \ \hat{x}_v^{(0)} \leftarrow y_v$
$\mathbf{s}^{(0)} \leftarrow \hat{\mathbf{x}}^{(0)} H^T \ (\forall c \in C: \ s_c^{(0)} \leftarrow \bigoplus_{u \in \mathcal{N}_c} \hat{x}_u^{(0)})$
$l = 0$
while $\mathbf{s}^{(l)} \neq \mathbf{0}$ and $l \leq L$ **do**
  $\forall v \in V$: Compute $\Lambda_v^{(l)}(\hat{\mathbf{x}}, \mathbf{y}))$
  $b^{(l)} \leftarrow \max_v(\Lambda_v^{(l)}(\hat{\mathbf{x}}, \mathbf{y})))$
  $v = 1$
  while $v \leq N$ **do**
    if $\Lambda_v^{(l)}(\hat{\mathbf{x}}, \mathbf{y}) = b^{(l)}$ **then**
      $\hat{x}_v^{(l+1)} \leftarrow a_v \oplus \hat{x}_v^{(l)}$
    else
      $\hat{x}_v^{(l+1)} \leftarrow \hat{x}_v^{(l)}$
    end if
    $v \leftarrow v + 1$
  end while
  $\mathbf{s}^{(l+1)} \leftarrow \hat{\mathbf{x}}^{(l+1)} H^T$
  $l \leftarrow l + 1$
end while
Output: $\hat{\mathbf{x}}^{(l)}$

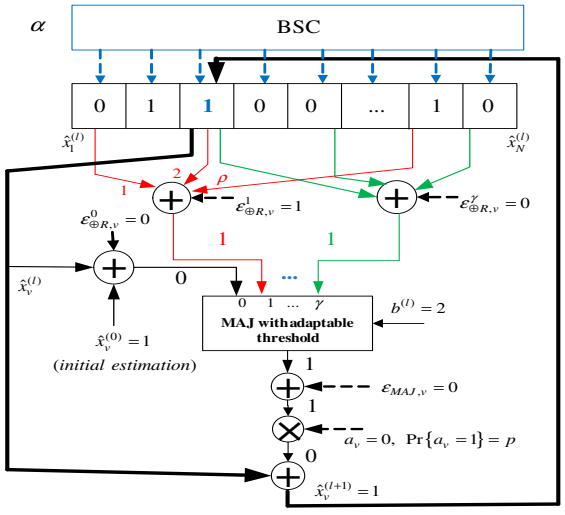

Fig. 1. Illustration of the variable node processing unit, operations performed in the l-th iteration. The randomness in the v-th variable node is modeled by adding a Bernoulli random variable $\varepsilon_{MAJ,v}$, and flipping the output of the c-th XOR is modeled by adding Bernoulli random variable $\varepsilon_{\oplus R,v}^c$.

The threshold $b^{(l)}$ may be initialized to the maximum value $(\gamma + 1)$, and decremented in those iterations when all MAJ gate outputs are zero. When the output of at least one MAJ gate is not equal to zero, the threshold is set to $b^{(l)} = \max_v(\Lambda_v^{(l)}(\hat{\mathbf{x}}, \mathbf{y}))$. The PGDBF results in a low

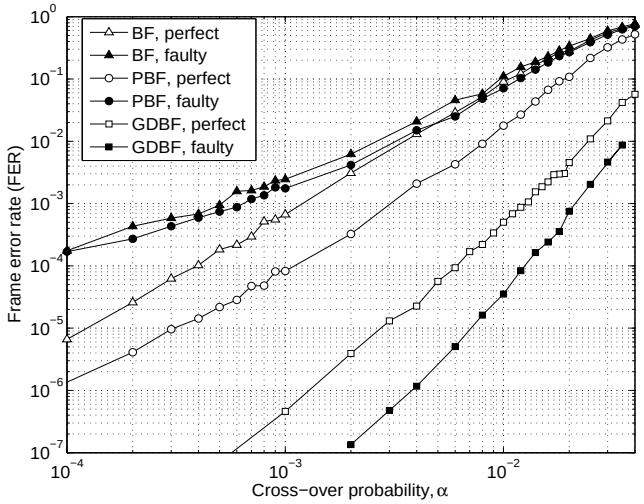

Fig. 2. FER performance comparison for the $(155, 64)$ Tanner code, perfect: $P_{MAJ} = 0$, $P_{\oplus,R} = 0$, faulty: $P_{MAJ} = 10^{-3}$, $P_{\oplus,R} = 10^{-2}$.

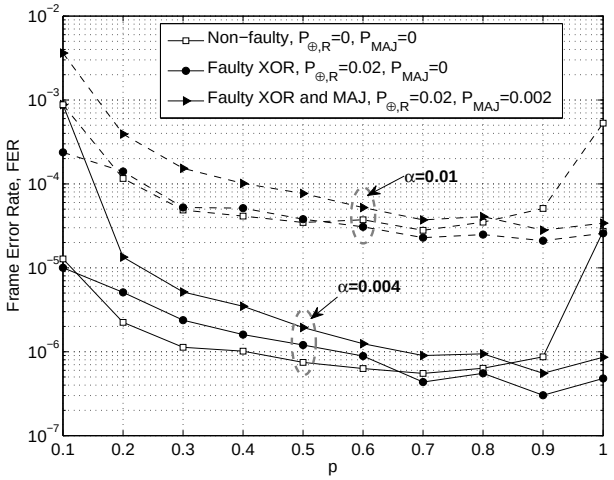

Fig. 3. Impact of the parameter p on PGDBF optimized for the BSC realized in faulty hardware. The plot is for the $(155, 64)$ Tanner code, $\alpha = 4 \times 10^{-3}$, $\alpha = 10^{-2}$ and $L = 100$.

complexity decoder that can be realized by using only XOR and MAJ logic gates, while the NGDBF decoder requires real-valued operations both for the inverse function calculation and for processing the AWGN samples inserted in variable nodes [10]. Moreover, in the NGDBF the threshold is adapted separately for every node, while in the PGDBF the threshold is the same for all variable nodes during the iteration.

V. Frame Error Rate Performance Analysis in the Presence of Hardware Failures

To assess the performance of the proposed decoding algorithm, we consider the canonical transient von-Neumann logic gate failure mechanism in which the failures in different gates and different time instants are independent and identically distributed. The failures manifest themselves as random bit flips at the gate outputs. All XOR gates have faulty probability $P_{\oplus}$, failures in the MAJ gates occur with probability P_{MAJ}, and the probability that a bit written in a register inside the decoder is incorrectly reconstructed is P_R. Computation of all other quantities is performed with perfect hardware.

The probability of flipping the output of a ρ-input XOR depends on the reliability of the memory cells connected to the gate inputs and the failure probability of the gate itself. Therefore, their combined effect can be modeled by flipping the output of the XOR gate with probability $P^c_{\oplus,R}$, which is obtained by using [6, Eqn. (3)] and given by

$$P^c_{\oplus,R} = \frac{1-(1-2P_R)^{q_c}}{2}(1-P_{\oplus}) + \frac{1+(1-2P_R)^{q_c}}{2}P_{\oplus}. \quad (4)$$

In the above expression, q_c corresponds to the number of inputs of the XOR gate connected with the register ($q_0 = 2$ for the gate that calculates $\hat{x}_v^{(l)} \oplus y_v$, and $q_c = \rho$ for parity check gates). The net result of this transformation is that $P^c_{\oplus,R}$ defines the probability of failure in the c-th check node, while P_{MAJ} corresponds to the variable nodes.

Now we present the numerical results of Monte Carlo simulations for a girth-8 regular codes with $\gamma = 3$. For a given crossover probability of BSC, α, and the failure probabilities $P_{\oplus,R}$, P_{MAJ}, the frame error rate (FER) of the faulty PGDBF decoder is estimated and compared with the existing decoders.

In Fig. 2, the FER performance of the $(155, 64)$ Tanner code is presented for the maximum number of decoding iterations $L = 100$. The results are presented for the BF, PBF and GDBF algorithms. In the case of non-faulty hardware in the decoder, GDBF algorithm results in lower FER values compared to the BF and PBF algorithms. As expected, the performances of faulty BF and PBF decoders are significantly degraded. For the case when $P_{\oplus,R} = 10^{-2}$, $P_{MAJ} = 10^{-3}$, the performance of the two are approximately the same. On the other hand, for the same failure rates, performance of the GDBF are improved compared to the non-faulty decoder case! This surprising effect is related to the finite set of possible values of $\Lambda_v^{(l)}(\hat{\mathbf{x}}, \mathbf{y})$. In such a case, the hardware failures can change its values in different variable nodes and the decoder could escape from a trapping set, as will be explained in Fig. 4. Therefore, the first goal is to optimize the parameter p in PGDBF to reduce the FER values for a wider range of the failure rates.

Fig. 3 shows the FER for the same code and fixed α, for various values of the parameter p. Numerical results are presented for the non-faulty case as well as for the case when $P_{\oplus,R} = 2 \times 10^{-2}$ and $P_{MAJ} = 0$ or $P_{MAJ} = 2 \times 10^{-3}$. In the non-faulty case it can be observed that the PGDBF decoder has the best performance for $p \approx 0.7$ while the best performances for the faulty case are obtained for $p \approx 0.9$. The optimal value of parameter p does not depend significantly on the BSC crossover probability, in contrast to NGDBF where the variance of the noise inserted to the variable nodes has to be approximately equal to the variance of the noise in the channel. For the case of perfect (reliable) logic gates, the PGDBF significantly reduces the FER compared to the GDBF (where $p = 1$). We have verified by simulation that the PGDBF decoder is almost insensitive to the logic gate failures in the range $P_{\oplus,R} < 3 \times 10^{-2}$ and $P_{MAJ} \leq 3 \times 10^{-3}$ for $p = 0.7$.

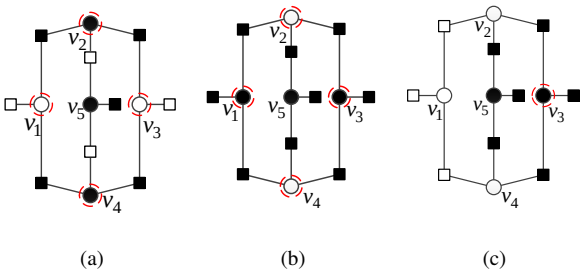

Fig. 4. Error patterns: (a) $l = 1$, GDBF; (b) $l = 2$, non-faulty GDBF; (c) $l = 2$, faulty GBDF / non-faulty PGBDF.

In Fig. 4 we present one motivating example based on an error pattern for a girth-8 code with $\gamma = 3$, known to be uncorrectable by using the parallel BF algorithm, as it corresponds to a trapping set [11]. Fig. 4(a) illustrates this error pattern, where variable nodes are denoted by circles, the check nodes with squares, where the full markers correspond to binary one and empty to binary zero. The variable nodes marked with the dashed circles have maximum value $\Lambda_v^{(1)} = 2$ and these nodes are flipped in the GDBF algorithm. In the second iteration, shown in Fig. 4(b), we obtain $\Lambda_v^{(2)} = 4$ for the same symbol nodes as in the previous iteration, and further convergence is not possible. However, if v_1 is flipped before the second iteration of the GDBF algorithm, due to a hardware failure, we obtain the pattern in Fig. 4(c) and decoding is successful after one additional iteration.

A similar effect can be observed even in the perfect decoder, if we intentionally avoid the flipping of one bit that satisfies the necessary condition. Although four variable nodes in Fig. 4(a) satisfy it, due to the probabilistic nature of the algorithm, only v_2, v_3 and v_4 can be actually flipped and the second iteration of non-faulty PGDBF is illustrated in Fig. 4(c). In such a case, the probabilistic approach results in the same effect as the hardware failure that inverts one of the variable bits inside the trapping set. If $p = 3/4$ three out of four bits inside the length-8 cycle from the above example should be flipped on average, and if failures are present in the logic gates the value of the parameter p can be increased. Although the exact analysis is nontrivial, this seems to be a good explanation for the results shown in Fig. 3. In contrast to the PBF [9], the value of p is constant during the iterations and it is large enough so the decoding process is not slowed down significantly.

In Fig. 5 we present the FER performances for a $(732, 551)$ quasi-cyclic (QC) code [11]. In the presence of failures in the decoder, the performance is degraded for the PBF and Gallager-B decoders, but is improved for the GDBF $(p = 1)$. For the case of PGDBF with $p = 0.8$, the performances are approximately the same for the faulty and non-faulty cases. Further simulations indicate that PGDBF has approximately the same performance if $P_{MAJ} \leq 1/(2N)$ and $P_{\oplus,R} \leq 5/N$.

VI. Conclusion and Discussion

By combining the ideas of GDBF and PBF, we have designed a hard decision decoder resilient to logic gate failures. In our approach, the probabilistic flipping is applied only to the variable nodes that satisfy a necessary condition for flipping. The corresponding flipping probability is fixed during iterations, the tuning of the parameters is simpler compared to previously proposed algorithms, and results in a minor increase of the decoding latency. Furthermore, we have shown that the proposed decoder not only has large immunity to gate failures but, surprisingly, can utilize the hardware failures to improve the decoding performance. We have considered some example QC LDPC codes and shown that the proposed algorithm is insensitive to hardware unreliability for a wide range of failure rates in combinational logic and memory cells in the decoder.

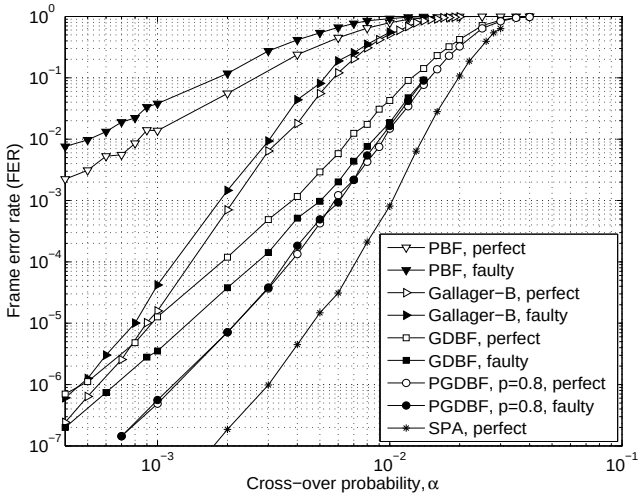

Fig. 5. FER for PGDBF algorithm, BSC, QC$(732, 551)$, $L = 100$, perfect: $P_{MAJ} = 0$, $P_{\oplus,R} = 0$, faulty: $P_{MAJ} = 2 \times 10^{-4}$, $P_{\oplus,R} = 2 \times 10^{-3}$.

References

[1] S. Ghosh and K. Roy, "Parameter variation tolerance and error resiliency: New design paradigm for the nanoscale era," *Proceedings of the IEEE*, vol. 98, no. 10, pp. 1718–1751, Oct. 2010.

[2] B. Vasic and S. K. Chilappagari, "An information theoretical framework for analysis and design of nanoscale fault-tolerant memories based on low-density parity-check codes," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 54, no. 11, pp. 2438–2446, Nov. 2007.

[3] L. Varshney, "Performance of LDPC Codes Under Faulty Iterative Decoding," *IEEE Transactions on Information Theory*, vol. 57, no. 7, pp. 4427–4444, July 2011.

[4] A. Balatsoukas-Stimming and A. Burg, "Density evolution for min-sum decoding of LDPC codes under unreliable message storage," *IEEE Communications Letters*, vol. 18, no. 5, pp. 849–852, May 2014.

[5] C.-H. Huang and L. Dolecek, "Analysis of finite-alphabet iterative decoders under processing errors," in *Proc. 2013 IEEE Int. Conf. Acoustics, Speech, Sig. Proc.*, May 2013, pp. 5085–5089.

[6] F. Leduc-Primeau and W. Gross, "Faulty Gallager-B decoding with optimal message repetition," in *Proc. 50th Annual Allerton Conference on Communication, Control, and Computing*, Oct 2012, pp. 549–556.

[7] S. Tabatabaei Yazdi, H. Cho, and L. Dolecek, "Gallager B Decoder on Noisy Hardware," *IEEE Transactions on Communications*, vol. 61, no. 5, pp. 1660–1673, May 2013.

[8] 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, June 2010.

[9] N. Miladinovic and M. Fossorier, "Improved Bit-Flipping decoding of low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 51, no. 4, pp. 1594–1606, April 2005.

[10] G. Sundararajan, C. Winstead, and E. Boutillon, "Noisy Gradient Descent Bit-Flip Decoding for LDPC Codes," 2014. [Online]. Available: http://arxiv.org/abs/1402.2773.

[11] D. Nguyen, S. Chilappagari, M. Marcellin, and B. Vasic, "On the Construction of Structured LDPC Codes Free of Small Trapping Sets," *IEEE Trans. Inform. Theory*, vol. 58, no. 4, pp. 2280–2302, April 2012.