Probabilistic Gradient Descent Bit-Flipping Decoders for Flash Memory Channels

Fakhreddine Ghaffari*, Bane Vasic†

*ETIS, UMR-8051, Université Paris Seine, Université de Cergy-Pontoise, ENSEA, CNRS, France,
fakhreddine.ghaffari@ensea.fr

†Dep. of Electrical and Computer Engineering, Uni. of Arizona, Tucson, AZ, 85719 USA, vasic@email.arizona.edu

***Abstract*—Low-density parity check (LDPC) codes are an attractive error correction scheme for ensuring data integrity in new generation of NAND flash memories. A quick assessment of the iterative decoders for LDPC codes reveals a wide range of varying complexities. The simple Bit-Flipping (BF) and binary-message-passing algorithms such as the Gallager A/B algorithms occupy one end of the spectrum, while Belief Propagation (BP) and A Posteriori Probability (APP) decoders lie at the other end. The gamut of existing decoders filling the intermediate space can simply be understood as the implementation of BP (and its variants, such as the min-sum algorithm) at different levels of message precision. Decoders with low-precision messages are desirable because of their low complexity and power efficiency, but in such decoders it is highly nontrivial to prevent performance degradation known to as error floor and to guarantee fast convergence to a codeword. In this paper we present our results on a new class of low-complexity iterative decoders for flash memory channels. They involve two main innovations: global computation and randomness. Our decoding algorithm, Probabilistic Gradient Descent Bit-Flipping (PGDBF) is motivated by the analogy between Tanner graphs and the graphical models used in statistical mechanics, and prescribe a rule for flipping a bit based on the so-called energy function and a binary random sequence associated to that bit. Energy function is computationally simple, but involves all the bits. We present the PGDBF algorithm analysis, explain how it benefits from global computation and randomness, and present the hardware synthesis results as well as comparisons with the state-of-the-art decoders.**

Keywords*—*Low-Density Parity-Check, Probabilistic Gradient Descent Bit Flipping, NAND Flash Memory, high decoding throughput, low-complexity FPGA implementation.

I. Introduction

NAND FLASH memories are used for storing information. Nearly all external storage memories (MultiMedia Card (MMC), Secure Digital (SD) card and Memory Stick (MS) card) use this technology. However manufacturers of NAND FLASH memories, generally, do not guarantee at 100% the integrity of data stored in it but usually they propose a lower error rate under a given limit. This limited reliability requires the implementation of an error management system (ECC - Error Correction Code, Bad Block Management, etc.) at the application level. This is the case for example, for hard disks SSDs (Solid State Drives). The LDPC based ECC are an interesting error correction approach for increasing data reliability in new generation of NAND flash memories. QC-LDPC (Quasi-Cyclic-LDPC) are a particular code constructing method of LDPC codes that are well-suited for the practical implementations. QC-LDPC can be effectively decoded by either *soft-information* message passing decoders, *i.e.* Min Sum (MS), Sum Product (SP) [1], or *hard-decision* decoders, *i.e.* Bit Flipping (BF), Gallager-A, B [2]. This paper focuses on the BF decoder due to its low-complexity as well as its high possible throughput. Additionally, recent BF optimizations proposed in [3][4] demonstrated good error correction capability, approaching the soft-decision decoders.

The PGDBF, proposed by Rasheed *et al.* in[3], is one of the best BF decoders in term of error correction on Binary Symmetric Channel (BSC) channel. PGDBF continually passes 1-bit messages between two groups of computing units: Variable Nodes (VNs) and Check Nodes (CNs). The CN computes the parity check function while the VN computes a so-called energy value. The difference between PGDBF and its deterministic version (the GDBF [5]) is that, all the VNs having the maximum energy will be flipped in GDBF while in PGDBF only a randomly selected subset is flipped. This process is controlled by a value p_0 ($p_0 < 1$).

The outstanding decoding performance of PGDBF comes at the price of the additional complexity needed to implement the probabilistic signal generator, compared to GDBF. Indeed, FPGA implementation results reported in [6] have shown a $\times 8$ and a $\times 1.64$ increase in registers and Look-Up Tables (LUTs) respectively, compared to the non-probabilistic GDBF decoder. ASIC results from [7] have shown a $5\% - 14\%$ cost increase of PGDBF with respect to GDBF. Note that these results have been obtained for the same QC-LDPC code. This high cost in complexity is also accompanied with loss in decoding throughput due to lower maximum frequency of the PGDBF decoder compared with GDBF. Moreover the main constraint for designing an LDPC code to be used for the error correction in Nand Flash memory is to keep the highest possible code rate. This constraint is intrinsic to this application in order to optimize the memory space for storing useful bits (data) in external embedded memories.

In this paper, we introduce a new LDPC decoder algorithm called FM-PGDBF for Flash Memory adapted PGDBF decoder. It consists to adapt the well known PGDBF decoder to be efficiently used for applications using NAND Flash memory. To increase the throughput of our decoder we propose to remove the most critical computation part consisting to compute the maximum of energy among N computing units (N is the size of the codeword). We replace this maximum finder unit by an off-line prediction of the a energy maximum for each decoding iteration. Our simulation results show that this imprecision induced by errors in the offline prediction of the maximum energy improves the decoding performances of the PGDBF decoder by shaking it to escape most of trapping sets. This new version of PGDBF implementations are shown to not only reduce the decoder complexity but also improve decoding performance with respect to the state-of-the-art implementations.

The paper is organized as following. In Section II, the PGDBF algorithm and the optimized implementation in the literature are reviewed. Section III introduces the principle of our Flash memory adapted PGDBF (FM-PGDBF) and its hardware implementation. In Section IV, we show for several test codes that, our FM-PGDBF provides a decoding performance as good as the optimized PGDBF implementation in literature while reducing 60% the hardware cost comparing with GDBF. The adapted PGDBF for NAND Flash memory provides a significant gain in error correction, around 2dB gain over the traditional GDBF. Section V concludes the paper.

II. PGDBF Decoding and its implementation

A. Notations

An LDPC code is defined by a sparse parity-check matrix H with size (M, N), where $N > M$. A codeword is a vector $\mathbf{x} = (x_1, x_2, \ldots, x_N) \in \{0,1\}^N$ which satisfies $H\mathbf{x}^T = 0$. We denote by $\mathbf{y} = \{y_1, y_2, \ldots, y_N\} \in \{0,1\}^N$ the output of a Flash Memory channel. We simplify the Flash Memory channel is as a BSC model in which the bits of the stored codeword $\mathbf{x}$ have been flipped with crossover probability α and leave the other complex characterizations such as the unbalance flip *i.e.* the flip probability from 1 to 0 is different from the flip probability from 0 to 1 as the future works. The graphical representation of an LDPC code is a bipartite graph called Tanner graph composed of two types of nodes, the Variable Node Unit (VNUs) v_n, $n = 1, \ldots, N$ and the Check Node Unit (CNUs) c_m, $m = 1, \ldots, M$. In the Tanner graph, a VNU v_n is connected to a CNU c_m if $H(m,n) = 1$. Let us also denote $\mathcal{N}(v_n)$ the set of CNUs connected to the VNU v_n, with a connection degree $d_{v_n} = |\mathcal{N}(v_n)|$, and denote $\mathcal{N}(c_m)$ the set of VNUs connected to the CNU c_m, with a connection degree $d_{c_m} = |\mathcal{N}(c_m)|$. We will study in this paper only regular $d_v = 3$ and $d_v = 4$ LDPC codes for Nand Flash memory.

B. Gradient Descent Bit Flipping Concept

The GDBF algorithm has been introduced by Wadayama *et al.* in [5]. This algorithm is derived from a gradient descent formulation and it consists of finding the most suitable bits to be flipped in order to maximize a pre-defined objective function. The GDBF algorithm shows an error correction capability superior than most known BF algorithms while still maintaining a lower hardware complexity compared to the soft decision decoders. In [3], the authors proposed to incorporate a probabilistic feature in the flipping step, inspired from the probabilistic BF algorithms of [8]. In the PGDBF decoder, all the bits that satisfy the gradient descent condition are not flipped by default, but instead, only a randomly chosen fraction p of them are flipped. Curiously, this small modification of the GDBF algorithm led to a large performance improvement, with error correction capability approaching the soft-decision message passing decoders [3]. An efficient hardware (HW) implementation of the FM-PGDBF decoder is proposed in this paper, which minimizes the resource overhead needed to implement the random perturbations and replace the maximum finder of the PGDBF by a predefined set of values computed in off-line.

C. Probabilistic Gradient Descent Bit Flipping Decoder (PGDBF) and its adaptation to Flash memory

A BF decoder is defined as an iterative update of the variable node values over the decoding iterations. We denote in this paper by $v_n^{(k)}$ the value of the VN v_n at the k-th iteration. We correspondingly denote by $c_m^{(k)}$ the binary value of the CN c_m parity check node at iteration k, which indicates whether the fact that the m-th parity-check equation is satisfied or not. The BF decoding process is terminated when all CNs values are satisfied or a maximum number of iteration It_{max} is reached. The CN calculation in BF algorithms can be written as

$$c_m^{(k)} = \bigoplus_{v_n \in \mathcal{N}(c_m)} v_n^{(k)},$$

where $\bigoplus$ is the bit-wise exclusive-OR (XOR) operation.

For the n-th VN calculation, the update rule uses the information on satisfiability of the neighboring CN $\mathcal{N}(v_n)$ to keep or flip the value of $v_n^{(k)}$. In the case of GDBF algorithms, a function called *inversion function* or *energy function* is defined for each VN, and is used to evaluate whether the value $v_n^{(k)}$ should be flipped or not. The original GDBF has been proposed in communication for the Additive White Gaussian Nosie (AWGN) channel [5] and the energy function was defined as in (1), where γ_n is the Log-Likelihood Ratio (LLR) received from AWGN channel.

$$\Lambda_{v_n}^{(k)} = (1 - 2\, v_n^{(k)})\gamma_n + \sum_{c_m \in \mathcal{N}(v_n)} (1 - 2\, c_m^{(k)}) \qquad (1)$$

For the GDBF on the AWGN channel, the energy function is real valued, and has a unique minimum, corresponding to the bit with lowest reliability. In [5], two modes for the bit-flipping rule at iteration k are proposed: either only the bit having smallest energy function is flipped (single flip), or a group of bits having energy function lower than a predefined threshold are flipped (multiple flips).

For the Binary Symmetric Channel (BSC), the energy function can be modified with the following equation:

$$E_{v_n}^{(k)} = v_n^{(k)} \bigoplus y_n + \sum_{c_m \in \mathcal{N}(v_n)} c_m^{(k)} \qquad (2)$$

In this case, the energy function is an integer, varies from 0 to $(d_{v_n} + 1)$, and the bits which have the maximum value $E_{max}^{(k)} = \max_n E_{v_n}^{(k)}$ are flipped. Due to the integer representation of the energy function, many bits are most likely to have the maximum energy, leading to the multiple flips mode. Let us use an indicator variable to indicate the VNs which have the maximum energy at iteration k, *i.e.* $I_n^{(k)} = 1$ if $E_{v_n}^{(k)} = E_{max}^{(k)}$, and $I_n^{(k)} = 0$ otherwise. The fact that the number of bits to be flipped cannot be precisely controlled induces a negative impact to the convergence of the GDBF, as the analysis of [3] shows. To avoid this effect, the PGDBF has been proposed with the following modification: instead of flipping all the bits with maximum energy function value, only a random fraction of those bits are flipped. The random fraction is fixed to a pre-defined value $p_n^{(k)}$, which could be different for each VN and each iteration. In this work, we restrict our study to the case for which $p_n^{(k)}$ are constant for all iterations and all VNs, denoted $p_0 \in [0,1]$ hereafter.

The PGDBF of [3] can be implemented using a sequence of N random bits, generated following a Bernoulli distribution $\mathcal{B}$ with parameter p_0, with different realizations at each iteration. We denote the random generator (RG) sequence at the k-th iteration by $R^{(k)}$. In the GDBF algorithm, a VN $v_n^{(k)}$ at iteration k is flipped (is XOR-ed by '1') when its energy function is a maximum (the indicator variable is $I_n^{(k)} = 1$)

while in PGDBF, a VN $v_n^{(k)}$ is flipped *if and only if* the two conditions $I_n^{(k)} = 1$ and $R_n^{(k)} = 1$, are both satisfied. In the proposed FM-PGDBF a VN $v_n^{(k)}$ is flipped when its energy function is greater or equal to a predefined threshold (maximum of energy) computed offline based on several Monte Carlo Simulation results. We need to predict in offline the maximum of energy for each decoding iteration. However this number of iteration can be a great number (several hundreds) and storing the whole sequence of thresholds is memory consuming. We propose to shorten this sequence and store only a fixed number of thresholds. Depending on the construction of the code, the parameters of the decoder (d_{v_n} and d_{c_m}), this size l of the predefined sequence of thresholds is calibrated. Among iteration the decoder will use the sequence of predefined threshold in a circular way, *i.e* when it reaches the last value of the sequence it restarts from the beginning and take the first value as the threshold of the next iteration. The FM-PGDBF algorithm is presented in Algorithm 1.

Algorithm 1 FM-PGDBF algorithm

```
**Generate** $TH_n$, $n = 1, \ldots, l$, in offline.
**Initialization** $k = 0$, $v_n^{(0)} \leftarrow y_n, n = 1, \ldots, N$.
$s = H\mathbf{v}^{(0)^T}$ mod 2
**while** $s \neq 0$ *and* $k \leq It_{max}$ **do**
    **Generate** $R_n^{(k)}$, $n = 1, \ldots, N$, from $\mathcal{B}(p_0)$.
    **Compute** $E_{v_n}^{(k)}$, $n = 1, \ldots, N$, using Eq. (2).
    **for** $n = 1, \ldots, N$ **do**
        **if** $E_{v_n}^{(k)} >= TH_n$ **and** $R_n^{(k)} = 1$ **then**
            $v_n^{(k+1)} = v_n^{(k)} \oplus 1$
        **end if**
    **end for**
    $s = H\mathbf{v}^{(k+1)^T}$ mod 2
    $k = k + 1$
**end while**
**Output:** $\mathbf{v}^{(k)}$
```

III. Hardware Architectures

The state-of-the-art implementations of PGDBF can be found several works such as [6][7]. These implementations follow the flooding scheduling where N Variable Node Processing Units (VNUs) are implemented to process N VNs in parallel. Similarly, M d_c-XOR gates are implemented for M CNs (each XOR gate is called the check node processing unit - CNU). The maximal operating frequency of PGDBF is limited by the length of the longest data path (and so-called, the critical path) as $f_{max} < 1/t_D$ where t_D is the length of the critical path. The longest data path in PGDBF implementation is shown by the dashed green path in Fig. 1. This critical path starts from the output of D-FlipFlop storing $v_n^{(k)}$ to the updated VN value for the next iteration, ($v_n^{(k+1)}$). A noticeable part of this critical path comes from the global maximum finder (MF), which globally computes the maximum energy among N values. Many implementing methods of the MF can be found in [9]. We illustrate the implementation of MF in Fig. 1 by the binary tree of the compare-and-swap units (CASUs) [10]. The CASU passes at its output the larger value of the 2 input values. The latency of the MF module can be formulated as $t_{MF} = t_c\lceil log_2(N)\rceil$ where t_c is the delay of a single CASU module. This critical path may become longer

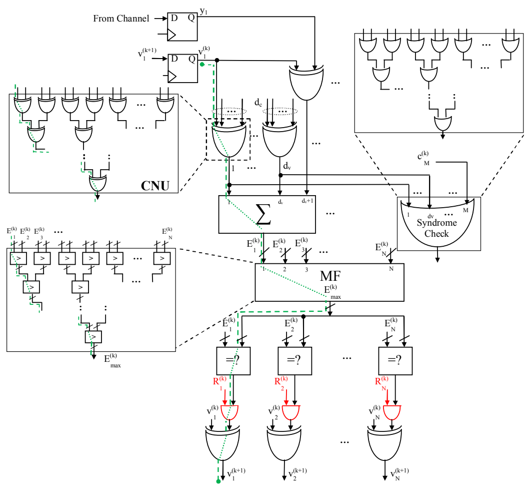

Figure 1. The implementation of PGDBF and its critical path.

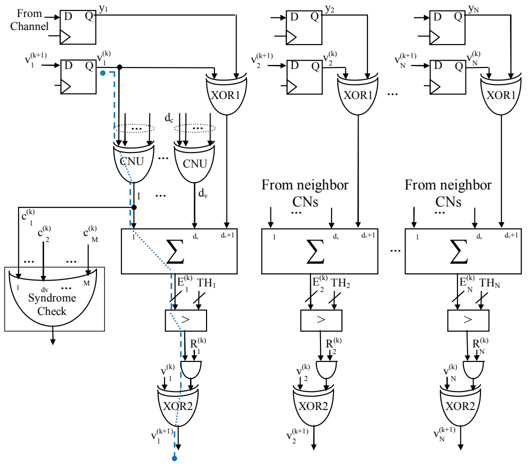

Figure 2. The proposed architecture for FM-PGDBF.

when longer code is used. This MF is also the large part of PGDBF in term of complexity since it requires approximately $\sum_{i=1}^{\lceil log_2(N)\rceil} \lceil N/2^i \rceil$ the CASUs.

The architecture of our FM-PGDBF, on contrary, can avoid the negative effect of the MF, which help enhance the decoding throughput and reduce decoder complexity. Indeed, as see in Fig. 2 for each VN, a comparing module is implemented in the place of the MF and equality comparator (Fig. 1). The critical path is significantly reduced since the latency part of MF is replaced by the one of the comparator which is obviously smaller. It can be also seen that, all the hardware cost for the MF can be avoidable in the MF-PGDBF implementation which is expected to reduce the decoder complexity.

IV. Synthesis results and decoding performance

The FM-PGDBF decoder implementations are implemented and synthesized by using the Xilinx tool ISE 14.7 with the target device Xilinx Virtex 6 (XC6VLX240T-1FF1156). The FPGA synthesis results (number of registers, number of Look-Up Tables) reported in this section have been obtained after place route process. We implement our LDPC decoders for various LDPC codes from a short to very long codeword

Table I. Hardware resource used to implement GDBF, PGDBF and FM-PGDBF decoders for different LDPC codes from short to very long codeword lengths and different code rate.

	Registers						
	dv= 3			dv= 4			
	R040N155	**R050N1296**	**R075N1296**	**R050N1296**	**R075N1296**	**R0857N2212**	**R0868N9520**
GDBF	329 (+0%)	2613 (+0%)	2614 (+0%)	2613 (+0%)	2614 (+0%)	4445 (+0%)	19063 (+0%)
PGDBF (S= M/2)	410 (+24.6%)	2971 (+13.7%)	2810 (+7.5%)	2971 (+13.7%)	2810 (+7.5%)	4637 (+4.3%)	19657 (+3.1%)
PGDBF (S= M/3)	394 (+19.8%)	2863 (+9.6%)	2756 (+5.4%)	2863 (+9.6%)	2756 (+5.4%)	4585 (+3.1%)	19471 (+2.1%)
FM-PGDBF	410 (+24.6%)	2971 (+13.7%)	2810 (+7.5%)	2971 (+13.7%)	2810 (+7.5%)	4637 (+4.3%)	19657 (+3.1%)
	Look-Up-Tables (LUTs)						
GDBF	1735 (+0%)	12331 (+0%)	13426 (+0%)	13228 (+0%)	12809 (+0%)	22045 (+0%)	94978 (+0%)
PGDBF (S= M/2)	1757 (+1.3%)	13650 (+10.7%)	14755 (+9.8%)	13973 (+5.6%)	13553 (+5.8%)	22326 (+1.3%)	95002 (+0%)
PGDBF (S= M/3)	1757 (+1.3%)	13650 (+10.7%)	14752 (+9.8%)	13976 (+5.7%)	13553 (+5.8%)	22322 (+1.3%)	95010 (+0%)
FM-PGDBF	702 (-59.5%)	5460 (-55.7%)	5902 (-56%)	5589 (-57.7%)	5421 (-57.7%)	8930 (-59.5%)	38000 (-60%)

lengths and different code rate, i.e. the smallest LDPC code is Tanner code with length $N = 155$ and rate 0.4 (denoted as R040N155) and the longest is LDPC code for flash memory with $N = 9520$ and rate 0.868(denoted as R0868N9520). We can compare the FM-PGDBF complexity with the non-probabilistic GDBF and PGDBF implementation architecture found in [7].

Note that, since the FPGA results are not available in that paper, we have obtained them by re-implementing these decoders and synthesizing them on FPGA. The synthesis results are presented in table I along with those of PGDBF and non-probabilistic GDBF which are shown as a baseline in comparison. All absolute values of registers and LUTs are presented followed by the their normalized values with respect to the GDBF.

The synthesis results show that, FM-PGDBF helps reduce significantly the complexity of PGDBF implementations as expected. Indeed, FM-PGDBF requires only 40% of the GDBF LUTs to be efficiently implemented. It is reasonable that the FM-PGDBF requires slightly more registers than GDBF i.e. $+25\%$ for low rate and $+3\%$ for high rate codes since it still needs the implementation of the random generator. This extra cost in term of registers is exactly the same as PGDBF decoder (s=M/2) since both are using the same probabilistic generator implementation. By reading vertically data column presented in table I, the hardware resources needed for all tested LDPC codes follow exactly our previous analysis. Interestingly, the high code rate and long codeword, the overhead LUTs required by PGDBF decoders are negligible i.e. for dv4R0868N9520, PGDBF requires 0.03% more LUTs while FM-PGDBF needs even -60%, for dv4R0857N2212, these numbers are around 1.3% for PGDBF decoders and -59% for FM-PGDBF. We compute the decoding throughput for dv3R050N1296 code as $\theta = \frac{N * f_{max}}{It_{ave} * n_c}$ where It_{ave} denotes the average number of iterations, f_{max} is the maximum decoding frequency, and n_c is the number of clock cycles required for a decoding iteration. $n_c = 1$ for all GDBF-based decoders while $n_c = 6$ for the MS. At $\alpha = 0.01$, FM-PGDBF obtains 11.2Gbps while is it 6480 Mbps for GDBF and 5355.37 Mbps for PGDBF (s= M/2). The PGDBF decoding throughput is, in general, inferior to the one of GDBF decoder. It is due to the fact that, the probabilistic behavior slows the decoding speed leading to higher average iteration numbers. However with our new proposed FM-PGDBF we overcome this problem and we even double the throughput of the decoder compared with PGDBF.

The simulated decoding performance of the decoders are shown in Fig. 3. FM-PGDBF provides an equivalent decoding performance of the PGDBF with slightly earlier error

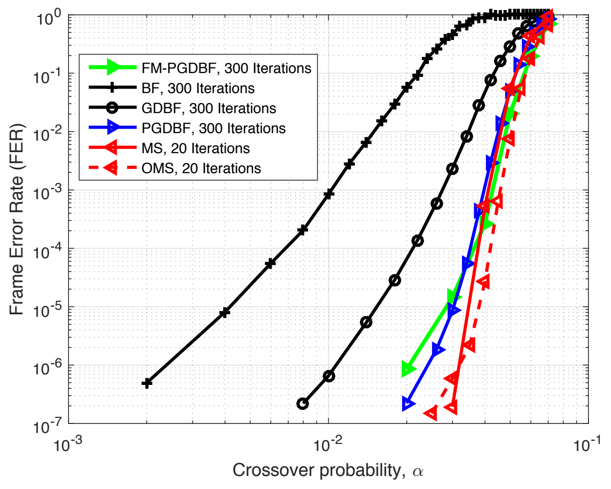

Figure 3. Comparison of decoders performance on a quasi-cyclic ($d_v = 4, d_c = 8$) LDPC code, with codeword length $N = 1296$.

floor. The simulation result re-confirms the superiority of FM-PGDBF over the GDBF with around 2dB gain in FER ($\alpha = 0.02$). We can notice here the interesting decoding performances of our proposed FM-PGDBF decoder in the waterfall region of the curve. Note that, operating at the $It_{max} = 300$ in BF decoders is only in the worst case. PGDBF decoder is advantageous in term of decoding throughput as seen in [7]. Also, the comparison on the number of average iteration can be found in that paper. OMS decoder is the best in term of decoding performance with the cost of high complexity and low decoding throughput.

V. Conclusion

We propose in this paper an efficient hardware architecture to implement PGDBF decoder on QC-LDPC codes, called Flash Memory adapted PGDBF. The Flash Memory adapted PGDBF uses a predefined list of maximum values for the energy function instead of computing it in run-time.

The simulation performance and FPGA synthesis results confirm that the Flash Memory adapted PGDBF decoders have lower complexity, higher throughput and good performance with respect to other implementations in literature.

Acknowledgment

This work was supported by the National Science Foundation under Grant ECCS-1500170 and CCF- 1314147, the Indo-US Science and Technology Forum (IUSSTF) through the Joint Networked Center for Data Storage Research (JC-16-2014-US), and the French ANR project NAND under grant agreement ANR-15CE25- 0006-01

References

[1] D. Declercq, M. Fossorier, and E. Biglieri, “Channel coding: Theory, algorithms, and applications,” *Academic Press Library in Mobile and Wireless Communications, Elsevier, ISBN: 978-0-12-396499-1*, 2014.

[2] R. G. Gallager, “Low density parity check codes,” *MIT Press, Cambridge, 1963, Research Monograph series*, 1963.

[3] O. A. Rasheed, P. Ivanis, and B. Vasić, “Fault-tolerant probabilistic gradient-descent bit flipping decoder,” *IEEE Communications Letters*, vol. 18, no. 9, pp. 1487–1490, Sept 2014.

[4] G. Sundararajan, C. Winstead, and E. Boutillon, “Noisy gradient descent bit-flip decoding for ldpc codes,” *IEEE Transactions on Communications*, vol. 62, no. 10, pp. 3385–3400, Oct 2014.

[5] 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.

[6] K. Le, D. Declercq, F. Ghaffari, C. Spagnol, E. Popovici, P. Ivanis, and B. Vasíc, “Efficient realization of probabilistic gradient descent bit flipping decoders,” in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2015, pp. 1494–1497.

[7] K. Le, F. Ghaffari, D. Declercq, and B. Vasić, “Efficient hardware implementation of probabilistic gradient descent bit-flipping,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 4, pp. 906–917, April 2017.

[8] N. Miladinovic and M. Fossorier, “A improved bit-flipping decoding of low-density parity-check codes,” *IEEE Transactions on Information Theory*, vol. 51, no. 4, pp. 1594–1606, Apr. 2005, cCF-0963726 (Medium Decoders), CCF-1314147 (TDMR), ECCS-1500170 (FSO), i-RISC, Fulbright.

[9] B. Yuce, H. F. Ugurdag, S. Goren, and G. Dundar, “Fast and efficient circuit topologies forfinding the maximum of n k-bit numbers,” *IEEE Transactions on Computers*, vol. 63, no. 8, pp. 1868–1881, Aug 2014.

[10] J. Jung and I. C. Park, “Multi-bit flipping decoding of ldpc codes for nand storage systems,” *IEEE Communications Letters*, vol. PP, no. 99, pp. 1–1, 2017.