Efficient Hardware Implementation of Probabilistic Gradient Descent Bit-Flipping

Khoa Le, *Student Member, IEEE*, Fakhreddine Ghaffari, *Member, IEEE*, David Declercq, *Senior Member, IEEE*, and Bane Vasić, *Fellow, IEEE*

***Abstract*— This paper deals with the hardware implementation of the recently introduced Probabilistic Gradient-Descent Bit-Flipping (PGDBF) decoder. The PGDBF is a new type of hard-decision decoder for Low-Density Parity-Check (LDPC) code, with improved error correction performance thanks to the introduction of deliberate random perturbation in the computing units. In the PGDBF, the random perturbation operates during the bit-flipping step, with the objective to avoid the attraction of so-called trapping-sets of the LDPC code. In this paper, we propose an efficient hardware architecture which minimizes the resource overhead needed to implement the random perturbations of the PGDBF. Our architecture is based on the use of a Short Random Sequence (SRS) that is duplicated to fully apply the PGDBF decoding rules, and on an optimization of the maximum finder unit. The generation of good SRS is crucial to maintain the outstanding decoding performance of PGDBF, and we propose two different methods with equivalent hardware overheads, but with different behaviors on different LDPC codes. Our designs show that the improved PGDBF performance gains can be obtained with a very small additional complexity, therefore providing a competitive hard-decision LDPC decoding solution for current standards.**

***Index Terms*— Gradient descent bit-flipping, high throughput decoder, low complexity implementation, low-density parity-check codes, random generation.**

I. Introduction

LOW-DENSITY parity-check (LDPC) codes have attracted much attention in the past several years due to their excellent performance under iterative decoding. These studies focus either on improving the error correction performance of the LDPC codes or on reducing the implementation complexity of the decoders for practical applications. Soft decision iterative decoding, such as Belief Propagation (BP) or Min-Sum (MS) algorithms offers the best error correction performance, close to the theoretical coding bounds, but comes along with an intensive computation cost [1]. On the contrary, the class of hard-decision iterative decoders is often seen as a very low-complexity solution, with an associated performance loss. Of special interest is the class of A Posteriori Probability (APP) based hard decision decoders, such as Bit-Flipping (BF) or majority-logic decoders where, contrary to message-passing decoders, both the extrinsic and the intrinsic information are exchanged between the nodes of the Tanner graph [2], [3].

The BF algorithms significantly reduce the required hardware resources due to their simple computation units, but lead to a non-negligible performance loss compared to the BP or MS algorithms. In order to improve the performance while keeping the low complexity, many modifications of BF decoders have been introduced, such as Weighted BF (WBF) [4], improved WBF [2] and Gradient Descent Bit Flipping (GDBF) [5]. All the BF decoders share the same concept of passing only one bit of information between the Variable Nodes (VNs) and Check Nodes (CNs) of the LDPC Tanner graph. The difference between BF variants lies on the mechanism proposed to select the bits to be flipped, based on the computation of the so-called *energy function* or *inversion function*. Another difference between BF variants lies in the number of bits that are flipped during one decoding iteration: the serial BF decoder flips only one bit at a time, while the parallel or semi-parallel BF flips many bits at the same time. For weighted BF decoders [4], [6], [7], the energy function is specifically designed for the Additive White Gaussian Noise (AWGN) channel through a combination of the CN values and the channel-output values. All these BF variants achieve different levels of error correction and different convergence speeds, impacting on the performance-throughput trade-off of the LDPC decoder.

More recently, the GDBF algorithm has been introduced by Wadayama *et al.*. in [5]. This GDBF algorithm is derived from a gradient descent formulation and its principle consists of finding the most suitable bits to be flipped in order to maximize a pre-defined objective function. The GDBF algorithm showed an error correction capability superior to most known BF algorithms while still keeping the hardware implementation simplicity. A very promising generalization of the GDBF has been proposed by Rasheed *et al.* in [8]. The authors proposed to incorporate a probabilistic feature in the flipping step, inspired from the probabilistic BF algorithms of [2]. In the PGDBF decoder, the bits that satisfy the gradient condition

Manuscript received August 11, 2016; revised October 28, 2016; accepted November 14, 2016. Date of publication December 16, 2016; date of current version March 27, 2017. This work was funded in parts by the Seventh Framework Programme of the European Union, under Grant Agreement number 309129 (i-RISC project), the Franco-Romanian (ANR-UEFISCDI) Joint Research Program (DIAMOND project), and the NSF under grants CCF-0963726, CCF- 1314147 and ECCS-1500170. Bane Vasić acknowledges the support of the Fulbright Scholar Program. The authors would like to thank Truong Nguyen-Ly (thientruong.nguyen-ly@cea.fr) for providing the ASIC synthesis results. This paper was recommended by Associate Editor F. J. Kurdahi.

K. Le, F. Ghaffari, and D. Declercq are with the ETIS, ENSEA/University of Cergy-Pontoise/CNRS, Cergy-Pontoise 95014, France (e-mail: khoa.letrung,fakhreddine.ghaffari,declercq@ensea.fr).

B. Vasić is with the Department of Electrical and Computer Engineering, University of Arizona, Tucson, AZ 85719 USA (e-mail: vasic@ece.arizona.edu).

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.2016.2633581

are not flipped by default, but instead, only a randomly chosen fraction p_0 of them are flipped. Interestingly, this small modification of the GDBF algorithm led to a large performance improvement, with error correction capability approaching the soft-decision message passing decoders [8].

We propose in this paper an efficient hardware (HW) implementation of the PGDBF decoder, which minimizes the resource overhead needed to implement the random perturbations of the PGDBF. We restrict our work to decoders for regular LDPC codes used over the binary symmetric channel (BSC). Although many wireless communication standards use irregular LDPC codes, regular LDPC codes could still be interesting for practical applications, such as fiber optics transmissions with the IEEE 802.3an 10GBASE-T standard [9], satellite communications using short frames [10] or storage applications (hard drives, flash memories, etc) [11]–[13].

The paper is organized as follows. In Section II, we present the main principles of the PGDBF algorithm and conduct a statistical analysis in order to have a precise characterization of its key parameters, especially the values of p_0 that lead to the maximum coding gains. This analysis is performed through Monte Carlo simulations in both the waterfall and the error floor regions. In the second part of the paper, Section III, we present our optimized HW architecture for the PGDBF decoder. Our architecture is based on the use of a short random signal sequence that is duplicated to fully apply the PGDBF decoding rules on the whole codeword. We propose two different solutions with equivalent HW overheads, but with different behaviors on different LDPC codes. We also propose an optimization of the maximum finder unit of the PGDBF algorithm in order to reduce the critical path and improve the decoding throughput. Finally, in Section IV, we provide synthesis results on ASIC 65 nm technology, and run Monte-Carlo simulations with a bit-accurate C implementation of our PGDBF architecture on LDPC codes with various rates and lengths.

II. Description and Analysis of the PGDBF

A. Notations and PGDBF Algorithm

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 BSC, in which the bits of the transmitted codeword $\mathbf{x}$ have been flipped with crossover probability α. The decoders presented in this paper are dedicated to the BSC channel. The graphical representation of an LDPC code is a bipartite graph called Tanner graph composed of two types of nodes, the VNs v_n, $n = 1, \ldots, N$ and the CNs c_m, $m = 1, \ldots, M$. In the Tanner graph, a VN v_n is connected to a CN c_m if $H(m,n) = 1$. Let us also denote $\mathcal{N}(v_n)$ the set of CNs connected to the VN v_n, with a connection degree $d_{v_n} = |\mathcal{N}(v_n)|$, and denote $\mathcal{N}(c_m)$ the set of VNs connected to the CN c_m, with a connection degree $d_{c_m} = |\mathcal{N}(c_m)|$. When an LDPC code is regular, its connection degrees are equal for all nodes, i.e., $d_{c_m} = d_c, \forall m$ and $d_{v_n} = d_v, \forall n$. In this paper, we will mainly study the case of regular Quasi-Cyclic LDPC (QC-LDPC) codes. A QC-LDPC code is obtained by a specific construction in which a small *base matrix* H_B is expanded by replacing each vertex in H_B by a circulant permutation of main diagonal $Z \times Z$ matrix, in order to obtain the actual LDPC matrix H. QC-LDPC codes are often preferred in practical applications because of their hardware friendly structure, and have been adopted by several standards [14] [15]. Although applicable to irregular LDPC codes, the bit-flipping decoders are specially attractive and efficient for regular LDPC codes, and we will restrict the study in this paper to regular $d_v = 3$ and $d_v = 4$ LDPC codes.

A BF decoder is defined as an iterative update of the variable node values over the decoding iterations. We denote 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 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

$$c_m^{(k)} = \bigoplus_{v_n \in \mathcal{N}(c_m)} v_n^{(k)} \tag{1}$$

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 for the AWGN channel [5] and the energy function was defined as in (2), where γ_n is the Log-Likelihood Ratio (LLR) received from AWGN channel.

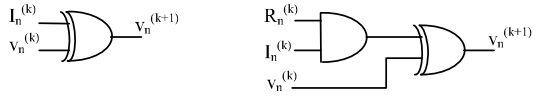
$$\Lambda_{v_n}^{(k)} = (1 - 2\, v_n^{(k)})\gamma_n + \sum_{c_m \in \mathcal{N}(v_n)} (1 - 2\, c_m^{(k)}) \tag{2}$$

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 a single 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 BSC channel, the energy function can be modified with the following equation:

$$E_{v_n}^{(k)} = v_n^{(k)} \bigoplus y_n + \sum_{c_m \in \mathcal{N}(v_n)} c_m^{(k)} \tag{3}$$

In this case, the energy function is integer-valued, 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 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 [8] shows. To avoid this effect, the authors



GDBF:
$I_n^{(k)}= 1$ if $E_n^{(k)}= E_{max}^{(k)}$ else $I_n^{(k)}= 0$

PGDBF:
$I_n^{(k)} = 1$ if $E_n^{(k)}= E_{max}^{(k)}$ else $I_n^{(k)}= 0$.
$R_n^{(k)}= 1$ with probability of p_0

Fig. 1. The difference in the flipping operator between GDBF and PGDBF algorithms.

Algorithm 1 Probabilistic Gradient Descent Bit-Flipping

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 (3).
$E_{max}^{(k)} = max_n(E_{v_n}^{(k)})$,
for $n = 1, \ldots, N$ **do**
if $E_{v_n}^{(k)} = E_{max}^{(k)}$ **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: $\mathbf{v}^{(k)}$

in [8] proposed the PGDBF algorithm, in which 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 ourself to the case for which $p_n^{(k)}$ are constant for all iterations and all VNs, denoted $p_0 \in [0, 1]$ hereafter.

The random signals in PGDBF 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 signal (RS) sequence at the k-th iteration by $R^{(k)} = \{R_n^{(k)} | 1 \leq n \leq N\}$ in which the random signal $R_n^{(k)}$ is triggered correspondingly to VN n-th. The difference in flipping decision between GDBF and PGDBF is described in Fig. 1. 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. The PGDBF algorithm is presented in Algorithm 1.

B. Statistical Analysis of PGDBF

Several recent papers showed that the decoding performance of PGDBF is superior to all known BF algorithms [8], [16] as illustrated in Fig. 2 for a regular $(d_v, d_c) = (4, 8)$ QC-LDPC code of length $N = 1296$ bits. As we can see on this figure, the PGDBF performs halfway between the GDBF and the Min-Sum decoder, which is promising in terms of error correction,

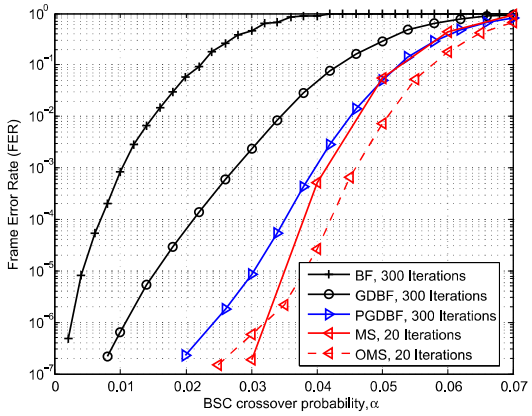


Fig. 2. Performance Comparison between LDPC decoders: BF, GDBF, PGDBF ($p_0 = 0.9$), Quantized MS, Quantized Offset Min-Sum (OMS) with offset factor of 1. Regular QC-LDPC code ($d_v = 4, d_c = 8, Z = 54$), ($N = 1296, M = 648$).

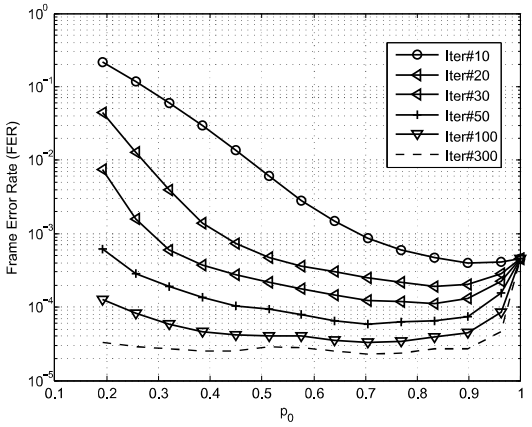


Fig. 3. Frame Error Rate versus p_0 in the waterfall region ($\alpha = 0.01$). Tanner code ($d_v = 3, d_c = 5, Z = 31$), ($N = 155, M = 93$).

provided that the extra hardware complexity to implement the generation of the random sequences $R^{(k)}$ is small enough.

In order to better understand the impact of the randomness of PGDBF on the Frame Error Rate (FER), we have conducted a statistical analysis of PGDBF using Monte Carlo simulations. Our objective is to identify which features of the probability density function of the binary random sequence $R^{(k)}$ are the most critical for the performance improvements.

1) Waterfall Analysis: In this section, we focus on the effect of the relative occurrence of zeros and ones in the sequence $R^{(k)}$ in the waterfall region of the decoder. The simulations are performed on the BSC channel with cross-over probability α. For each noisy codeword, a fraction of $p_0 N$ ones are put in the random sequence $R^{(k)}$, and we draw the FER of the PGDBF decoder as a function of p_0, for different iteration and a value of α corresponding to the waterfall region. The results in Fig. 3 are presented for the $(N = 155, M = 93)$ Tanner code [17] which is a regular QC-LDPC code with $(d_v = 3, d_c = 5, Z = 31)$.

Based on these results, two interesting conclusions can be drawn. First, during the first decoding iterations ($k \leq 10$), using randomness does not help. On the contrary, it degrades

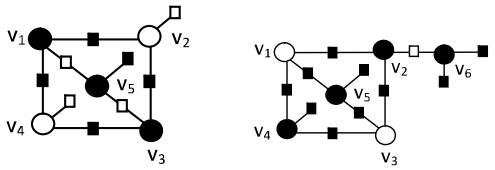


Fig. 4. Error configurations with (a) 3 erroneous bits and (b) 4 erroneous bits located on a TS(5, 3). Black/white circles denote erroneous/correct variable nodes, and black/white squares denote unsatisfied/satisfied check nodes.

the decoding performance since the FER of PGDBF is worse than that of GDBF for almost all values of p_0 (note that GDBF corresponds to PGDBF for $p_0 = 1$). This comes from the fact that the random part of the PGDBF slows down the convergence speed, since fewer bits are flipped than what the energy function indicates. Second, after a sufficient number of iterations, the performance gain is substantial and does not depend much on p_0. This means in particular that optimizing RS sequence probability p_0 does not impact significantly the performance gain, as was already observed in [8].

We have confirmed these conclusions for several regular LDPC codes with different lengths and values of d_v.

2) Error Floor Analysis: Similar to other iterative LDPC decoders, the GDBF algorithm fails to correct some low-weight error patterns concentrated on trapping sets (TS) [1], [18], giving rise to the so-called error floor region. The PGDBF has been introduced to overcome the attraction of TS. In this section, we conduct the same experiment as in Section II-B.1 for the Tanner code ($d_v = 3, d_c = 5, Z = 31$), ($N = 155, M = 93$), but in the error floor region. The smallest trapping set for this code is composed of 5 VNs and 3 odd-degree CNs, denoted TS(5, 3) (see Fig. 4).

The minimum number of bits that cannot be corrected by the deterministic GDBF is three [19], and they are located in the TS(5, 3) of the LDPC code as indicated in Fig. 4(a) with black circles. Note that (v_1, v_2, v_3) and (v_1, v_4, v_3) are also weight-3 error patterns which cannot be corrected by the GDBF. The PGDBF can potentially help correct these low weight error patterns, resulting in a coding gain in the error floor region. Fig. 4(b) shows also a weight-4 error pattern which is uncorrectable by the GDBF. In order to analyze the PGDBF in the error floor, we fix the channel errors on the positions indicated in Fig. 4, and evaluate whether a random sequence $R^{(k)}$ with probability p_0 can correct these error patterns. The results are shown in Figs. 5 and 6.

The first remark that can be made is that, contrary to the conclusion of the waterfall analysis, it is useful to use the random feature of the PGDBF, even during the first decoding iterations. This is verified for both the weight-3 error and the weight-4 error patterns. The weight-3 error pattern is eventually corrected when the number of iterations increases, for all values of $p_0 \in [0.3, 0.9]$. The weight-4 error patterns can also be corrected by the PGDBF for a wide range of p_0 values, but flattens at a FER equal to 0.5. This means that half of the generated random sequences $R^{(k)}$ can correct the weight-4 error pattern, while the other half does not

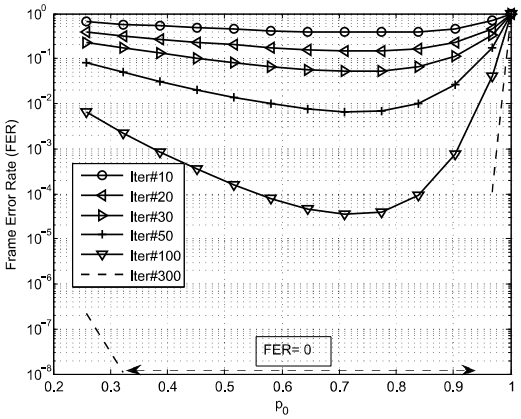


Fig. 5. Frame Error Rate versus p_0 in the error floor region with 3 erroneous bits. Tanner code ($d_v = 3, d_c = 5, Z = 31$), ($N = 155, M = 93$).

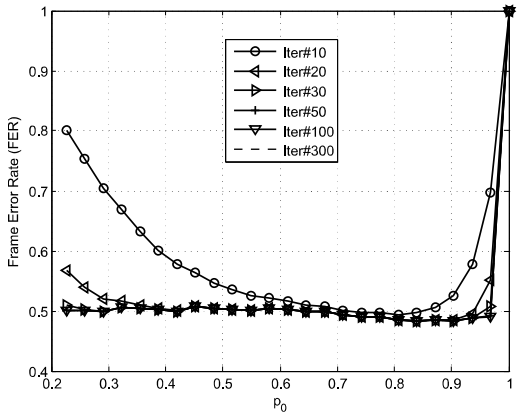


Fig. 6. Frame Error Rate versus the p_0 in the error floor region with 4 erroneous bits. Tanner code ($d_v = 3, d_c = 5, Z = 31$), ($N = 155, M = 93$).

improve the performance. PGDBF performance gains depend on the realizations of $R^{(k)}$, and in some situations requires the concept of PGDBF decoder rewinding [19], that is to start again the decoder from the initial values, but with different random sequences $R^{(k)}$. For the example of the weight-4 error patterns of Fig. 4(b), a PGDBF with k rewinding stages would correct the error patterns with probability $1 - (0.5)^k$. Since our work deals with low complexity implementation of the PGDBF decoder and because the PGDBF gain is already significant without rewinding, we will not discuss decoder rewinding in this paper.

As a conclusion of this section, our statistical analysis reveals that the random generator does not need to have a specific value for p_0 in order to provide the performance gains of the PGDBF, as long as it is bounded away from $p_0 = 1$. This motivated the proposition of a simplified, low complexity hardware realization for the generation of sequence $R^{(k)}$, which we describe in the next section.

III. Optimized Hardware Implementation

The random feature of PGDBF plays an important role in improving the decoder performance as presented in the previous section. However, a hardware overhead is unavoidable due to the fact that a binary random generator is required on top

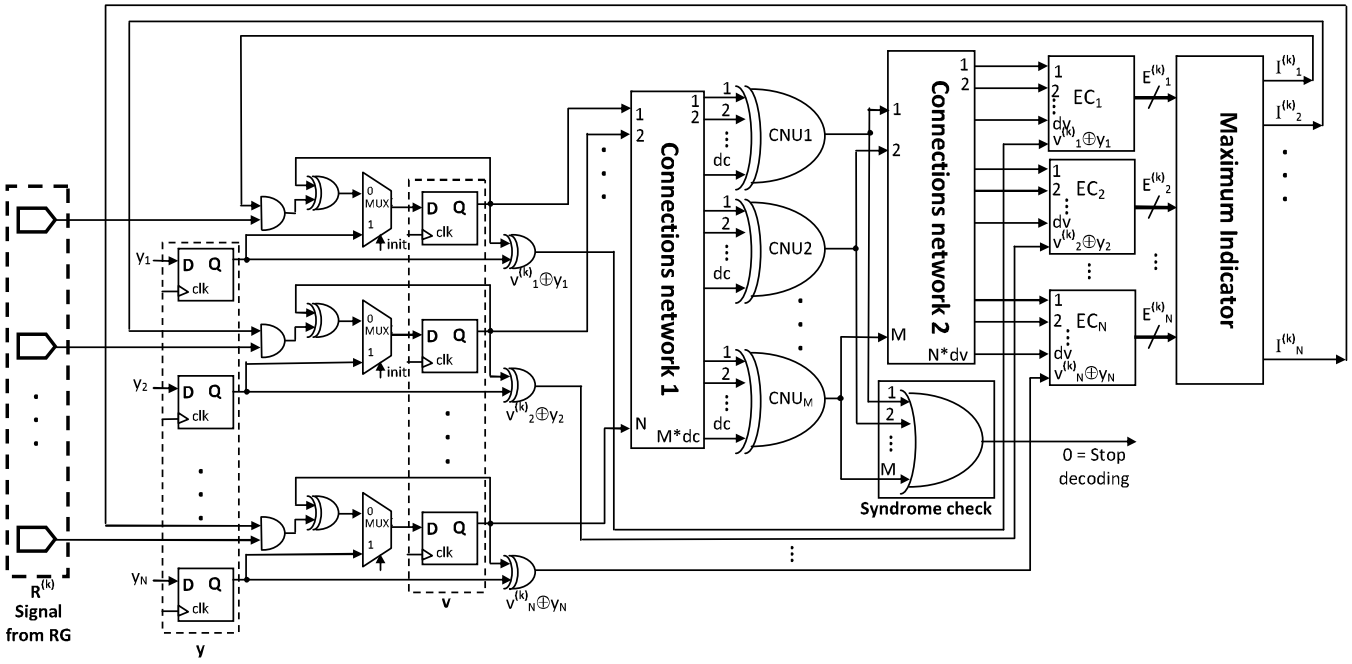


Fig. 7. The global architecture of the PGDBF. The PGDBF follow precisely the data flow of GDBF with difference coming from the random generator and the AND-gates.

of the original GDBF structure. We present in this section an optimized hardware implementation of the PGDBF, with the objective of keeping the coding gain at a minimum hardware cost.

A. PGDBF Architecture

The global architecture of the PGDBF decoder is shown in Fig. 7. The organization of the blocks and the data flow follow precisely the organization of the GDBF decoder, with the difference being in the additional block which produces the random signals $R^{(k)}$.

Let us first briefly present how the decoder described in Algorithm 1 operates on this hardware architecture. Since we focus in this paper on the BSC channel, two register arrays represented as two sequences of D-Flip Flops (D-FFs) are required to store the noisy codeword $\mathbf{y}$ and the estimated codeword at the current iteration $\mathbf{v}^{(k)}$. At the initialization of the decoder, the signal *init* triggers the copy of $\mathbf{y}$ into $\mathbf{v}^{(0)}$. Then, the CNUs compute the parity of their neighboring bits in $\mathbf{v}^{(k)}$, after properly driven by a first connection network. The second connection network drives the CN values to the energy computation blocks, for each VN. The maximum indicator module is composed of a maximum finder component and comparators which outputs $I_n^{(k)} = 1$ whenever the corresponding energy is equal to the maximum, and $I_n^{(k)} = 0$ otherwise. Indicator values $I_n^{(k)}$ are propagated to the AND gates, and combined with the RS sequence. This series of AND gates highlights the difference between PGDBF and GDBF. In the GDBF decoder, all bits with $I_n^{(k)} = 1$ are flipped, while in the PGDBF algorithm, only the bits with $I_n^{(k)} = 1$ and $R_n^{(k)} = 1$ are flipped. New values of the bits stored in $\mathbf{v}^{(k)}$ are used for the next decoding iteration.

At each iteration, the syndrome check module performs an OR operation on the CNs values to verify whether the intermediate sequence $\mathbf{v}^{(k)}$ is a codeword, in which case the decoding process is halted. Another instance when the decoding halted is when no codeword $\mathbf{v}^{(k)}$ has been found, and a predetermined maximum number of iterations It_{max} has been reached, in which case a decoding failure is declared. Note that all components in the decoder architecture are combinational circuits except the registers $\mathbf{v}^{(k)}$ and $\mathbf{y}$. Therefore, new values of the bits in $\mathbf{v}^{(k)}$ are updated after each clock cycle. In order to optimize the proposed architecture, we focus on the following two important issues. First, as identified in [16], the hardware overhead induced by the RS is not negligible with a naive implementation, and we propose in Section III-B different low complexity methods to generate the sequences $R^{(k)}$. Second, we optimize the architecture of the maximum indicator module in order to maximize the decoding throughput, as explained in Section III-D.

B. Cyclically Shifted Truncated Random Sequences

We showed in Section II-B that the performance gain in the PGDBF algorithm comes from the introduction of the random sequence $R^{(k)}$ which makes a perturbation in bit flips. In our theoretical description of the PGDBF, for each and every codeword, the sequences at different iterations $(R^{(0)}, \ldots, R^{(It_{max})})$ are independent and identically distributed. However, a direct and naive generation of the sequences $R^{(k)}$ with linear feedback shift register (LFSR) random generators is costly, and requires many times more registers than the non-probabilistic GDBF [16]. We propose in this section an approach to reduce the hardware overhead required to generate the RS sequences.

The first modification that we propose is to reduce the register requirements by storing only $S \leq N$ random values

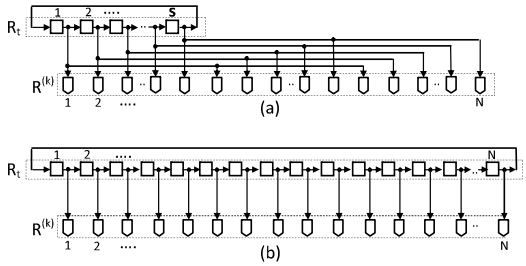


Fig. 8. Generation of the random signals. (a) corresponds to the use of truncated sequences, and (b) to the use of full sequences.

in a shorter sequence denoted as $R_t^{(k)}$, and produce the RS sequence $R^{(k)}$ with a hard-wired duplication network. The duplication network can be restricted to a simple *copy and concatenate*, as illustrated by the example in Fig. 8(a), or can be implemented as a more complex connection network, allocating S register outputs to N signals. When $S = N$, the duplication network is a simple copy of the register outputs, i.e., $R^{(k)} = R_t^{(k)}$ as shown in Fig. 8(b).

Although the length S of the truncated sequence can take in principle any value, in this paper we restrict the possible values of S to integer multiples of the QC-LDPC circulant size Z. We also enforce the duplication network to copy the register outputs to distinct circulant blocks, such that two copies of the same random bit do not belong to the same circulant block. The duplication network comes at very little cost in hardware implementation, and is simpler than the connection networks of the GDBF decoder.

The second modification that we propose is to re-use the same sequence of random bits for all the decoding iterations, instead of generating a new random sequence at each iteration. However, in order to preserve the performance gains of the PGDBF, the sequences $R_t^{(k)}$ need to be distinct from one iteration to another. Our solution is to simply rotate cyclically the sequence by one position, as shown in Fig. 8, such that:

$$R_t^{(k+1)}\left((n+1)\ mod\ S\right) = R_t^{(k)}(n) \qquad n = 1, \ldots, S \tag{4}$$

With these two modifications, our objective is to reduce to the maximum the hardware overhead induced by the RS sequence generation, while maintaining the necessary randomness of the PGDBF algorithm. Our proposed solution, named Cyclically Shifted Truncated Sequences (CSTS), has also the following two advantages with respect to the desired statistical properties of the random sequence. First, the initialization step starts with a random truncated sequence $R_t^{(0)}$ having a given number $n_0 = p_0 S$ of 1's, the number of 1's in the complete sequence $R^{(0)}$ will be in the range $\{\lfloor N/S \rfloor n_0, \ldots, (\lfloor N/S \rfloor + 1)\, n_0\}$. This means that the value of p_0 fixed by the statistical analysis in Section II-B would be the same for $R_t^{(0)}$ and $R^{(0)}$. Second, since we just cyclically shift the sequences $R_t^{(k)}$ from one iteration to another, the number of 1's is kept constant throughout the decoding iterations if N/S is an integer, and almost constant if N/S is not an integer.

One of the drawbacks of our reduced complexity method for generating the random signals is the induced correlation. In the theoretical PGDBF, the assumption is that the sequences $R^{(k)}$ should be independent from each others. However, it is obvious that by using the duplication from $R_t^{(k)}$ to $R^{(k)}$, and by shifting the sequences from one iteration to another, the independence assumption is no longer valid. It is also clear that the sequences will be more correlated as S gets smaller. However, we have verified through Monte Carlo simulations that the induced correlation has actually very little impact on the decoder performance. In Fig. 9, we show the FER performance of the PGDBF with CSTS, and compare it with the perfect PGDBF performance. The simulations are performed on a rate R = 1/2 regular ($d_v = 4, d_c = 8$) QC-LDPC code, with circulant size $Z = 54$ and length $N = 1296$ and the BSC channel. The labels LFSR-CSTS and IVRG-CSTS correspond to the two versions of the CSTS decoders that we propose in this paper, described in details in the next section. As we can see from these results, the use of CSTS does not degrade the error correction performance, even for small values of S. This behavior has been also observed for other code rates and lengths.

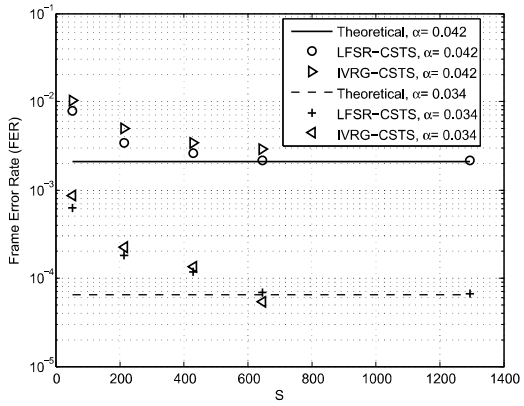


Fig. 9. Decoding performance of CSTS-PGDBF as a function of the size S of $R_t^{(0)}$.

C. Initialization of Truncated Sequence for CSTS-PGDBF Decoders

In the previous section, we showed that once the initial truncated sequence $R_t^{(0)}$ has been generated, our simplified CSTS architecture can build efficiently the RS sequences $R^{(k)}$, $k = 0, \ldots, It_{max}$, with no performance loss. In this section we describe two solutions to initialize $R_t^{(0)}$ with S random binary values, which will be compared in terms of hardware resources in Section IV. The proposed solutions are based on two random generating methods, namely linear feedback shift register (LFSR) and intrinsic valued random generator (IVRG). For simplicity, we drop the CSTS notation in the algorithm names and will refer to them as LFSR-PGDBF and IVRG-PGDBF.

1) Initialization With Linear Feedback Shift Register: A conventional method to produce pseudo-random bits is by making use of the LFSR. The outputs of the LFSR registers define an integer number, which is compared to a pre-determined threshold in order to finally produce a bit. The value of the threshold is tuned so that the appearance

probability of 1's at the comparator output is equal to the desired value, i.e., p_0 in our case. For large enough LFSR memory η, the correlation of the output bit sequence could be made negligible [20]. We use an $\eta = 32$-LFSR pseudo-random generator, to initialize the initial truncated sequence $R_t^{(0)}$ for each noisy codeword. Note that the latency to produce the S random values can be neglected since the LFSR can produce those values while the previous noisy codeword is decoded.

One drawback of this approach corresponds to the event that $R_t^{(0)}$ contains only binary ones, in which case all sequences $R^{(k)}$ are also filled with binary ones, thus reducing the PGDBF to deterministic GDBF. The probability of having $R_t^{(0)} = \mathbf{1}$ is equal to $(p_0)^S$, which can be considered negligible only if it is smaller than the target FER in the error-floor region. For example, with $p_0 = 0.75$ and $S = 64$, $p(R_t^{(0)} = \mathbf{1}) = 1e-8$. The LFSR method therefore requires an extra control module that tests if $R_t^{(0)} \neq \mathbf{0}$, or by forbidding too large values of p_0 and too small values of S.

2) Initialization With the Intrinsic-Valued Random Generator: In [16], we have introduced a novel and specific way for generating a sequence of random bits in an LDPC decoder without relying on an actual random generator. Our Intrinsic-Valued Random Generator (IVRG) approach was based on the use of the CN values which are already computed at the output of the CNU (see Fig. 7). The CNU blocks compute the parity-check node values, and if the estimated codeword contains errors, they typically contain a fraction of ones. We make use of the CN values at the first iteration $k = 0$, which depend on the noisy codeword generated by a random realization of the BSC channel. As a result, the sequence of CN values will appear as a random-like sequence that can be used to initialize $R_t^{(0)}$.

To simplify the discussion, let us consider the special case of $S = M$, the number of parity-check equations in the LDPC code, such that $R_t^{(0)}$ can be filled directly with the complemented outputs of the CNUs at the first iteration, as described in Fig. 10. With the IVRG approach, we need to complement the CN values since the number of unsatisfied check nodes is usually smaller than the number of satisfied CNs, and we saw in the statistical analysis of Section II-B that the interesting range of p_0 is greater than 0.5.

Also, with the IVRG approach, the number of ones in $R_t^{(0)}$ cannot be controlled and depends on the channel noise realization. Let us denote by $p(c^{(0)} = 1)$ (respectively $p(c^{(0)} = 0)$), the probability that a CN c is unsatisfied (respectively satisfied), at the initialization $k = 0$. Because of the complement after the CN computation, the probability of the binary ones in the RS sequence in the IVRG approach is equal to $p_0 = p(c^{(0)} = 0)$. For a BSC with crossover probability α, we can approximate this probability by $p_0 = p(c^{(0)} = 0) \simeq \frac{1}{2} + \frac{1}{2}(1-2\alpha)^{d_c}$. For large values of α, as in the waterfall region of the LDPC decoder, p_0 converges to $1/2$, and for small values of α, in the error-floor region, it converges to $p_0 = 1$, which corresponds to the GDBF.

Contrary to the LFSR method, the case where $R_t^{(0)} = \mathbf{1}$ is less problematic since it happens only when the decision at the channel output is already a codeword, and then no decoding

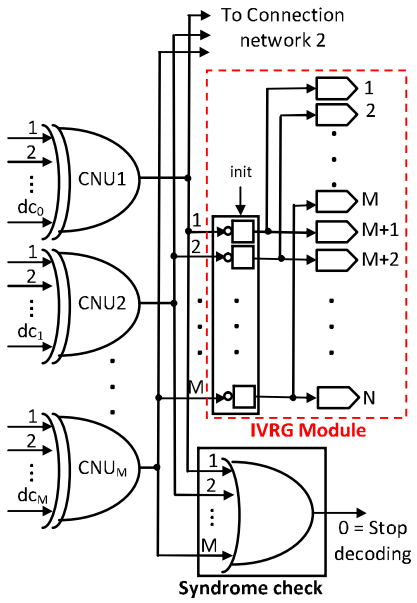


Fig. 10. A block diagram of the Intrinsic-Valued Random Generator module for $S = M = N/2$. The CNs values are copied into the $R_t^{(0)}$ at the first iteration, then cyclically shifted at each iteration.

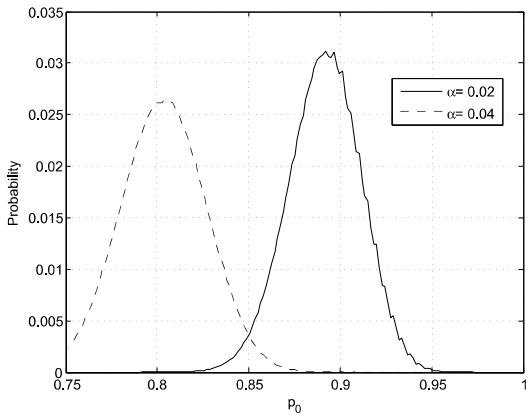


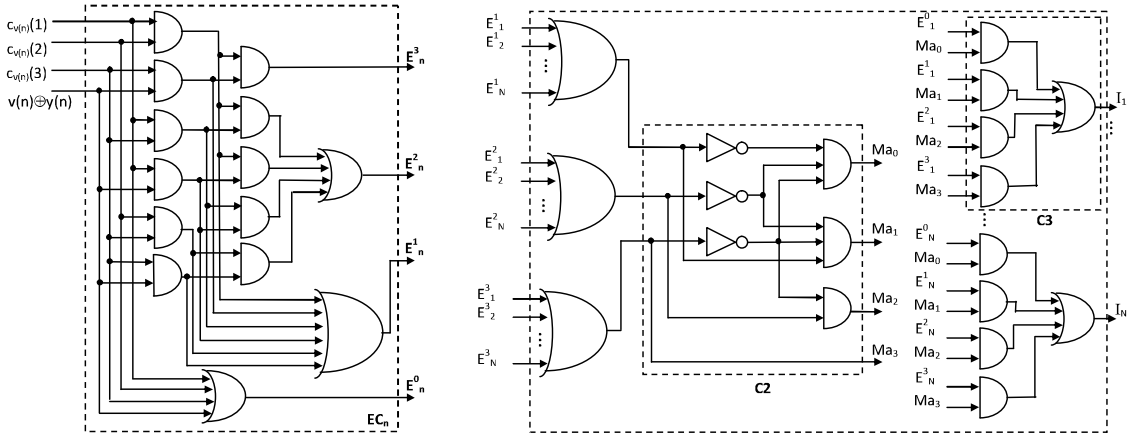
Fig. 11. The distribution of p_0 for the IVRG-PGDBF and a $(d_v = 3, d_c = 6, N = 1296)$ QC-LDPC code, for $\alpha = 0.02$ and $\alpha = 0.04$.

iteration is required. Fig. 11 shows the distribution of p_0 for a $(d_v = 3, d_c = 6, N = 1296)$ QC-LDPC code and two different channel error probabilities α. It can seen that p_0, although not fixed to a constant value with the IVRG, falls in the range of interest, predicted by the statistical analysis of Section II-B.

Another feature of the IVRG initialization is that copying the CN values in the $R_t^{(0)}$ registers can be done on-the-fly, and does not induce any extra latency. Note finally that, although the discussion in this section has been made for $S = M = N\,d_v/d_c$, the IVRG technique is also applicable for any value of $S \leq M$.

D. Maximum Indicator Optimization

Another important module of the PGDBF decoder is the maximum indicator (MI) module, which contains the critical path of the global architecture (Fig. 7), and therefore limits the achievable working frequency and decoding throughput [21].



(a) (b)

Fig. 12. Detailed circuits of implemented Energy Computation and Maximum Indicator blocks for $d_v = 3$ LDPC codes. (a) Energy Computation block, (b) Maximum Indicator block.

A rigorous survey on the maximum finder architectures can be found in [21], in which the authors confirmed that the implementation of finding the maximum value out of a list of N numbers is always a trade-off between computation cost and area/energy consumption. Since our goal is not only to get an optimized decoder with respect to multiple objectives, including hardware cost minimization, but also fast decoding throughput, we decided to focus on the minimization of the critical path for the MI module, even if it comes at the cost of an increased hardware resource. We therefore leave aside the solutions which minimize the area/consumption, such as the Traditional Binary Tree (TBT) method, and focus on the method which maximizes the decoding speed, i.e., the Leading-zero Counting Topology (LCT) [21]. For simplicity, we omit the iteration index (k) in this section.

The timing efficiency of LCT comes from the principle of processing the bits of an operand independently from other bits. In order to do this, the LCT method requires the operand to be represented in a special format called *one-hot format* with $q = \delta + 1$ quantization bits, where δ is the maximum value of the operand. In order to find the maximum in a set of N values by LCT, a q-bits vector is produced by applying N-inputs bit-wise OR operations on the N one-hot-format values. The maximum value is then sorted out by using a priority encoder on the results of these OR operations [21]. The complexity of LCT maximum finder comes from the N-inputs OR gates, and a reduction of the number of N-inputs OR gates would reduce significantly the complexity of decoders. In our LCT-based MI for PGDBF ($\delta = d_v + 1$), since E_{max} can never be equal to 0 except when $\mathbf{y}$ is already a codeword or $\mathbf{v}^{(k)}$ is converging to a codeword, we optimize the implementation of the LCT and reduce its complexity by representing the energy function values in a format with the length of $q - 1$ bits (instead of q bits) such that $E_n = E_n^{q-2}...E_n^1E_n^0$, $E_n^j = 1$ if $E_n = j + 1$, $E_n^i = 0$, $\forall i > j$ and E_n^i are don't-care otherwise. By doing this, all the operation of LCT is preserved while we can reduce by 1 N-inputs OR-gate.

TABLE I

HARDWARE RESOURCE USED TO IMPLEMENT THE PGDBF DECODERS AS A FUNCTION OF S. THE PERCENTAGES IN BRACKETS INDICATE THE ADDITIONAL HARDWARE COMPARED TO THE GDBF

dv3R050N1296 - AREA (μm^2)				
	S = Z = 54	**S = 4Z = 216**	**S = 12Z = 648**	**S = 24Z = 1296**
GDBF	53692 (+0%)			
LFSR-PGDBF	55837 (+4.00%)	57342 (+6.80%)	60360 (+12.42%)	64897 (+20.87%)
IVRG-PGDBF	55586 (+3.53%)	57295 (+6.71%)	60815 (+13.27%)	N/A

We further reduce by 1 the N-inputs OR-gates by applying logic minimization. Indeed, we can easily ignore the case of $E_{max} = 1$ and consider only the remaining possibilities (i.e., $E_n = j$, where $j = 2, \ldots, q - 1$). The case of $E_{max} = 1$ will be automatically considered when all the other possibilities are discarded. Thus, we reduce finally the number of N-inputs OR-gate to only $q - 2$ (originally q). Fig. 12 shows the detailed circuits for energy computation and MI blocks of $d_v = 3$ LDPC codes. The energy computation block (Figure 12(a)) produces the energy value, E_n, for VN v_n in the required format by using the neighbor CN values and XOR-ing results of v_n to y_n as its inputs. The MI block (Figure 12(b)) requires only 3 N-inputs OR gates (instead of 5) and block $C2$ realizes the logic minimization process as mentioned above. The $I_n, n = 1, \ldots, N$, is produced by the circuit as in Fig. 12(b) (block $C3$). In the $C3$ circuit, $I_n = 1, n = 1 \ldots N$ if and only if $Ma_j = 1$ and $E_n^j = 1$, $0 \leq j \leq q - 1$, concurrently, otherwise $I_n = 0$.

IV. SYNTHESIS RESULTS AND DECODING PERFORMANCE

A. PGDBF Synthesis Results

In this section, we report the ASIC results at post-synthesis level, of the proposed PGDBF implementations. The synthesis has been done targeting a 65 nm CMOS technology, using Synopsys tools.

For the first synthesis comparison, our goal is to demonstrate the area gains that one can achieve using the CSTS-PGDBF

TABLE II

FREQUENCY AND THROUGHPUT COMPARISON BETWEEN GDBF DECODER, PGDBF DECODERS, AND MS DECODERS [22], [23]. QC-LDPC $(d_v, d_c) = (3, 6)$, $R = 1/2$, $N = 1296$, $Z = 54$

	Code length	Code rate	AREA (μm^2)	kGE	Power (mW)	f_{max} (MHz)	N_c	$FER = 1e-5$		$\alpha = 0.01$	
								It_{ave}	θ (Gbit/s)	It_{ave}	θ (Gbit/s)
GDBF	1296	1/2	87810	75	337	222	1	2.00 (@$\alpha = 0.005$)	144.00	2.95 ($FER = 3e^{-4}$)	97.63
LFSR-PGDBF (S = M = 4Z = 216)	1296	1/2	100521 (+14.5%)	86	339	232	1	3.50 (@$\alpha = 0.012$)	86.11	2.88 ($FER = 4e^{-6}$)	104.65
IVRG-PGDBF (S = M = 4Z = 216)	1296	1/2	92645 (+5.5%)	79	323	232	1	3.50 (@$\alpha = 0.012$)	86.11	2.88 ($FER = 5e^{-6}$)	104.65
MS [22]	1296	1/2	720000	615	317	250	6	2.34 (@$\alpha = 0.025$)	23.08	1.29 ($FER = 1e^{-7}$)	41.86
MS [23]	648 - 1944	1/2 - 5/6	1023000	-	-	400	-	108 - 337 (Mbps) at $It_{max} = 20 - 25$ iterations 1.39 - 4.34 (Gbps) at $It_{ave} = 1.94$			

TABLE III

FREQUENCY AND THROUGHPUT COMPARISON BETWEEN GDBF, PGDBF DECODERS AND MS DECODER FROM [24]. QC-LDPC $(d_v, d_c) = (4, 34)$, $R = 0.88$, $N = 9520$, $Z = 140$

	Code length	Code rate	AREA (μm^2)	kGE	Power (mW)	f_{max} (MHz)	N_c	$FER = 5e-7$		$alpha = 0.03$	
								It_{ave}	θ (Gbit/s)	It_{ave}	θ (Gbit/s)
GDBF	9520	0.88	514760	439.6	173	100	1	2.1 (@$\alpha = 0.001$)	453.33	4.73 ($FER = 2e^{-4}$)	201.27
LFSR-PGDBF (S = M = 8Z = 1120)	9520	0.88	533394 (+3.61%)	455.52	182	100	1	3.95 (@$\alpha = 0.0025$)	241.01	4.68 ($FER = 2e^{-6}$)	203.42
IVRG-PGDBF (S = M = 8Z = 1120)	9520	0.88	533574 (+3.66%)	455.67	121	100	1	3.95 (@$\alpha = 0.0025$)	241.01	4.68($FER = 2e^{-6}$)	203.42
MS [24] **180nm**	8192	0.875	11300000	-	-	317	-	$It_{max} = 15$, $\theta = 5.1$			
scaled to 65nm	8192	0.875	1470000	-	-	877.8	-	$It_{max} = 15$, $\theta = 14.12$			

approach, i.e., using short size S for the random sequence $R_t^{(0)}$. The results are reported in Table I for a QC-LDPC code, with parameters $(d_v, d_c) = (3, 6)$, $R = 1/2$, $N = 1296$ and circulant size $Z = 54$ (denoted as dv3R050N1296). In this Table, we have constrained the implementations to run at the same clock frequency, by setting the timing constraint identical for all decoders, fixed to 8 ns. We choose this strategy to measure precisely the impact of S on the hardware cost, even if the working frequency is not maximized. We indicate in brackets the additional cost in percentage compared to the deterministic GDBF implementation. As a first remark, we can see that the LFSR-based and the IVRG-based implementations have similar costs. So, both solutions for the implementation of the RS sequence are equally competitive with respect to the hardware resource usage. As expected, the overhead is roughly proportional to the size of the RS memory S, and becomes very small (6% – 7%) for small values of $S \leq 4\,Z$. As we demonstrate in the next section, this small value of S is sufficient to obtain important error correction gains. We have verified that similar conclusions could be made for LDPC codes with different parameters, with various lengths, rates and values of d_v.

In the second synthesis comparison, we report working frequency and throughput in Tables II and III. Table II shows the results for the GDBF, PGDBF and two MS decoders taken from the literature [22], [23]. The code used for the GDBF, PGDBF and [22] is the $(d_v, d_c) = (3, 6)$, $R = 1/2$, $N = 1296$, $Z = 54$ LDPC code, while [23] considers the IEEE 802.11n standard codes with various lengths and rates. In Table III, a large length and high rate LDPC code is considered, and our decoders are compared with the MS implementation of [24].

For the GDBF and the PGDBF decoders, we performed the synthesis with the objective of optimizing the timing constraint, which results in the maximum frequency at which the decoder can operate.

We furthermore considered the following two scenarios: one scenario when the RS sequence is applied from the beginning of the decoding, and the other scenario when the deterministic GDBF is used for the first 10 iterations, and the RS sequence used for the remaining iterations. The reason of this later scenario is twofold. First, our statistical analysis from Fig. 3 showed that using the random sequences during the first iterations could be detrimental to the decoding performance, and that it is better to trigger the random feature of the PGDBF after a small number of iterations. Second, many of the noisy codewords do not require the strength of the PGDBF, and a simple GDBF could already correct them in a few iterations. Since the GDBF converges faster than the PGDBF, the average throughput is larger if we use GDBF, instead of the PGDBF, for the first 10 iterations. To support this claim, we plotted on Fig. 13 the average number of iterations as a function of α, for the $d_v = 3$, $d_c = 6$, $N = 1296$, QC-LDPC code. It can be seen that when we start using the PGDBF after 10 iterations, the average number of iterations is always lower or equal than the deterministic GDBF. We will use this scenario for the average throughput computations. The average throughput (θ) of the decoders is computed as $\theta = \frac{f_{max} * N}{It_{ave} * N_c}$ where f_{max} is the maximum working frequency provided by the synthesizer, It_{ave} is the average number of iterations and N_c is the number of clock cycles needed for one decoding iteration.

As seen in Table II, the hardware cost of the MS decoders is a lot larger than the BF-based decoders, and requires 7 to 10 times more area than the PGDBF. Our implementation of PGDBF allows to perform one iteration in $N_c = 1$ clock cycle, which results in a very important throughput gain of GDBF and PGDBF decoders over MS. We compared the average throughput of the decoders under two settings. For the same target performance, i.e., at FER$= 1e{-}5$, the PGDBFs have a throughput 4 times larger than the MS of [22], and only 67% lower than the deterministic GDBF. The second setting

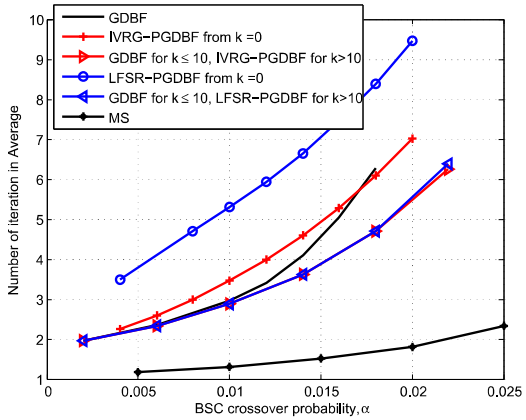


Fig. 13. Average number of iterations for GDBF, PGDBF and MS decoders on the dv3R050N1296 regular LDPC code. For PGDBF decoders, $S = 4Z$ and $p_0 = 0.7$ in LFSR-PGDBF.

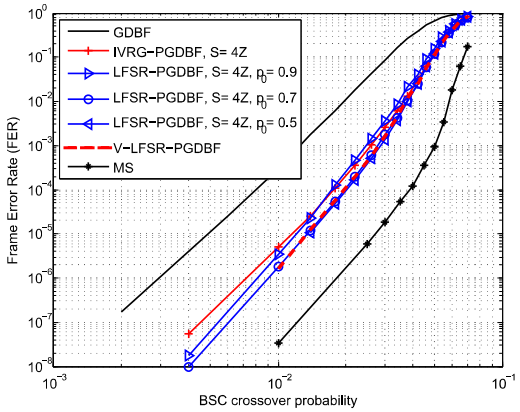


Fig. 14. Decoding performance of GDBF, LFSR-PGDBF, IVRG-PGDBF and MS decoders on the LDPC code $d_v = 3$, $d_c = 6$, $N = 1296$ (dv3R050N1296).

corresponds to all decoders working at the same level of channel noise, i.e., $\alpha = 0.01$. In this case, the PGDBFs have a larger throughput than GDBF and a gain of two orders of magnitude in FER. Compared to MS, the PGDBF have poorer performance (FER= $5e^{-6}$ compared to FER= $1e^{-7}$), but with 2 times faster throughput. The conclusions are confirmed when comparing with the MS architecture of [23], in which the authors report an architecture for which the area is 10 times larger than our PGDBF solutions, and with a throughput smaller than the one in [22].

Another demonstration of the advantages of our PGDBF implementations is presented in Table III, in which we considered LDPC code parameters that are especially interesting for storage applications. The code used for GDBF and PGDBF decoders is a QC-LDPC with parameters $(d_v, d_c) = (4, 34)$, $R = 0.88$, $N = 9520$, $Z = 140$. The average number of iterations is considered for the GDBF and PGDBF decoders, while we only have access to the maximum number of iterations for the MS architecture of [24]. For this long, high rate code, and $S = M$, the area overhead of our PGDBF compared to GDBF is slightly smaller than for the shorter code of Table II, which means that our implementations are more and more efficient as the code rate and codeword length increase. Compared with the MS implementation of [24], we can see that our PGDBF implementations occupy 3 times less area, while having a throughput at least 4 times faster than the MS (taking into account the difference between It_{max} and It_{ave}).

Overall, we can conclude that only with a few percents of hardware overhead compared to the classical GDBF, our proposed implementations of the PGDBF represent an interesting and competitive solution for very high throughput decoders.

B. PGDBF Performance

In this section, we illustrate the difference in decoding performance of PGDBF decoders in comparison with MS and GDBF decoders, on the BSC channel. The MS decoder is a layered version with 6 quantization bits for the APP-LLR and 4 quantization bits for the extrinsic messages, with a maximum of $It_{max} = 20$ decoding iterations. We consider two regular LDPC codes for the simulations: a QC-LDPC code with parameters $d_v = 3$, $d_c = 6$, rate 0.5, $N = 1296$ (dv3R050N1296), and a QC-LDPC code with $d_v = 4$, $d_c = 8$, rate 0.5, $N = 1296$ (dv4R050N1296). For the GDBF and the PGDBF, the maximum number of iterations is set to $It_{max} = 300$.

In Fig. 14, we show the result of testing different implementations of the PGDBF decoder, i.e., the IVRG-PGDBF and the LFSR-PGDBF with different RS probabilities p_0, on the dv3R050N1296 code. We have also simulated the LFSR-PGDBF with varying values of the RS probabilities $p_0^{(k)}$ along the iterations, under the following setting: $p_0 = 0.9$ for the first 100 iterations, $p_0 = 0.7$ if $100 < k \leq 200$, and $p_0 = 0.5$ if $200 < k \leq 300$. We labeled this decoder V-LFSR-PGDBF. It can be seen that all PGDBF decoders have roughly the same performance, as well as the V-LFSR-PGDBF, which confirms that a wide range of value of p_0 achieve approximately the same coding gain, and also that LFSR and IVRG versions are both competitive in terms of performance.

The PGDBF decoders perform halfway between the GDBF and the MS decoder. For the IVRG-PGDBF, the gain compared to GDBF decoder comes at only 6% chip area overhead and no throughput degradation as shown in Table II. The performance loss compared to MS was expected, and could be acceptable for applications that are very demanding in energy/throughput.

Figs. 15 and 16 show the decoding performance of PGDBF decoders for increasing values of the random sequence length S, for the dv4R050N1296 code. As expected, the greater value of S, the better decoding performance and $S = 12\,Z$ is sufficient to reach the performance of the theoretical PGDBF. The IVRG-PGDBF with $S = 12Z$ performs even better compared to the theoretical PGDBF at small α. For example FER $= 2e^{-6}$ for the IVRG-PGDBF compared to FER $= 4e^{-6}$ for the theoretical PGDBF, at $\alpha = 0.026$. This interesting behavior can be explained by the fact that IVRG-PGDBF has the feature of *adapting* the number of ones in the $R_t^{(0)}$ sequence to the channel noise realization, by producing more ones when the channel contains many errors, and less ones when it contains only a few errors. This feature of the IVRG-PGDBF can be interpreted as a PGDBF

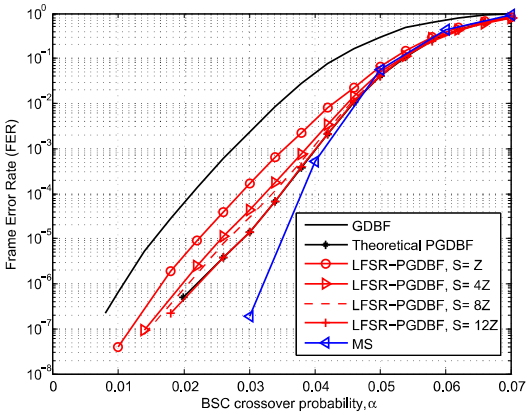


Fig. 15. Effect of the RS length S on the decoding performance of LFSR-PGDBF decoders ($p_0 = 0.9$) for the dv4R050N1296 code.

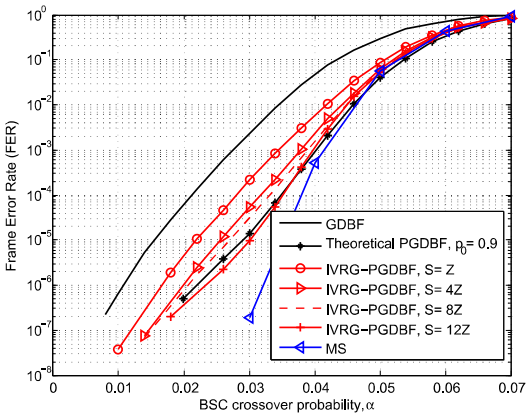


Fig. 16. Effect of the RS length S on the decoding performance of IVRG-PGDBF decoders for the dv4R050N1296 code.

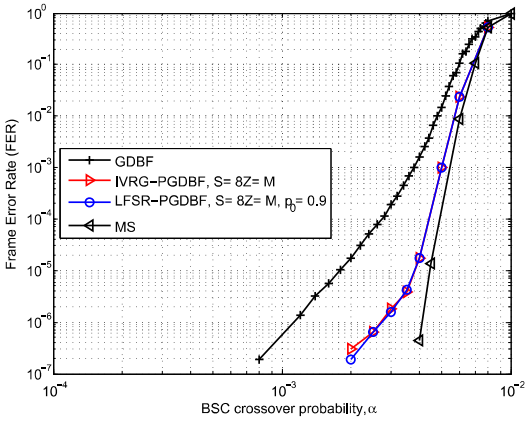


Fig. 17. Decoding performance of GDBF, PGDBF ($It_{max} = 300$) and MS ($It_{max} = 20$) decoders on a QC-LDPC code with $dv = 4$, $Rate = 0.88$, $Z = 140$, $M = 1120$ and $N = 9520$.

with adaptive p_0 parameter, the adaptation coming from the channel quality. Another interesting result is that the PGDBF decoders approach closely the performance of the MS decoder especially in the waterfall region. This means that for $d_v = 4$ codes, the PGDBF solution is even more competitive than for $d_v = 3$ codes.

Finally, we present on Fig. 17 the performance of our decoders for the $(d_v, d_c) = (4, 34)$, $R = 0.88$, $N = 9520$, $Z = 140$ of Table III. For this code, the PGDBF is especially very good as it closes to the MS results in the waterfall region, and starts showing an error floor at $FER < 10e^{-5}$. This very good performance results compared to the deterministic GDBF come at only 3.5% extra hardware cost, as indicated in Table III.

V. Conclusion

In this paper, we proposed an efficient hardware architecture to implement the PGDBF algorithm proposed recently as a promising hard-decision type iterative decoder with a performance approaching the MS decoder. We have focused on minimizing the resource overhead needed to implement the random perturbations of the PGDBF and on the optimization of the maximum indicator unit. Our random perturbation block is based on the use of a short random sequence that is duplicated to fully apply the PGDBF decoding rules. We also propose two different methods to initialize the short RS, LFSR-based and IVRG-based, with equivalent hardware overheads but with different behaviors on different LDPC codes. We showed, by implementing the LFSR-PGDBF and IVRG-PGDBF decoders on ASIC, that the proposed random perturbations require a very small extra complexity compared to the GDBF. We further improve the decoding throughput of our BF decoders by optimizing the Maximum Indicator using the LCT maximum finder in order to shorten the critical path. Compared to the MS decoder, the proposed PGDBF implementation offer 5 to 7 times faster throughput and requires 7 to 10 times less chip area, at the cost of a performance degradation, which is in all our simulations smaller than all the known hard decision decoders. These advantages in throughput and area make our PGDBF decoders a competitive hard-decision LDPC decoding solution for current and future standards.

References

[1] D. Declercq, M. Fossorier, and E. Biglieri, Eds., *Channel Coding: Theory, Algorithms, and Applications* (Academic Press Library in Mobile and Wireless Communications). New York, NY, USA: Elsevier, 2014.

[2] 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.

[3] Q. Huang, J. Kang, L. Zhang, S. Lin, and K. Abdel-Ghaffar, "Two reliability-based iterative majority-logic decoding algorithms for LDPC codes," *IEEE Trans. Commun.*, vol. 57, no. 12, pp. 3597–3606, Dec. 2009.

[4] M. Jiang, C. Zhao, Z. Shi, and Y. Chen, "An improvement on the modified weighted bit flipping decoding algorithm for LDPC codes," *IEEE Commun. Lett.*, vol. 9, no. 9, pp. 814–816, Sep. 2005.

[5] 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.

[6] J. Zhang and M. P. C. Fossorier, "A modified weighted bit-flipping decoding of low-density Parity-check codes," *IEEE Commun. Lett.*, vol. 8, no. 3, pp. 165–167, Mar. 2004.

[7] T. M. N. Ngatched, F. Takawira, and M. Bossert, "A modified bit-flipping decoding algorithm for low-density parity-check codes," in *Proc. IEEE Int. Conf. Commun.*, Glasgow, U.K., Jun. 2007, pp. 653–658.

[8] 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.

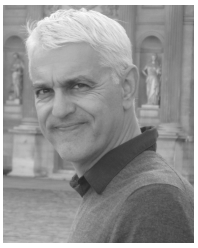
[9] *IEEE Standard for Information Technology—Corrigendum 2: IEEE Std 802.3an-2006 10GBASE-T Correction*, IEEE Standard 802.3-2005/ Cor 2-2007 (Corrigendum to IEEE Standard 802.3-2005), Aug. 2007.

[10] D. Divsalar, S. Dolinar, and C. Jones, "Short protograph-based LDPC codes," in *Proc. IEEE Military Commun. Conf. (MILCOM)*, Orlando, FL, USA, Oct. 2007, pp. 1–6.
[11] S. Jeon and B. V. K. Vijaya Kumar, "Performance and complexity of 32 k-bit binary LDPC codes for magnetic recording channels," *IEEE Trans. Magn.*, vol. 46, no. 6, pp. 2244–2247, Jun. 2010.
[12] K.-C. Ho, C.-L. Chen, Y.-C. Liao, H.-C. Chang, and C.-Y. Lee, "A 3.46 Gb/s (9141,8224) LDPC-based ECC scheme and on-line channel estimation for solid-state drive applications," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Lisbon, Portugal, May 2015, pp. 1450–1453.
[13] G. Dong, N. Xie, and T. Zhang, "On the use of soft-decision error-correction codes in NAND flash memory," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 58, no. 2, pp. 429–439, Feb. 2011.
[14] *IEEE-802.11n, Wireless LAN Medium Access Control and Physical Layer Specifiations: Enhancements For Higher Throughput*, IEEE Standard P802.11n/D3.07, Mar. 2008.
[15] *IEEE-802.16, Local and Metropolitan Area Networks—Part 16*, IEEE Standard 802.16a-2003.
[16] K. Le *et al.*, "Efficient realization of probabilistic gradient descent bit flipping decoders," in *IEEE Int. Symp. Circuits Syst. (ISCAS)*, Lisbon, Portugal, May 2015, pp. 1494–1497.
[17] R. M. Tanner, D. Sridhara, and T. Fuja, "A class of group-structured LDPC codes," in *Proc. 5th Int. Symp. Commun. Theory App.*, Ambleside, U.K., Jul. 2001.
[18] T. Richardson, "Error floors of LDPC codes," in *Proc. 41st Annu. Allerton Conf. Commun., Control, Comput.*, Oct. 2003, pp. 1426–1435.
[19] B. Vasić, P. Ivanis, and D. Declercq, "Approaching maximum likelihood performance of LDPC codes by stochastic resonance in noisy iterative decoders," in *Proc. Inf. Theory Appl. Workshop*, San Diego, CA, USA, Feb. 2016.
[20] A. K. Panda, P. Rajput, and B. Shukla, "FPGA implementation of 8, 16 and 32 Bit LFSR with maximum length feedback polynomial using VHDL," in *Proc. Int. Conf. Commun. Syst. Netw. Technol. (CSNT)*, May 2012, pp. 769–773.
[21] 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.
[22] 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. 2016, pp. 230–237.
[23] T. Brack *et al.*, "Low complexity LDPC code decoders for next generation standards," in *Proc. Design, Autom. Test Eur. Conf. Exhibit.*, Nice, France, Apr. 2007, pp. 331–336.
[24] J. Sha, Z. Wang, M. Gao, and L. Li, "Multi-Gb/s LDPC code design and implementation," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 2, pp. 262–268, Feb. 2009.

Khoa Le received his B.S. and M.S. degrees in electronics engineering from Ho Chi Minh City University of Technology (HCMUT), Vietnam, in 2010 and 2012, respectively. He is working toward the Ph.D degree at ETIS Laboratory, ENSEA, University of Cergy-Pontoise, CNRS UMR-8051, France. His research interests are in error correcting code algorithms, analysis and their implementations in FPGA/ASIC.

Fakhreddine Ghaffari received the Electrical Engineering and M.S. degrees from the National School of Electrical Engineering (ENIS), Tunisia, in 2001 and 2002, respectively. He received the Ph.D degree in electronics and electrical engineering from the University of Sophia Antipolis, France in 2006. He is currently an Associate Professor at the University of Cergy Pontoise, France. His research interests include VLSI design and implementation of reliable digital architectures for wireless communication applications in ASIC/FPGA platform and the study of mitigating transient faults from algorithmic and implementation perspectives for high-throughput applications.

David Declercq (SM'11) was born in June 1971. He received his Ph.D. degree in statistical signal processing from the University of Cergy-Pontoise, France, in 1998. He is currently a full Professor at the ENSEA in Cergy-Pontoise. He is the General Secretary of the National GRETSI association. He held a junior position at the Institut Universitaire de France from 2009 to 2014. His research topics lie in digital communications and error-correction coding theory. He worked several years on the particular family of LDPC codes, both from the code and decoder design aspects. Since 2003, he developed a strong expertise on non-binary LDPC codes and decoders in high order Galois fields GF(q). A large part of his research projects are related to non-binary LDPC codes. He mainly investigated two aspects: i) the design of GF(q) LDPC codes for short and moderate lengths, and ii) the simplification of the iterative decoders for GF(q) LDPC codes with complexity/performance tradeoff constraints. Dr. Declercq published more than 40 papers in major journals (IEEE TRANSACTIONS ON COMMUNICATIONS, IEEE TRANSACTIONS ON INFORMATION THEORY, COMMUNICATIONS LETTERS, EURASIP JWCN), and more than 120 papers in major conferences in information theory and signal processing.



Bane Vasić is a Professor of Electrical and Computer Engineering and Mathematics at the University of Arizona, Tucson, AZ, USA. He is affiliated with BIO5, the Institute for Collaborative Bioresearch, and is a Director of the Error Correction Laboratory. He is an inventor of the soft error-event decoding algorithm, and the key architect of a detector/decoder for Bell Labs data storage chips which were regarded as the best in industry. Dr. Vasić is a Chair of the IEEE Data Storage Technical Committee, Editor of the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS Special Issues on Data Storage Channels, and Member of the Editorial Board of the IEEE TRANSACTIONS ON MAGNETICS. He is da Vinci Circle and Fulbright Scholar and founder and Chief Scientific Officer of Codelucida.