Hardware Optimization of the Perturbation for Probabilistic Gradient Descent Bit Flipping Decoders

Khoa Le*, Fakhreddine Ghaffari*, David Declercq*, Bane Vasic†

*ETIS, ENSEA/UCP/CNRS UMR-8051, France, {khoa.letrung, fakhreddine.ghaffari, declercq}@ensea.fr

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

***Abstract*—The Probabilistic Gradient Descent Bit-Flipping (PGDBF) decoder has been proposed as a very promising hard-decision Low-Density Parity-Check (LDPC) decoder with a large gain in error correction. However, this impressive decoding gain is reported to come along with a non-negligible extra complexity due to the additional Perturbation Block (PB) required on top of the Gradient Descent Bit-Flipping (GDBF) decoder. In this paper, an efficient solution to implement this PB is introduced which is shown to keep the decoding gain as good as the theoretical PGDBF decoder while requiring a very small hardware overhead compared to the non-probabilistic GDBF. The proposed architecture is designed basing on a statistical analysis conducted to find the key features of the randomness needed to maintain the decoding gain and to reveal the simplification directions. The efficiency of our proposed method is confirmed by the synthesis results of decoder implementations on ASIC with 65nm CMOS technology and performance simulations.**

Keywords*—*Low-Density Parity-Check, Bit-Flipping decoder, random generator, low-complexity implementation.

I. Introduction

Low-Density Parity-Check (LDPC) codes provide an outstanding error correction capability and have been adopted to several communication standards such as IEEE 802.11n, 802.16a... Their practical decoders are classified into two groups: either *soft-information* decoders, *i.e.* Belief Propagation (BP), Min-Sum (MS) [1], which are very good in term of error correction but require an intensive computation or *hard-decision* decoders, *i.e.* Bit-Flipping (BF), Gradient Descent Bit-Flipping (GDBF), which are usually weak in error correction but very low-complexity. The BF-based decoders iteratively exchange only 1 bit information between their processing units known as Variable Nodes (VNs) and Check Nodes (CNs). These short messages therefore require very simple computation blocks and this is the source of simplicity of such decoders. However, this simplification leads to a significant loss in performance inspiring several improvement works [2],[3].

Gradient Descent Bit-Flipping algorithm was proposed by Wadayama *et al.* [2] and it offers a decoding performance superior to all known BF-based decoders applied on Additive White Gaussian Noise (AWGN). In GDBF decoder, Check Node (CN) unit computes the parity check on all VNs connected to it while Variable Node (VN) computes a function called *inversion* or *energy* function. GDBF provides a principle of finding the best suitable Variable Node (VN) (or group of VNs) to flip by deriving from a gradient descent formulation and minimizing a pre-defined function. The value of a VN is flipped when its energy function value (called hereafter as *energy value*) is a minima compared to those of all other VNs. Soon afterwards, authors in [3] introduced a variant of GDBF called Probabilistic Gradient Descent Bit-Flipping (PGDBF) by transforming GDBF to apply on Binary Symmetric Channel (BSC) and by adding a *probabilistic* manner in flipping of the VNs. PGDBF follows similarly the decoding steps of GDBF and the difference comes only from the flipping step of VN. All the VNs flipped in GDBF are actually flipped only with a probability of p ($p < 1$) in PGDBF. This modification surprisingly improved the decoding performance, far better than the original GDBF and very close to MS [3].

The PGDBF improvement in error correction comes with the cost of additional resources. Indeed, as reported in [4], a straightforward implementation of PGDBF on FPGA would multiply by 8 the number of registers and by 1.64 the Look-Up-Tables (LUTs) compared to non-probabilistic GDBF due to the implementation of the PB. Although the decoding gain of PGDBF is significant, this extra resource is non-negligible and need to be optimized. In this paper, an optimal solution to implement the PB of PGDBF is proposed and is shown to keep the decoding gain as the theoretical PGDBF while requires a very small extra resource. The proposed method is drawn through a statistical analysis on random sequence features affecting to the decoding performance. An important conclusion shown is that the binary random sequence triggered to VNs is not need to be too strong as in the straightforward implementation mentioned above. This random sequence can be re-used from one iteration to another or one random bit can even be triggered to multiple VNs while preserving the performance as using the independent random sequence.

The rest of the paper is presented as following. In Section II, the PGDBF algorithm are described. We also present the statistical analysis on the impact of random sequence to the decoding performance, whose conclusions guide the PB optimization. In Section III, the proposed PB with 2 initialization methods is described. The global PGDBF decoder architecture is also briefly presented in this section. We restrict ourselves to forcus on the PB in this paper and leave many other optimizations of the PGDBF decoder in its longer version [5]. In Section IV, we show for a test code, that the PGDBF with our PB requires only 5.5% hardware overhead compared to GDBF while preserving the theoretical PGDBF performance. This very small complexity overhead along with a large gain in decoding performance makes PGDBF algorithm as a very competitive hard-decisions LDPC decoder for current and next communication standard. Section V concludes the paper.

II. Probabilistic Gradient Descent Bit Flipping

A. Notations

A binary LDPC code is defined by a sparse parity check matrix H (M by N) ($M < N$). Each row represents a parity check function computed by a CN and each column represents a VN. The VN n ($0 \leq n \leq N-1$) and the CN m ($0 \leq m \leq M-1$) are called neighbors if the entry $H(m,n) = 1$. The neighbors set of the VN n is denoted as $\mathcal{N}(v_n)$ and similarly, the one of the CN m is denoted as $\mathcal{N}(c_m)$. The size of neighbor set is called node degree. We restrict ourselves to work only on the regular LDPC code, *i.e.* $|\mathcal{N}(c_m)| = d_c$, $|\mathcal{N}(v_n)| = d_v$, $\forall m, n$. A binary vector $\mathbf{x}$ of size N is called a codeword of LDPC code with H if $H.x^T = \mathbf{0}$. This codeword $\mathbf{x}$ is then transmitted through a Binary Symmetric Channel (BSC) in which each bit of $\mathbf{x}$ is flipped with a probability α called channel crossover probability. The output of the channel is denoted as $\mathbf{y}$. A Quasi-Cyclic LDPC code is the code that H is composed by the Z by Z sub-matrices which are either the cyclic-shift versions of a main diagonal or all-zero Z by Z matrices (Z is called the circulant size) [1].

B. *Probabilistic Gradient Descent Bit Flipping Decoder*

A PGDBF decoding process is defined as an iterative update of the VN values through the decoding iterations. We denote v_n (resp. c_m) as VN (resp. CN) itself while $v_n^{(k)}$ (resp. $c_m^{(k)}$) shows the node value at k-th iteration. The CN computation in PGDBF is written in Equ. 1 where $\oplus$ is the bit-wise Exclusive-OR operation. The PGDBF decoding process is terminated when all CNs equations are satisfied, *i.e.* $c_m^{(k)} = 0, \forall m$, or k reaches to the maximum value K allowed ($1 \leq k \leq K$), otherwise, the PGDBF decoder continues to decode. In the case of continuing to decode, the VNs follow steps to update their values, described in Equ. 2 and 3. First, each VN evaluates its energy value using Equ. 2. A VN energy value is a sum of the neighbors CN values and the similarity between the VN current value and its channel output, computed by $\oplus$ function. This energy function is sent to a module called Maximum Finder (MF) to sort out the maximum energy value among N values as in Equ. 4. Second, in Equ. 3, the VN n-th is flipped (XOR-ing with 1) if and only if the two following conditions are satisfied concurrently: its energy value is equal to the maximum energy indicated as 1 by the indicator function ($\mathbb{1}(.)$) and the value of $R_n^{(k)} = 1$ where $R_n^{(k)}$ is the n-th random bit in a random sequence R of length N at the k-th iteration ($R^{(k)}$), $R^{(k)} = \{R_n^{(k)} | 0 \leq n \leq N-1\}$, ($Pr(R_n^{(k)} = 1) = p$, $Pr(R_n^{(k)} = 0) = 1 - p$). The updated values of VNs are sent to the next decoding iteration.

$$c_m^{(k)} = \underset{v_n \in \mathcal{N}(c_m)}{\psi} (v_n^{(k)}) = \underset{v_n \in \mathcal{N}(c_m)}{\oplus} v_n^{(k)} \quad (1)$$

$$E_n^{(k)} = \underset{c_m \in \mathcal{N}(v_n)}{\phi} (v_n^{(k)}, y_n, c_m^{(k)}) = v_n^{(k)} \oplus y_n + \sum_{c_m \in \mathcal{N}(v_n)} c_m^{(k)} \quad (2)$$

$$v_n^{(k+1)} = v_n^{(k)} \oplus (\mathbb{1}(E_n^{(k)} = E_{max}^{(k)}) * R_n^{(k)}) \quad (3)$$

$$E_{max}^{(k)} = \max_{0 \leq n \leq N-1} (E_n^{(k)}) \quad (4)$$

The random sequence $R^{(k)}$ is produced by a PB block in which each bit is, in principle, independent to others and the $R^{(k)}$ is different from one iteration to another. The straightforward implementation of PB as in [4] is, however, costly and it is a bottleneck of the PGDBF decoder. We show in the next section that the PB can be significantly simplified in order to reduce the overhead resource, while preserving the decoding performance as good as the theoretical PGDBF decoder.

C. *Statistical analysis on the random sequence of PGDBF*

We conduct a statistical analysis by running the PGDBF decoder with the following setting. At $k = 0$, we generate a random binary sequence $R^{(0)}$ of length N with weight $\omega(R^{(0)})$, *i.e.* $\omega(R^{(0)}) = \sum_{n=0}^{N-1}(R_n^{(0)})$. During the decoding process ($k \geq 1$), we produce the new random sequence for next iteration, $R^{(k+1)}$, basing on the current random sequence $R^{(k)}$ by one of the following 2 ways: $R^{(k+1)} = F_r(R^{(k)})$ or $R^{(k+1)} = F_c(R^{(k)})$ where $F_r(R^{(k)})$ returns a random shuffling version of $R^{(k)}$ and $F_c(R^{(k)})$ does a cyclic-shift on $R^{(k)}$, *i.e.* $R_{(n+1)\%N}^{(k)} = R_n^{(k)}$ (% is modulus operation). The regular QC-LDPC code with $d_v = 3$, $d_c = 6$, code rate $R = 0.5$, $Z = 54$ and $N = 1296$ [7] is the test code throughout this paper. The decoding performance shown in the Fig. 1(a) as functions of $\omega(R^{(0)})$ and α, leads to 2 interesting remarks. First, the two methods of producing a new random sequence ($F_r(.)$ and $F_c(.)$) have strictly similar performance. Although the later method produces a less "random-like" and more correlation between iterations compared to the former one, its decoding performance is equivalent to other one. Second, an optimization on the value of p is not essential to decoding performance. Indeed, the curves flatten on a large range of $\omega(R^{(0)})$ which means that any PB generating a random sequence with weight between 800 to 1250 is sufficient to have good decoding performance for this LDPC code.

We push further the analysis by using a shorter sequence (denoted as R', and $|R'| = S$, $S < N$) and use $F_g(.)$ to produce a N-bits sequence such that: $R^{(k)} = F_g(R'^{(k)})$, where $F_g(R'^{(k)}) = \bigcup(\bigcup_{1}^{\lfloor \frac{N}{S} \rfloor}(R'^{(k)}), R'^{(k)}_{0,\ldots,N\%S-1})$ and $\bigcup(.)$ is the vector concatenation operation. From iteration k to $k+1$, we apply $F_c(.)$ only on $R'^{(k)}$ to produce the new sequence, $R'^{(k+1)} = F_c(R'^{(k)})$. This method obviously makes more correlation in the produced random sequence since some VNs can share a common random bit at a decoding iteration especially at the small value of S. Interestingly, the good performance is still maintained when using the short sequence R' with size only $S = 4Z$ (Fig. 1(b)). We can even use a smaller value of $S = Z = 54$ with an acceptable performance loss. We have verified all these conclusions with different LDPC codes with different codeword lengths and rates.

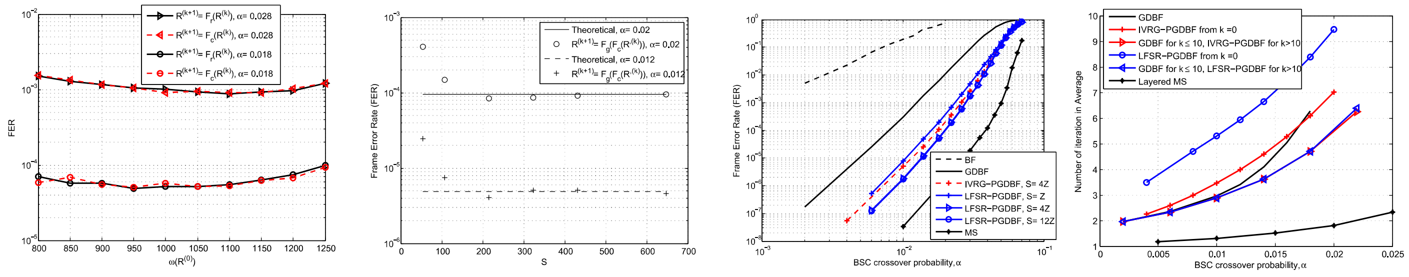
III. The Proposed Hardware Architecture

A. *The proposed perturbation architecture*

1) Cyclic shift and concatenation of a short sequence R': As a realization of the statistic analysis setting, our proposed PB architecture makes use a band of registers R', with size $S < N$, to store a generated random bits. A hard connection network is implemented to produce R from R', *i.e.* $R^{(k)} = F_g(R'^{(k)})$, and fully apply to N VNs as described in Fig. 2. In order to make the $R^{(k)}$ be different from an iterations to another, a cyclic shift operation is operated on R' after each iteration ($R'^{(k+1)} = F_c(R'^{(k)})$). By using the short sequence R' and the simple cyclic shift, the proposed PB significantly reduces the decoder complexity presented in the next section.

2) Two initializing methods for the short sequence R': When the registers band R' and the hard connection network are allocated, the registers in R' are needed to be initialized so that the total number of 1's in the projected sequence R falls into the interest range shown in Section II-C. We introduce in this section two initializing methods which satisfy this requirement with different decoding behavior.

In the first method, we make use of the conventional pseudo-random generating method, known as Linear Feedback Shift Register (LFSR), to initialize $R'^{(0)}$. The LFSR forms at its output an integer which is compared to a pre-defined threshold to produce a bit. By changing this threshold value,



(a) Decoding performance of PGDBF decoder with 2 methods of producing the random binary sequences.

(b) PGDBF decoder with a short sequence $R'^{(k)}$ with size of S, compared to theoretical PGDBF decoder.

(c) The comparison on decoding performance. BF-based decoders with $K = 300$ and MS with $K = 20$.

(d) The average number of iteration.

Fig. 1: Figures of the statistical analysis and decoding performance.

AREA (μm^2)					
	S = Z = 54	**S = 4Z = 216**	**S = 8Z = 432**	**S = 12Z = 648**	**S = 24Z = 1296**
GDBF	53692 (+0%)				
LFSR-PGDBF	55837 (+4.00%)	57342 (+6.80%)	58840 (+9.59%)	60360 (+12.42%)	64897 (+20.87%)
IVRG-PGDBF	55586 (+3.53%)	57295 (+6.71%)	59042 (+9.96%)	60815 (+13.27%)	N/A

TABLE I: Decoder complexity comparison.

the appearance probability of bit 1's (or 0's) is controlled. We use the 32-bits LFSR to make the generated correlation negligible [9]. One LFSR module is implemented and serially produces S random bits to initialize the registers of R'. The PGDBF decoder using the LFSR module to initialize R' is denoted as LFSR-PGDBF.

In the second method, we have introduced a specific way to generate a random sequence without using a real random generator (RG) [4]. Our method, called as Intrinsic-Valued Random Generator (IVRG), showed that CN values computed at the first iteration is already a random sequence. Typically, there is a fraction of bits 1's in the CN sequence (at the first iteration) when errors appear in the received codeword. The probability of a CN to be unsatisfied (be 1's) can be formulated as a function of α [4] and we found that, by complementing this sequence, the number of 1's always falls into the interest range shown in Section II-C. In order to initialize R', a hard connection, formed by a complement gate, is implemented connecting the output of a CN to the input of a register in sequence R'. At the first iteration, a signal triggers to store into the registers the complement of connected CN outputs, forming $R'^{(0)}$ and decoder is ready to decode. Since one CN output connects to one register in R', IVRG method can be applied for any value of $S \leq M$ and we denote the PGDBF with IVRG initialization as IVRG-PGDBF.

B. Implemented PGDBF Architecture

The other parts of PGDBF decoder are implemented as in Fig. 3. The CN operation is realized by a d_c-inputs XOR gate. Each VN unit requires 2 1-bit registers to store its channel output bit and intermediate value. Since the energy computation (the sum module) as well as the equality comparator and flipping gate (XOR gate) are implemented as combinatory circuit, the intermediate-value register is updated at each clock cycle. Because of that, our PGDBF decoder requires only 1

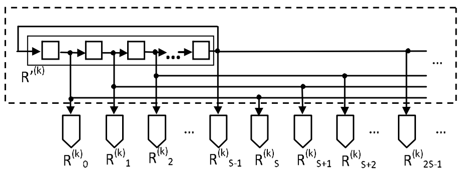


Fig. 2: The proposed perturbation architecture for PGDBF decoder.

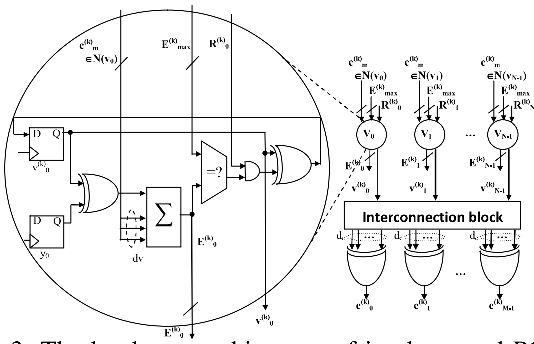


Fig. 3: The hardware architecture of implemented PGDBF.

clock cycle for 1 decoding iteration. We make use of Leading-Zero Counting Topology (LCT) [6] method to implement the Maximum Finder to shorten the critical path and maximize the decoding throughput. This MF receives the computed energies, $E_n^{(k)}$, from N VNs and is in charge of finding $E_{max}^{(k)}$ to send back at the $E_{max}^{(k)}$-input of each VN. Due to the lack of space, we do not show the details of the optimized MF as well as other PGDBF optimizations in this paper and put them in a longer version [5].

IV. Synthesis Results and Decoding Performance

A. Synthesis results

The designed BF decoders are synthesized on 65nm CMOS technology using Synopsys tools. We compare our implemented PGDBF decoders with the non-probabilistic GDBF and also with MS decoder benchmarks [7], [8] in literature. The MS implementation in [7] is for the same LDPC code ($d_v = 3$, $d_c = 6$, code rate $R = 0.5$, $Z = 54$ and $N = 1296$) while [8] considers the IEEE 802.11n standard codes with various lengths and rates.

We first highlight the effect of the size S of R' on the decoder complexity by keeping the timing constraint of all BF decoders identical (8ns) in synthesis process, assuming that all BF decoder are working on the same clock frequency. The results are reported in Table I. It can be seen that the hardware overhead of PGDBF is proportional to S and the 2 initializing methods are equivalent in term of hardware resource. Interestingly, the extra hardware for PGDBF decoder with $S = 4Z$, which is shown in the next section to be sufficient to have decoding performance similar to the theoretical PGDBF decoder, is only 6.71% for IVRG-PGDBF and 6.8% for LFSR-PGDBF. The hardware overhead can even be negligible (only 3.97%) by using $S = Z$ at a cost of small performance lost.

For the second comparison, we search for the maximum frequency (f_{max}) at which the decoders can operate and compute correspondingly the decoding throughput θ as $\theta = \frac{N * f_{max}}{k_{ave} * n_c}$ where k_{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 one decoding iteration. The corresponding area is also reported in the table (Table II). It can be seen that the PGDBF decoders require differently on the extra hardware. The LFSR-PGDBF requires 14.5% additional hardware compared to GDBF while IVRG-PGDBF needs only 5.5% at the same maximum operating frequency. The MS, as expected, is very much higher in complexity compared to BF decoders. The MS decoder in [7] is more than 7 times complexity compared to LFSR-PGDBF while in [8], it needs even more than 10 times. We make comparison on the decoding throughput in two scenarios. The fist scenario is for the same target performance, *i.e.* $FER = 1e^{-5}$, the IVRG-PGDBF decoder is 3 times faster than MS of [7] and it is $75.5/144 \simeq 52\%$ of GDBF. Note that, GDBF requires a very low level of noise ($\alpha = 0.005$) while other decoders are much higher ($\alpha = 0.014$ of LFSR-PGDBF and $\alpha = 0.025$ of MS). The second setting is that all decoders operate at the same level of noise, *i.e.* $\alpha = 0.01$. The PGDBF decoders again offer a very competitive throughput with 2 times faster than MS and comparable to GDBF (87 compared to 97 Gbps). The PGDBF decoders are, of course, weaker in error correcting, $FER = 5e^{-6}$, compared to $FER = 1e^{-7}$ of MS but are a lot stronger than GDBF ($FER = 3e^{-4}$). The advantage in throughput of PGDBF decoders over MS is confirmed by comparison with the MS implemented in [8]. The IVRG-PGDBF is more advantageous compared to LFSR-PGDBF in term of decoding throughput (around 10%) which comes from the lower average number of iterations as seen in Figure 1(d).

The PGDBF decoding throughput are, in general, inferior to the one of GDBF decoder. We further optimize the number of average iteration of PGDBF by applying the randomness only after a subsequent iterations, *i.e.* decoding by GDBF algorithm for $1 \leq k \leq 10$ and PGDBF algorithm with $10 < k \leq K$. This modification comes from the fact that some error patterns can be easily corrected by GDBF in few iterations without the strength of PGDBF (which decelerates the converging speed). The average number of iteration is significantly reduced and even better than GDBF decoder as shown in Figure 1(d). This results the better throughput of PGDBF, for example, with the modification at $\alpha = 0.01$, $k_{ave} = 4.68$, $\theta_{IVRG-PGDBF} = \theta_{LFSR-PGDBF} = 104.65$ Gbps while $\theta_{GDBF} = 97.63$ Gbps.

B. Decoding performance

The simulated decoding performance of decoders are shown in Fig. 1(c). In general, PGDBF decoder is the best BF decoder and approaching to a soft decision decoder (MS). We observe that with $S \geq 4Z$, decoding performance of PGDBF decoder is identical to the theoretical PGDBF. A small loss appears when $S = Z$ at the gain in complexity as mentioned previously. MS decoder, as expected, is the best in performance as seen in the Fig. 1(c).

V. Conclusion

We propose in this paper a simplified, low-complexity PB structures for the PGDBF decoder. Our PB is based on the use of a short register sequence and duplicated to apply to all VNs. Two initializing methods are introduced which form 2 types of PGDBF decoders named as LFSR-PGDBF and IVRG-PGDBF. The simulated performance and ASIC implementations confirm that PGDBF decoder has a very low-complexity, high throughput and good performance and it is a competitive candidate for current and next communication standard.

Acknowledgment

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 by the NSF under grants CCF-0963726, CCF-1314147 and ECCS-1500170. Bane Vasić acknowledges the support of the Fulbright Scholar Program.

References

[1] D. Declercq, M Fossorier, E Biglieri (Editors), "*Channel Coding: Theory, Algorithms, and Applications*", Academic Press Library in Mobile and Wireless Communications, Elsevier, ISBN: 978-0-12-396499-1, 2014.

[2] T. Wadayama*et al.* "Gradient descent bit flipping algorithms for decoding LDPC codes", *IEEE Trans. on Communications*, vol. 58, no. 6, pp. 1610–1614, June 2010.

[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, Sep. 2014.

[4] K. Le, D. Declercq, F. Ghaffari, C. Spagnol, P. Ivanis and B. Vasić, "Efficient Realization Of Probabilistic Gradient Descent Bit Flipping Decoders", *IEEE International Symposium on Circuits and Systems (ISCAS)* pp. 1494-1497, Lisbon, Portugal, May, 2015.

[5] K. Le, F. Ghaffari, D. Declercq and B. Vasić, "Efficient Realization Of Probabilistic Gradient Descent Bit Flipping Decoders", *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. PP, no. 99, 2016.

[6] B. Yuce, H. F. Ugurdag, S. Goren and G. Dundar, "Fast And Efficient Circuit Topologies For Finding The Maximum Of *n k-bit* Numbers", *IEEE Trans. on Computers*, vol. 63, no. 8, pp. 1868–1881, Aug. 2014.

[7] T. Nguyen-Ly, T. Gupta, M. Pezzin, V. Savin, D. Declercq and Sorin Cotofana. "Flexible, Cost-Efficient, High-Throughput Architecture For Layered LDPC Decoders With Fully-Parallel Processing Units", *Euromicro Conference on Digital System Design*, Cyprus, 2016.

[8] T. Brack *et al.*, "Low Complexity LDPC Code Decoders For Next Generation Standards", *Design, Automation and Test in Europe Conference and Exhibition*, pp. 1-6, Nice, 2007.

[9] A. K. Panda, P. Rajput and B. Shukla, "FPGA Implementation Of 8, 16 And 32 Bit LFSR With Maximum Length Feedback Polynomial Using VHDL", *IEEE International Conference on Communication Systems and Network Technologies*, Gujarat, India, pp. 769-773, May 2012.

	Code length	**Code rate**	**AREA** (μm^2)	f_{max} (MHz)	n_c	$FER = 1e^{-5}$		$\alpha = 0.01$	
						k_{ave}	θ (Gbit/s)	k_{ave}	θ (Gbit/s)
GDBF	1296	1/2	87810 (+0%)	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%)	232	1	4.89 (@$\alpha = 0.014$)	61.6	3.84 ($FER = 4e^{-6}$)	78.5
IVRG-PGDBF (S = M = 4Z = 216)	1296	1/2	92645 (+5.5%)	232	1	3.99 (@$\alpha = 0.012$)	75.5	3.45 ($FER = 5e^{-6}$)	87.4
MS [7]	1296	1/2	720000	250	6	2.34 (@$\alpha = 0.025$)	23.08	1.29 ($FER = 1e^{-7}$)	41.86
MS [8]	648 - 1944	1/2 - 5/6	1023000	400	-	108 - 337 (Mbps) at $K = 20 - 25$ iterations 1.39 - 4.34 (Gbps) at $k_{ave} = 1.94$			

TABLE II: Comparison between GDBF decoder, PGDBF decoders, and MS decoders [7], [8].