# Lightweight Hardware Architecture for Probabilistic Gradient Descent Bit Flipping on QC-LDPC Codes

Khoa Le*, Fakhreddine Ghaffari*, Lounis Kessal*, David Declercq*, Valentin Savin†, Oana Boncalo‡
*ETIS, UMR-8051, Université Paris Seine, Université de Cergy-Pontoise, ENSEA, CNRS, France,
{khoa.letrung, fakhreddine.ghaffari, kessal, declercq}@ensea.fr
†CEA-LETI, MINATEC Campus, Grenoble, France, valentin.savin@cea.fr
‡ Computer Engineering Department, University Politehnica Timisoara, Romania, oana.boncalo@cs.upt.ro

***Abstract*—The Probabilistic Gradient Descent Bit-Flipping (PGDBF) decoder offers a significant improvement in decoding performance for Low-Density Parity-Check (LDPC) codes on Binary Symmetric Channel (BSC). However, this outstanding decoding performance comes along with a non-negligible extra hardware cost to realize the probabilistic behavior on top of the deterministic Gradient Descent Bit-Flipping (GDBF) decoder. This paper presents a novel solution to implement PGDBF decoder on Quasi-Cyclic LDPC codes. The proposed architecture takes advantage of the cyclic shift permutation nature of QC-LDPC and changes the message flow such that a probabilistic behavior is emulated without the cost of an actual probabilistic signal generator. It is shown that, the proposed architecture improves the PGDBF decoding performance with respect to the state-of-the-art implementation while reducing hardware complexity, even being lower than that of the deterministic GDBF. The efficiency of our proposed method is verified through the ASIC 90nm CMOS technology implementations and decoding simulations.**

## I. Introduction

Low-Density Parity-Check (LDPC) code was introduced by Gallager in 1963 [1] and it has gained interest over the last several years due to the near-capacity error correction capability [2]. QC-LDPC are a specific code constructing method of LDPC codes that are well-suited for the practical implementations. QC-LDPC can be effectively decoded by the either *soft-information* message passing decoders, *i.e.* Min Sum (MS), Sum Product (SP) [2], or *hard-decision* decoders, *i.e.* Bit Flipping (BF), Gallager-A,B [1]. This paper focuses on the BF decoder due to its low-complexity, high throughput. Furthermore, recent BF optimizations proposed in [3][4] demonstrated good error correction capability, approaching and even surpassing 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 BSC channel. PGDBF iteratively 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 counterpart (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.

In this work, we introduce a new hardware architecture, called Variable-Node Shift Architecture (VNSA), to implement the PGDBF on QC-LDPC codes. The VNSA uses the regular structure of QC-LDPC codes to change the message flow of the decoder (by cyclically shifting the messages), and by doing that, a VN is cyclically processed by different VN processing units (VNUs) during the decoding process. Different VNU types are implemented and when the VNs are cyclically shifted, the probabilistic behavior is realized without the requirement of random generator. The VNSA-based PGDBF implementations are shown to not only reduce the decoder complexity, even being lower than GDBF, 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 VNSA and two implementations of VNSA-based PGDBF, the VNSA-PGDBF and its imprecise version called VNSA-IM-PGDBF. In Section IV, we show for a test code that, the VNSA-PGDBF provides a decoding performance as good as the optimized PGDBF implementation in literature while reducing $11\%$ hardware cost of the GDBF. The VNSA-IM-PGDBF reduces $18\%$ hardware cost of GDBF while providing a significant gain in error correction, around 1dB gain over the VNSA-PGDBF. Section V concludes the paper.

## II. PGDBF Decoding and its implementation

### A. Notations

An LDPC code is defined by a sparse parity-check matrix $H$ $(M, N)$, $N > M$. Each row represents a parity check function, computed by a CN, on the VNs represented by the columns. The Quasi-Cyclic is a specific constructing method of $H$ in which each entry of a small *base matrix* $H_B$ $(n_r \times n_c)$ is expanded to obtain the full matrix $H$ ($N = n_c \times Z$, $M = n_r \times Z$). $H_B = \{h_{a,i}\}^{n_r \times n_c}$ where $1 \leq a \leq n_r, 1 \leq i \leq n_c, -1 \leq h_{a,i} \leq Z - 1$. The entries $h_{a,i} \neq -1$ are replaced by the circulant shift of a $Z \times Z$ diagonal matrix with circulant shift factor $h_{a,i}$ while the entries $h_{a,i} = -1$ are replaced by the *all-zero* $Z \times Z$ matrices. In QC-LDPC code, each base column corresponds to $Z$ VNs and each base row corresponds to $Z$ CNs. We denote the $j$-th VN ($1 \leq j \leq Z$) belonging to the $i$-th base column ($1 \leq i \leq n_c$) as $v_{i,j}$, and similarly, the $b$-th CN ($1 \leq b \leq Z$) belonging to the $a$-th base row ($1 \leq a \leq n_r$) as $c_{a,b}$. The superscript $(k)$ is added on the node notations to indicate the values at the $k$-th iteration. For example, the VN output vector at the $k$-th iteration is expressed as $\mathbf{v}^{(k)} = (\mathbf{v}_1^{(k)}, \mathbf{v}_2^{(k)}, ..., \mathbf{v}_{n_c}^{(k)})$ where $\mathbf{v}_i^{(k)} = (v_{i,1}^{(k)}, v_{i,2}^{(k)}, ..., v_{i,Z}^{(k)})$, $1 \leq i \leq n_c$ and the CN output vector at the $k$-th iteration is as $\mathbf{c}^{(k)} = (\mathbf{c}_1^{(k)}, \mathbf{c}_2^{(k)}, ..., \mathbf{c}_{n_r}^{(k)})$

978-1-5386-4881-0/18/$31.00 ©2018 IEEE

where $\mathbf{c}_a^{(k)} = (c_{a,1}^{(k)}, c_{a,2}^{(k)}, ..., c_{a,Z}^{(k)})$, $1 \le a \le n_r$. The neighbor set of the VNs $\mathbf{v_i}$ is defined by the set of the CNs $\mathbf{c_a}$ with $h_{a,i} \neq -1$ and denoted by $\mathcal{N}(\mathbf{v_i})$. Similarly, the neighbor set of the CNs $\mathbf{c_a}$ is all the VNs $\mathbf{v_i}$ with $h_{a,i} \neq -1$ and denoted by $\mathcal{N}(\mathbf{c_a})$. It is clear that, each VN in $\mathbf{v_i}$ has $|\mathcal{N}(\mathbf{v_i})|$ neighbor CNs and the value $|\mathcal{N}(\mathbf{v_i})|$ is called the VN degree. Also, each CN in $\mathbf{c_a}$ has $|\mathcal{N}(\mathbf{c_a})|$ neighbor VNs and the value $|\mathcal{N}(\mathbf{c_a})|$ is called CN degree. In this work, we focus on the regular QC-LDPC code, *i.e.* $|\mathcal{N}(\mathbf{v_i})| = d_v$ and $|\mathcal{N}(\mathbf{c_a})| = d_c$, $\forall i, a$.

A vector $\mathbf{x} = \{0,1\}^N$ is called a codeword if and only if $H\mathbf{x}^T = 0$. We follow the notation of VNs in denoting $\mathbf{x}$ such that, $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{n_c})$ where $\mathbf{x}_i = (x_{i,1}, x_{i,2}, ..., x_{i,Z})$, $1 \le i \le n_c$. The analysis in this work is for the BSC where each bit $x_{i,j}$ is flipped with a probability $\alpha$, called the channel crossover probability. $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_{n_c})$, $\mathbf{y}_i = (y_{i,1}, y_{i,2}, ..., y_{i,Z})$, $1 \le i \le n_c$ is denoted as the vector received at the channel output, *i.e.*, $\Pr(y_{i,j} = x_{i,j}) = 1 - \alpha$ and $\Pr(y_{i,j} = 1 - x_{i,j}) = \alpha$, $1 \le i \le n_c, 1 \le j \le Z$.

In order to present the QC-LDPC BF decoding algorithms, we define the vector cyclic-shifting functions $S(\ell, \mathbf{v_i})$ and $\bar{S}(\ell, \mathbf{c_a})$, where $\ell$ is an integer, $0 \le \ell \le Z-1$, called the cyclic shift factor. If $\tilde{\mathbf{v}}_\mathbf{i} = S(\ell, \mathbf{v_i})$, then $\tilde{v}_{i,l+1} = v_{i,(l+\ell)\%Z+1}$, $\forall l = 0 \dots Z-1$ and if $\bar{\mathbf{c}}_\mathbf{a} = \bar{S}(\ell, \mathbf{c_a})$ then $\bar{c}_{a,l+1} = c_{a,(Z+l-\ell)\%Z+1}$, $\forall l = 0 \dots Z-1$.

### B. PGDBF decoder for QC-LDPC codes

In PGDBF, the CN computes the parity check of the neighboring VNs and its computing equation is formulated as in Equ. 1a (each element is computed by Equ. 1b) where $\oplus$ is the bit-wise Exclusive-OR (XOR) operation and $\tilde{\mathbf{v}}_i^{(k)} = S(h_{a,i}, \mathbf{v}_i^{(k)})$. The decoding process is terminated when all CNs are satisfied, *i.e.* $\mathbf{c}^{(k)} = \mathbf{0}$, or $k$ reaches the maximum $It_{max}$ iterations.

$$\mathbf{c}_a^{(k)} = \underset{\mathbf{v}_i \in \mathcal{N}(\mathbf{c}_a)}{\psi} (\tilde{\mathbf{v}}_i^{(k)}) = \underset{\mathbf{v}_i \in \mathcal{N}(\mathbf{c}_a)}{\oplus} (\tilde{\mathbf{v}}_i^{(k)}) \tag{1a}$$

$$c_{a,b}^{(k)} = \underset{\mathbf{v}_i \in \mathcal{N}(\mathbf{c}_a)}{\oplus} (\tilde{v}_{i,b}^{(k)}), \quad 1 \le b \le Z \tag{1b}$$

At each iteration, every VN of PGDBF computes its energy $E_{i,j}^{(k)}$ ($E_{i,j}^{(k)}$ denotes the energy of VN $v_{i,j}$ at iteration $k$). We also denote vector $\mathbf{e}^{(k)}$ gathering all VN energies, $\mathbf{e}^{(k)} = (\mathbf{e}_1^{(k)}, \mathbf{e}_2^{(k)}, ..., \mathbf{e}_{n_c}^{(k)})$, $\mathbf{e}_i^{(k)} = (E_{i,1}^{(k)}, E_{i,2}^{(k)}, ..., E_{i,Z}^{(k)}), 1 \le i \le n_c$. The energy computation is described in Equ. 2a and each element is computed as in Equ. 2b where $\bar{\mathbf{c}}_a^{(k)} = \bar{S}(h_{a,i}, \mathbf{c}_a^{(k)})$.

$$\mathbf{e}_i^{(k)} = \underset{\mathbf{c}_a \in \mathcal{N}(\mathbf{v}_i)}{\phi} (\mathbf{v}_i, \mathbf{y}_i, \bar{\mathbf{c}}_a^{(k)}) \tag{2a}$$

$$E_{i,j}^{(k)} = x_{i,j} \oplus y_{i,j} + \sum_{\mathbf{c_a} \in \mathcal{N}(\mathbf{v_i})} \bar{c}_{a,j}^{(k)} \tag{2b}$$

The maximum energy, denoted by $E_{max}^{(k)}$, is computed at each iteration by Equ. 3a. The vector $\mathbf{q}^{(k)}$ indicates whether or not the energy value of each VN equals to the maximum energy. $\mathbf{q}^{(k)} = (\mathbf{q}_1^{(k)}, \mathbf{q}_2^{(k)}, ..., \mathbf{q}_{n_c}^{(k)}), \mathbf{q}_i^{(k)} = (q_{i,1}^{(k)}, q_{i,2}^{(k)}, ..., q_{i,Z}^{(k)}), 1 \le i \le n_c$. $q_{i,j}^{(k)} = 1$ if $E_{i,j}^{(k)} = E_{max}^{(k)}$, otherwise, $q_{i,j}^{(k)} = 0$.

$$E_{max}^{(k)} = \max_{1 \le i \le n_c, 1 \le j \le Z} (E_{i,j}^{(k)}) \tag{3a}$$

$$\mathbf{q}^{(k)} = \mathbb{1}(E_{max}^{(k)}, \mathbf{e}^{(k)}) \tag{3b}$$

The flipping operation of PGDBF is described by Equ. 4a where $\mathbf{r}^{(k)}$ is the random generated vector and $\otimes$ is the logic AND operation. $\mathbf{r}^{(k)} = (\mathbf{r}_1^{(k)}, \mathbf{r}_2^{(k)}, ..., \mathbf{r}_{n_c}^{(k)}), \mathbf{r}_i^{(k)} = (R_{i,1}^{(k)}, R_{i,2}^{(k)}, ..., R_{i,Z}^{(k)}), 1 \le i \le n_c$, $\Pr(R_{i,j}^{(k)} = 1) = p_0$, $\Pr(R_{i,j}^{(k)} = 0) = 1 - p_0$. It can be seen that, each VN of GDBF is flipped (XORed by 1) only when the two conditions are simultaneously satisfied: 1. the energy is equal to the maximum energy ($q_{i,j}^{(k)} = 1$) and 2. the random signal $R_{i,j}^{(k)} = 1$. We also introduce the flipping operation of the deterministic GDBF in Equ. 4b. It becomes apparent that, a VN of GDBF will be flipped only when its energy equals to the maximum energy.

$$\mathbf{v}^{(k+1)} = \mathbf{v}^{(k)} \oplus (\mathbf{q}^{(k)} \otimes \mathbf{r}^{(k)}) \tag{4a}$$

$$\mathbf{v}^{(k+1)} = \mathbf{v}^{(k)} \oplus \mathbf{q}^{(k)} \tag{4b}$$

The implementations of PGDBF can be found in [6][7]. In these implementations, $N$ VNUs ($n_c$ groups of $Z$ VNUs) are implemented to process $N$ VNs in parallel. Similarly, $M$ $d_c$-XOR gates ($n_r$ groups of $Z$ $d_c$-XOR gates) are implemented for $M$ CNs (each XOR gate is called the check node processing unit - CNU). Each VN (resp. CN) is processed by the same (fixed) VNU (resp. CNU) during the decoding process. For each iteration, each VNU has 2 different behaviors corresponding to 2 possibilities of the probabilistic signals ($R_{i,j}^{(k)} = 0$ and $R_{i,j}^{(k)} = 1$). The above implementations of PGDBF require an inevitable additional hardware compared to GDBF due to the fact that on top of the GDBF, the random signal generating module is required. We refer hereafter this particular PGDBF implementation as the conventional implementation and its VNU as the conventional VNU. The connection networks in QC-LDPC decoders are the realizations (by the hard wires) of the functions $S(\ell, \mathbf{v_i})$ or $\bar{S}(\ell, \mathbf{c_a})$ and therefore, they are highly regular such that, 2 consecutive VNUs (CNUs) in a base column will receive the messages from 2 consecutive CNUs (VNUs). We exploit this property to propose the VNSA architecture described in the next section.

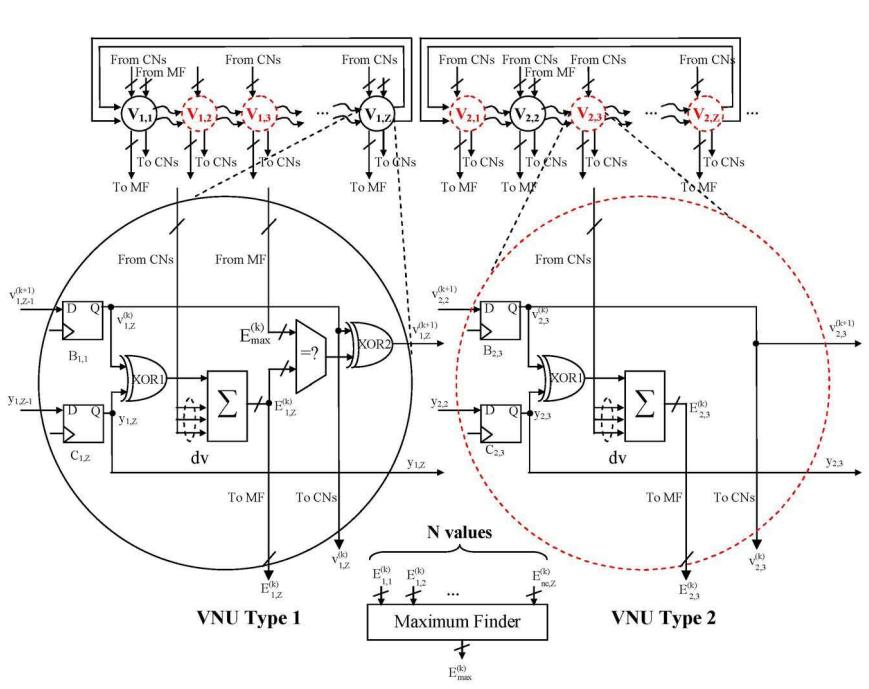
## III. The implementations of PGDBF using VNSA

### A. The principle of VNSA

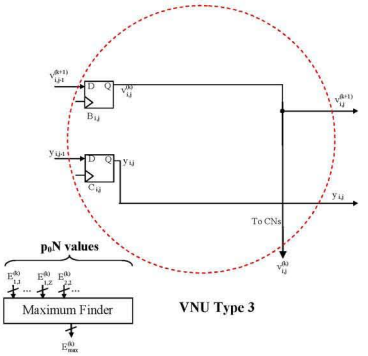
The key feature of VNSA is that, each VN (and also, each CN) is processed by different types of VNUs (CNUs) from one iteration to another while not changing the implemented connection networks. Implementation wise, the main difference of VNSA with respect to the conventional architecture is that, the VN messages and the channel values are cyclically shifted by a factor of $\ell$ before being conveyed by the connection network at each iteration, *i.e.* $\mathbf{v'}_i^{(k)} = S(\ell, \mathbf{v}_i^{(k)})$ and $\mathbf{y}_i'^{(k)} = S(\ell, \mathbf{y}_i^{(k)})$, $\mathbf{y}_i^{(0)} = \mathbf{y}_i$, $\forall i = 1 \dots n_c$, where $\ell$ is a constant chosen when implementing the decoder ($1 \le \ell \le Z-1$) (the notations with the prime ($'$) is used to denote the messages and signals of the decoder running on the VNSA).

When implementing a deterministic decoder, VNSA provides the same decoding results to that one implemented in the conventional hardware architecture. We illustrate for the case of GDBF algorithm (Equ. 1a, 2a, 3a, 3b and 4b) as following. First, when a correct codeword is found, the decoding processes of the two implementations will be terminated at the same time. It is because $\mathbf{c}_a'^{(k)} = \underset{\mathbf{v}_i \in \mathcal{N}(\mathbf{c}_a)}{\psi} (S(h_{a,i}, S(\ell, \mathbf{v}_i^{(k)}))) = S(\ell, \underset{\mathbf{v}_i \in \mathcal{N}(\mathbf{c}_a)}{\psi} (S(h_{a,i}, \mathbf{v}_i^{(k)}))) = S(\ell, \mathbf{c}_a^{(k)})$ and when $\mathbf{c}_a^{(k)} = \mathbf{0}, \forall a = 1 \dots n_r$ then $\mathbf{c}_a'^{(k)} = \mathbf{0}, \forall a = 1 \dots n_r$. Second, at each

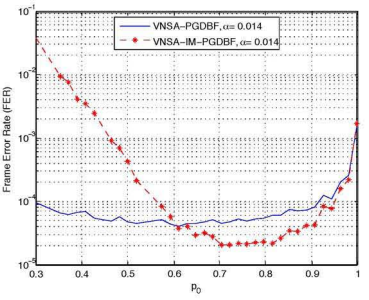
978-1-5386-4881-0/18/$31.00 ©2018 IEEE




(a) The PGDBF implementation using VNSA (VNSA-PGDBF).




(b) The type-3 VNU.




(c) FER vs $p_0$.

Figure 1. The VNSA-based PGDBF implementation and the statistical analysis.

iteration if the correct codeword is not found, the same VNs on the two implementations are flipped. It is due to the provable fact that, $\mathbf{e}'^{(k)}_i = S(\ell, \mathbf{e}^{(k)}_i)$, $E'^{(k)}_{max} = E^{(k)}_{max}$, $\mathbf{q}'^{(k)}_i = S(\ell, \mathbf{q}^{(k)}_i)$ and finally, $\mathbf{v}'^{(k+1)}_i = \mathbf{v}'^{(k)}_i \oplus \mathbf{q}'^{(k)}_i = S(\ell, \mathbf{v}^{(k+1)}_i)$. The cyclic shift of $\mathbf{v}'^{(k+1)}_i$ (and $\mathbf{c}'^{(k)}_a$), compared to that of the conventional implementation, implies the fact that, all VNs (and CNs) are processed cyclically by different VNUs (CNUs). Note that, the realization of $\mathbf{v}'^{(k)}_i = S(\ell, \mathbf{v}^{(k)}_i)$ (and also $\mathbf{y}'^{(k)}_i = S(\ell, \mathbf{y}^{(k)}_i)$) will not cost the hardware resource since it changes only the connection wires to store $\mathbf{v}^{(k+1)}_i$ into message memory (see in the next section). Although the VNSA works for any value of $\ell$, $1 \leq \ell \leq Z-1$, the applications of VNSA in this work are with $\ell = 1$.

### B. The Implementation of PGDBF using VNSA

The implementation of PGDBF using VNSA is presented in Fig. 1(a) and we denote it as VNSA-PGDBF. The implementation of the VNSA principle is the modifying of the hard-wires to the storing memory as seen in Fig. 1(a). The VN values for the next iteration, $v^{(k+1)}_{i,j}$ is stored to the memory $B_{i,(j\%Z)+1}$ instead of $B_{i,j}$. The same modification is applied for the channel values stored in $C_{i,j}$ memory.

In order to realize the probabilistic behavior of VNSA-PGDBF, 2 types of VNUs corresponding to the 2 possibilities of the probabilistic input signal are designed. The type-1 VNU mimics the operation of the conventional VNU when the random signal is equal to 1. In this case, the $\otimes$ in the Equ. 4a can be removed since $X$ **AND** $1 = X$, $\forall X$. It can be seen that, this type-1 VNU is the same as the VNU of GDBF decoder since the flip of the VN (operated by the XOR2) depends only on the equality between the VN energy and the maximum energy value. The type-2 VNU is designed by reproducing the operation of the conventional VNU when the random signal equals to 0. In this case, the $\otimes$ in the Equ. 4a produces 0 regardless the equality comparison result $q^{(k)}_{i,j}$. The VN value for the next iteration, $v^{(k+1)}_{i,j}$, is the one of the current iteration, $v^{(k)}_{i,j}$, since $v^{(k+1)}_{i,j} = v^{(k)}_{i,j}$ **XOR** $0 = v^{(k)}_{i,j}$. The type-2 VNU is designed by simply propagating directly the current value $v^{(k)}_{i,j}$ to the output and removing the XOR2 gate. Furthermore, the equality comparator can also be eliminated without affecting to the VNU operation.

Although several logic blocks of the conventional VNU are eliminated, leading the VNUs of VNSA-PGDBF to be simpler, the concept of PGDBF decoder is still preserved. The randomness is being emulated, without explicit use of probabilistic signal generator. Indeed, the operation of VNSA-PGDBF is identical to the version of CSTS-PGDBF decoder with $S = Z$. When the type-1 and type-2 VNUs are allocated in each base column similarly and correspondingly to the 1 and 0 in the $R_t$ ($|R_t| = Z$), by cyclically shifting the VNs through these VNUs, each VN will see the VNU types (type-1 and type-2) with the same order of having bit 1 and bit 0 at the random input of conventional VNU in CSTS-PGDBF when cyclically shifting $R_t$. Therefore, the VNSA-PGDBF and CSTS-PGDBF provide the same decoding results.

### C. An Imprecise Implementation of PGDBF on QC-LDPC

This section presents a further simplification of the VNSA-PGDBF, called VNSA-IM-PGDBF. The main modification of VNSA-IM-PGDBF is the replacement of the type-2 VNUs by type-3 VNUs (described in Fig. 1(b)). The type-3 VNU behaves similarly to the VNU in conventional PGDBF having 0 at the random input. However, this VNU type does not compute the energy value and corresponding logic is removed. The underlying idea is that since the VNU forwards to current iteration value regardless of the energy value, we may remove its energy contribution to the max finder as well. This helps

978-1-5386-4881-0/18/$31.00 ©2018 IEEE

further reduce the hardware complexity of VNSA-PGDBF decoder.

The low complexity of VNSA-IM-PGDBF is obtained due to the simpler VNUs and the simpler Maximum Finder (MF). Indeed, it can be seen that, the type-3 VNU does not cost any hardware resources except the memory elements and the saving is, therefore, controlled by the value of $p_0$. The MF in VNSA-IM-PGDBF (Figure 1(b)) needs to compute the maximum energy only among $p_0 N$ input values instead of $N$ which help save additional hardware resources and improve timing by reducing the critical path.

An issue of VNSA-IM-PGDBF is that, the maximum energy value found in some iterations may not be the true maximum. This comes from the fact that, the maximum energy is found among $p_0 N$ instead of $N$ candidates and the impreciseness manifests when type-3 VNUs are the only ones with the maximum energy value. In order to evaluate the effect of this impreciseness, we have carried out a statistical analysis of the error correction performance (the Frame Error Rate - FER) as a function of $p_0$ and the results are shown in Fig. 1(c). The test code used is with the parameters $(d_v, d_c) = (3, 6)$, $Z = 54$, $M = 648$, $N = 1296$, code rate $R = 0.5$ (denoted as dv3R050N1296). Although the range of $p_0$ maintaining the good decoding performance of VNSA-IM-PGDBF is more restrictive than that of VNSA-PGDBF, the decoding performance of VNSA-IM-PGDBF for $p_0 \geq 0.6$ is always better than VNSA-PGDBF. Similar observations have been obtained for other QC-LDPC codes. The theoretical proof for this behavior is to be addressed as future work.

## IV. Synthesis results and decoding performance

The VNSA-based PGDBF decoder implementations are synthesized on 90nm CMOS technology using Synopsys tools. We compare them with the non-probabilistic GDBF, Adaptive Threshold BF (ATBF) [8] and MS decoder benchmark [9]. The MS implementation in [9] uses the same dv3R050N1296 test code while [8] considers a $(d_v, d_c) = (3, 6)$, $M = 504$, $N = 1008$ code. Since the synthesis values of GDBF and CSTS-PGDBF of [7] for post-layout on 90nm technology were not provided in that paper, we re-synthesize and provide them in table I. The synthesis results show that, VNSA helps reduce significantly the complexity of PGDBF implementations as expected. Indeed, VNSA-PGDBF requires only $89\%$ of the GDBF complexity to be efficiently implemented. The reduction is more significant in the case of VNSA-IM-PGDBF, which needs only $81.7\%$ of the GDBF hardware resources. Note that, the CSTS-PGDBF, $S = Z$ provides an equivalent decoding performance to VNSA-PGDBF while requiring $2\%$ additional hardware with respect to GDBF decoder. The maximum frequency of VNSA-IM-PGDBF is the highest, 400Mhz compared to 357Mhz of CSTS-PGDBF and 385Mhz of GDBF. Although the code considered in this work is longer than in [8], 1296 compared to 1008, our BF decoders are, however, less than a half of the ATBF complexity while having higher operating frequencies, $370 - 400$Mhz compared to 250Mhz. The VNSA-based PGDBF decoders have much lower complexity in comparison to MS for the same codeword length. MS decoder is around $0.72mm^2$ on $65nm$ technology or $1.38mm^2$ when being scaled to $90nm$ technology, which is 4.7 times higher than VNSA-IM-PGDBF. This MS implementation has a reported maximum frequency of 250Mhz. The VNSA-based PGDBF decoders provide a better throughput than those of the ATBF and MS thanks to the higher operating frequency. We compute the decoding throughput 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 $\alpha = 0.01$, VNSA-PGDBF obtains 99.3Gbps while VNSA-IM-PGDBF has 97.4Gbps, compared to 41.9Gbps and 25.2Gbps of MS and ATBF respectively. 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.

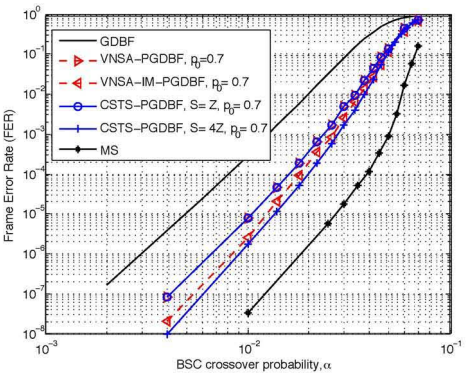



Figure 2. Error correction performance on the dv3R050N1296 QC-LDPC code.

Table I. The synthesis result comparisons.

| | AREA ($mm^2$) | $f_{max}$ (MHz) | $\alpha = 0.01$ | |
|---|---|---|---|---|
| | | | $It_{ave}$ | $\theta$ (Gbit/s) |
| **GDBF** | 0.360 (+0.0%) | 385 | 2.95 ($FER = 3e^{-4}$) | 169.1 |
| **CSTS-PGDBF** | 0.367 (+2.0%) | 357 | 5.15 ($FER = 7e^{-6}$) | 89.8 |
| **VNSA-PGDBF** | 0.320 (-11.1%) | 370 | 4.83 ($FER = 8e^{-6}$) | 99.3 |
| **VNSA-IM-PGDBF** | 0.294 (-18.3%) | 400 | 5.32 ($FER = 2.6e^{-6}$) | 97.4 |
| **ATBF** [8] | 0.63 | 250 | 25.2Gbps @ 10 iterations | |
| **MS** [9] | 0.72 | 250 | 1.29 ($FER = 1e^{-7}$) | 41.9 |

The simulated decoding performance of the decoders are shown in Fig. 2. VNSA-PGDBF provides an equivalent decoding performance of the CSTS-PGDBF $S = Z$. The simulation result re-confirms the superiority of VNSA-IM-PGDBF over the VNSA-PGDBF with around 1dB gain in FER ($\alpha = 0.01$). MS 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 VNSA. The VNSA uses the regular structure in connection networks of the QC-LDPC decoders to cyclic shift the messages without changing the computation results. Two implementations of PGDBF using VNSA are presented, the VNSA-PGDBF and the VNSA-IM-PGDBF. The simulation performance and ASIC synthesis results confirm that the VNSA-based PGDBF decoders have lower complexity, higher throughput and good performance with respect to other implementations in literature.

## Acknowledgment

This work was funded by the french ANR under grant number ANR-15-CE25-0006-01 (NAND project) and the Franco-Romanian (ANR-UEFISCDI) Joint Research Program (DIAMOND project).

978-1-5386-4881-0/18/$31.00 ©2018 IEEE

## REFERENCES


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

[2] 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.

[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] M. Ismail, I. Ahmed, , and J. Coon, "Low power decoding of ldpc codes," *ISRN Sensor Networks*, vol. 2013, no. 650740, 2013.

[9] T. T. Nguyen-Ly, T. Gupta, M. Pezzin, V. Savin, D. Declercq, and S. Cotofana, "Flexible, cost-efficient, high-throughput architecture for layered ldpc decoders with fully-parallel processing units," in *2016 Euromicro Conference on Digital System Design (DSD)*, Aug 2016, pp. 230–237.


978-1-5386-4881-0/18/$31.00 ©2018 IEEE