# Variable-Node-Shift Based Architecture for Probabilistic Gradient Descent Bit Flipping on QC-LDPC Codes

Khoa Le, David Declercq, *Senior Member, IEEE*, Fakhreddine Ghaffari, *Member, IEEE*, Lounis Kessal, *Member, IEEE*, Oana Boncalo, *Member, IEEE*, and Valentin Savin, *Member, IEEE*

***Abstract*— Probabilistic gradient descent bit-flipping (PGDBF) is a hard-decision decoder for low-density parity-check (LDPC) codes, which offers a significant improvement in error correction, approaching the performance of soft-information decoders on the binary symmetric channel. However, this outstanding performance is known to come with an augmentation of the decoder complexity, compared to the non-probabilistic gradient descent bit flipping (GDBF), becoming a drawback of this decoder. This paper presents a new approach to implementing PGDBF decoding for quasi-cyclic LDPC (QC-LDPC) codes, based on the so-called variable-node-shift architecture (VNSA). In VNSA-based PGDBF implementations, the regularity of QC-LDPC connection networks is used to cyclically shift the memory of the decoder, leading to the fact that, a variable node (VN) is processed by different computing units during the decoding process. With this modification, the probabilistic effects in VN operations can be produced by implementing different types of processing units, without requirement of a probabilistic signal generator. The VNSA is shown to further improve the decoding performance of the PGDBF, with respect to other hardware implementations reported in the literature, while reducing the complexity below that of the GDBF. The efficiency of the VNSA is proven by ASIC synthesis results and by decoding simulations.**

***Index Terms*— Low-density parity-check, quasi-cyclic, probabilistic gradient descent bit flipping, random generation, low complexity implementation, high throughput decoder.**

## I. Introduction

LOW-DENSITY Parity-Check (LDPC) codes have been increased research interest over the last years, due to their near-capacity error correction performance and their manageable complexity implementations [1]. Hence, they have been used in many practical applications, such as storage (hard drives, flash memories, *etc.*) [2]–[4], or wireless transmission standards such as IEEE 802.3an 10GBASE-T, IEEE 802.11n, 802.16a [5]–[7]. LDPC codes can be effectively decoded by either soft decision decoding algorithms, such as Belief Propagation (BP) or Min Sum (MS), providing very good error correction capability but the computation complexity is very high, or the hard-decision decoding algorithms, such as Bit Flipping (BF) or Gallager-A/B, which are very low in complexity but a weak performance is observed [1]. LDPC decoding algorithms follow an iterative manner where the messages are iteratively passed between two groups of nodes: Variable Nodes (VNs) and Check Nodes (CNs). The low complexity of hard-decision decoders comes from the fact that, only binary messages are iteratively passed between the VNs and CNs during the decoding process, which simplifies the connection network of the decoders. In addition, the computation units to process these binary messages, are very simple. This paper is concerned with the Bit Flipping (BF) hard decision decoder where, contrary to message passing hard decision decoders, both extrinsic and intrinsic information are passed between the computation nodes.

The conventional BF decoder, introduced by Gallager in 1963 [8], is a very low complexity decoder but being weak in performance, compared to soft-decision decoders. Many modifications have been introduced to fill this performance gap, such as Weighted BF (WBF) [9], Modified WBF (MWBF) [10], Improved MWBF (IMWBF) [11], especially the class of the Gradient Descent Bit Flipping (GDBF) [12] and its variants such as Probabilistic Gradient Descent Bit Flipping (PGDBF) [13], Noisy Gradient Descent Bit Flipping (NGDBF) [14]. The differences between these BF decoders can be summarized by the difference from the computation of the so-called *inversion function* (or *energy function*) and from the mechanism of choosing the VNs to flip at each iteration. In the conventional BF, at each iteration, the energy value of every VN is given by the number of unsatisfied neighboring CNs, and the VNs that have energy lower than a pre-defined threshold will be flipped. By this energy computation, the conventional BF considers the contribution of neighbor CNs to VN energy value equally. Several modifications such as Weighted BF (WBF) [9], Modified WBF (MWBF) [10], Improved MWBF (IMWBF) [11] *etc.*, have put

Manuscript received August 28, 2017; revised November 7, 2017; accepted November 19, 2017. This work was supported in part by French ANR through the NAND Project under Grant ANR-15-CE25-0006-01 and in part by the Franco-Romanian (ANR-UEFISCDI) Joint Research Program (DIAMOND Project). This paper was recommended by Associate Editor I. Kale. *(Corresponding author: Khoa Le.)*

K. Le, D. Declercq, F. Ghaffari, and L. Kessal are with ETIS, UMR8051, Université Paris Seine, Université de Cergy-Pontoise, ENSEA, CNRS, 95014 Cergy-Pontoise, France (e-mail: khoa.letrung@ensea.fr; declercq@ensea.fr; fakhreddine.ghaffari@ensea.fr; kessal@ensea.fr).

O. Boncalo is with the Computer Engineering Department, University Politehnica Timisoara, Timisoara 300006 Romania (e-mail: oana.boncalo@cs.upt.ro).

V. Savin is with CEA-LETI, MINATEC, 38054 Grenoble, France (e-mail: valentin.savin@cea.fr).

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.2017.2777802

weighting factors on these parity check values and made themselves differ from the weighting factor calculation. Many other interesting BF proposals, with the same spirit of modifying energy computation and/or flipping mechanism, can be found in [15]–[17] and combinations of several BF decoding principles into a single decoder were presented in [18], [19]. Although the above introduced decoders provide different levels of error correction improvement, they are still not comparable to the performance of the soft-decision decoders.

The Gradient Descent Bit Flipping decoder has been introduced by Wadayama *et al.* in [12] in which the authors considered the decoding process as an optimization process. In GDBF, an objective function is defined and a strategy to flip the VNs is derived to minimize this objective function. The GDBF offers a better decoding performance to all prior-introduced BF decoders while keeping the low-complexity property. A variant of GDBF, called Probabilistic GDBF (PGDBF), was introduced by Rasheed *et al.* in [13] by modifying GDBF to apply on Binary Symmetric Channel (BSC) and by incorporating the probabilistic feature in flipping the VNs. PGDBF follows precisely the decoding steps of GDBF and the difference comes from the fact that, the flipped VNs in GDBF are actually flipped with only probability $p_0 < 1$ in PGDBF. PGDBF obtains a large performance gain compared to GDBF, approaching the performance of soft-decision decoders [13].

The outstanding decoding performance of PGDBF comes along with the additional complexity, compared to GDBF, due to the additional probabilistic signal generator. Indeed, a straightforward FPGA implementation of PGDBF would multiply by 8 the number of registers and by 1.64 the number of Look-Up-Tables (LUTs) required to implement GDBF, as reported in [20]. Another optimized architecture of PGDBF was introduced in [21] where the random generator was significantly optimized. It is shown that PGDBF can be efficiently implemented with a moderate additional hardware resource ($5\% - 14\%$ hardware overhead on ASIC implementation). It can be seen in these PGDBF implementations that, the additional hardware cost is inevitable since a random generator is implemented on top of the GDBF decoder and they could be applied for any type of LDPC codes.

Quasi-Cyclic LDPC codes are a structural LDPC codes in which, due to their construction, the connections between VNs and CNs are highly regular [22]–[24], which enables the efficient hardware implementations while providing a comparable error correction performance to the random LDPC codes [25], [26]. Hence, QC-LDPC codes are adopted in many systems such as DVB-S2 and above-mentioned standards such as WLAN (IEEE 802.11n), WiMAX (IEEE 802.16e), 10-Gigabit-Ethernet (IEEE 802.3an) *etc*. Several interesting hardware implementations of BF decoders can be found in the literature such as the Adaptive Threshold Bit Flipping (ATBF) in [27], the NGDBF [14], the combined BF and Stochastic decoders in [28], *etc.* which are designed and tested on the additive white Gaussian noise (AWGN) channel. The recent introduced implementations of BF decoders on BSC can be found such as the Multi-BF in [29], the GDBF and PGDBF in [20], [21]. However, all of above BF implementations did not consider the structural property of QC-LDPC codes to reduce decoder complexity nor improve decoding performance.

We introduce in this work a particular hardware architecture, called Variable-Node Shift Architecture (VNSA), to implement PGDBF on QC-LDPC codes. VNSA could be also applied for other decoding algorithms. The VNSA exploits the regular structure of QC-LDPC codes to change the message flow, improving the dynamics of the decoder and from that, a VN is processed by different processing units in different iterations. By implementing different types of variable node processing units (VNU - to process the VN operations), the probabilistic behaviors are produced without using a probabilistic signal generator. The implemented processing units are simpler than those of the implemented PGDBF in the literature. The VNSA is shown to further improve the decoding performance of the implemented PGDBF decoder in the literature while reducing the complexity, being even lower than the deterministic GDBF.

The paper is organized as following. Section II introduces the notations related to LDPC codes and LDPC decoding algorithms. This section presents also the PGDBF decoding algorithm and reviews an optimized implementation previously in the literature. Section III first presents the principle of VNSA for QC-LDPC decoding. The advantages of VNSA are then discussed when different types of VNUs are implemented. In Section IV, the implementation of the PGDBF using VNSA (called VNSA-PGDBF) is introduced in which two types of VNU are implemented, preserving the probabilistic behavior while being simpler than those of the optimized implementation in [21]. A simplified version of VNSA-PGDBF is also introduced, called the imprecise VNSA-PGDBF (VNSA-IM-PGDBF). Synthesis results and decoding performance are presented in Section V. It is shown that VNSA-PGDBF preserves the outstanding decoding performance of PGDBF while the complexity is significantly reduced, even lower than the deterministic GDBF. The VNSA-IM-PGDBF is shown not only reduce the complexity, but also improve the decoding performance on comparison to the VNSA-PGDBF. Section VI concludes the paper.

## II. Preliminaries

### A. Notations

An LDPC code is defined by a sparse parity-check matrix $H$ $(M, N)$, $N > M$. Each row represents a parity check equation, computed by a CN, on the VNs represented by the columns. A QC-LDPC code is a specific construction method of $H$ in which each entry of a small *base matrix* $H_B$ $(n_r \times n_c)$ is replaced by either a circulant shift of a $Z \times Z$ diagonal matrix or an *all-zero* $Z \times Z$ matrix, ($N = n_c \times Z$, $M = n_r \times Z$). The columns of $H_B$ are called base columns and the rows are called base rows. In QC-LDPC code, $N$ VNs can be split into $n_c$ groups of $Z$ VNs corresponding to $n_c$ base columns of $H_B$ and similarly, $M$ CNs can also be grouped into $n_r$ groups of $Z$ CNs corresponding to $n_r$ base rows of $H_B$. We denote the $j$-th VN ($1 \leq j \leq Z$) belonging to the $i$-th base column ($1 \leq i \leq n_c$) of the base matrix $H_B$ as $v_{i,j}$, similarly, the $b$-th CN ($1 \leq b \leq Z$) belonging to the

$a$-th base row ($1 \le a \le n_r$) as $c_{a,b}$. The VN $v_{i,j}$ is checked by the CN $c_{a,b}$ if the entry $H(aZ+b, iZ+j) = 1$ and they are called neighbors. An QC-LDPC code can also be represented by a bipartite graph, called Tanner graph, comprising two groups of nodes, the CNs $c_{a,b}, 1 \le a \le n_r, 1 \le b \le Z$ and the VNs $v_{i,j}, 1 \le i \le n_c, 1 \le j \le Z$. In the Tanner graph, a VN $v_{i,j}$ connects to a CN $c_{a,b}$ when $H(aZ+b, iZ+j) = 1$. We denote the set of CNs connected to the VN $v_{i,j}$ as $\mathcal{N}(v_{i,j})$, $|\mathcal{N}(v_{i,j})|$ is called VN degree and similarly, $\mathcal{N}(c_{a,b})$ is the set of VNs connected to CN $c_{a,b}$, $|\mathcal{N}(c_{a,b})|$ is called CN degree. This work focuses on the regular QC-LDPC codes, *i.e.* $\left|\mathcal{N}(v_{i,j})\right| = d_v$ and $\left|\mathcal{N}(c_{a,b})\right| = d_c$ $\forall i, j, a, b$. A vector $\mathbf{x} = (x_{i,j} | 1 \le i \le n_c, 1 \le j \le Z) = (x_{1,1}, x_{1,2}, \ldots, x_{1,Z-1}, x_{1,Z}, x_{2,1}, x_{2,2}, \ldots, x_{n_c,Z-1}, x_{n_c,Z}) \in \{0,1\}^N$ is called a codeword if and only if $H\mathbf{x}^T = 0$. $\mathbf{x}$ is sent through a transmission channel and we denote by $\mathbf{y} = (y_{i,j} | 1 \le i \le n_c, 1 \le j \le Z) = (y_{1,1}, y_{1,2}, \ldots, y_{1,Z-1}, y_{1,Z}, y_{2,1}, y_{2,2}, \ldots, y_{n_c,Z-1}, y_{n_c,Z})$ the output of this channel. The decoders in this paper are restricted to work on the BSC where each bit $x_{i,j}$ is flipped with a probability $\alpha$, called channel crossover probability, when being transmitted, *i.e.*, $\Pr(y_{i,j} = x_{i,j}) = 1 - \alpha$ and $\Pr(y_{i,j} = 1 - x_{i,j}) = \alpha$, $1 \le i \le n_c, 1 \le j \le Z$.

### *B. The PGDBF Decoding Algorithm and Its Implementation*

PGDBF decoding process is an A Posteriori Probability (APP) message passing decoder which iteratively updates the VN values via the decoding iterations. We denote $c_{a,b}$ (*resp.* $v_{i,j}$) as CN (*resp.* VN) itself while $c_{a,b}^{(k)}$ (*resp.* $v_{i,j}^{(k)}$) denotes the value of the CN (*resp.* VN) at iteration $k$. We also denote the VN output vector $\mathbf{v}^{(k)} = (\mathbf{v}_1^{(k)}, \mathbf{v}_2^{(k)}, \ldots, \mathbf{v}_{n_c}^{(k)})$ where $\mathbf{v}_i^{(k)} = (v_{i,1}^{(k)}, v_{i,2}^{(k)}, \ldots, v_{i,Z}^{(k)})$, $1 \le i \le n_c$ and CN output vector $\mathbf{c}^{(k)} = (\mathbf{c}_1^{(k)}, \mathbf{c}_2^{(k)}, \ldots, \mathbf{c}_{n_r}^{(k)})$ where $\mathbf{c}_a^{(k)} = (c_{a,1}^{(k)}, c_{a,2}^{(k)}, \ldots, c_{a,Z}^{(k)})$, $1 \le a \le n_r$.

The CN computation in PGDBF is a parity check on all of its neighbor VNs and it is formulated as in Eq. (1) where $\oplus$ is the bit-wise Exclusive-OR (XOR) operation. The PGDBF decoding process is stopped when all CNs equations are satisfied, *i.e.* $c_{a,b}^{(k)} = 0, \forall a, b$, or when $k$ reaches the maximum number of iterations, denoted by $It_{max}$.

$$c_{a,b}^{(k)} = \bigoplus_{v_{i,j} \in \mathcal{N}(c_{a,b})} v_{i,j}^{(k)} \tag{1}$$

$$E_{i,j}^{(k)} = v_{i,j}^{(k)} \oplus y_{i,j} + \sum_{c_{a,b} \in \mathcal{N}(v_{i,j})} c_{a,b}^{(k)} \tag{2}$$

Each VN updates its value by the following steps. First, the VN evaluates its energy function using Eq. (2). An 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). Second, the VN $v_{i,j}$ is flipped if and only if the two following conditions are satisfied concurrently: its energy value is equal to the maximum energy ($E_{i,j}^{(k)} = E_{max}^{(k)}$) and the value of $R_{i,j}^{(k)} = 1$ where $R_{i,j}^{(k)}$ is a random bit in a random sequence $R$ of length $N$ at the $k$-th iteration ($R^{(k)}$), $R^{(k)} = \{R_{i,j}^{(k)} | 1 \le i \le n_c, 1 \le j \le Z\}$, ($\Pr(R_{i,j}^{(k)} = 1) = p_0$, $\Pr(R_{i,j}^{(k)} = 0) = 1 - p_0$). The maximum energy is computed by a module called Maximum Finder (MF) among $N$ energy values. The updated values of VNs are sent to the next decoding iteration. The PGDBF algorithm is described in Algorithm 1.

**Algorithm 1** Probabilistic Gradient Descent Bit Flipping

**Initialization** $k = 0$, $v_{i,j}^{(0)} \leftarrow y_{i,j}, 1 \le i \le n_c, 1 \le j \le Z$.
$\mathbf{s} = H\mathbf{v}^{(0)^T} \bmod 2$
**while** $\mathbf{s} \neq \mathbf{0}$ *and* $k \le It_{max}$ **do**
  **Compute** $c_{a,b}^{(k)}$, $1 \le a \le n_r, 1 \le b \le Z$, using Eq. (1).
  **Compute** $E_{i,j}^{(k)}$, $1 \le i \le n_c, 1 \le j \le Z$, using Eq. (2).
  **Search** $E_{max}^{(k)} = max_{i,j}(E_{i,j}^{(k)})$,
  **Generate** $R_{i,j}^{(k)}$, $1 \le i \le n_c, 1 \le j \le Z$, from $\mathcal{B}(p_0)$.
  **for** $1 \le i \le n_c, 1 \le j \le Z$ **do**
    **if** $E_{i,j}^{(k)} = E_{max}^{(k)}$ **and** $R_{i,j}^{(k)} = 1$ **then**
      $v_{i,j}^{(k+1)} = v_{i,j}^{(k)} \oplus 1$
    **end if**
  **end for**
  $\mathbf{s} = H\mathbf{v}^{(k+1)^T} \bmod 2$
  $k = k + 1$
**end while**
**Output:** $\mathbf{v}^{(k)}$

An optimized implementation of PGDBF is presented in [21] and recalled in Fig. 1. To the best of our knowledge, [21] is the only work in the literature presenting the ASIC implementation of a PGDBF decoder. In this implementation, two registers are used in each VNU, to store the channel output value, $y_{i,j}$ and the value of the VN at the iteration $k$, $v_{i,j}^{(k)}$. The summation block computes the energy value and the equality comparison block indicates whether or not the energy value is equal to the maximum. The AND gate incorporates the random signal $R_{i,j}^{(k)}$ with the equality comparison result. The XOR2 is used to flip the bit ($v_{i,j}^{(k)}$) whenever the output of AND-gate is 1, and this value is stored back to the register as the value for the next iteration, $v_{i,j}^{(k+1)}$.

One important issue in the PGDBF implementation is the probabilistic signal generator which is also optimized in [21], called Cyclically Shift Truncated Sequence (CSTS) method (Fig. 1). In CSTS probabilistic signal generator, a truncated sequence of registers, $R_t$, with size of $|R_t| = S < N$ is allocated and stores the randomly generated bits. $R_t$ is then duplicated to produce $R$ (of length $N$). This short register sequence is cyclically shifted to make $R$ different from one iteration to another (Fig. 1). It can be seen that, $R_t$ drives the VNU to have 2 different behaviors corresponding to 2 possibilities of the probabilistic signals ($R_{i,j}^{(k)} = 0$ and $R_{i,j}^{(k)} = 1$). In fact, during the decoding process, a VNU may produce different output values even when its receives the same inputs as another iteration, depending on the values of $R_{i,j}^{(k)}$.

It is noted that, the implementation of PGDBF requires an inevitable additional hardware compared to GDBF due to the fact that, on top of the GDBF, the random signal generating module and the AND gates (the dashed red parts in Fig. 1)

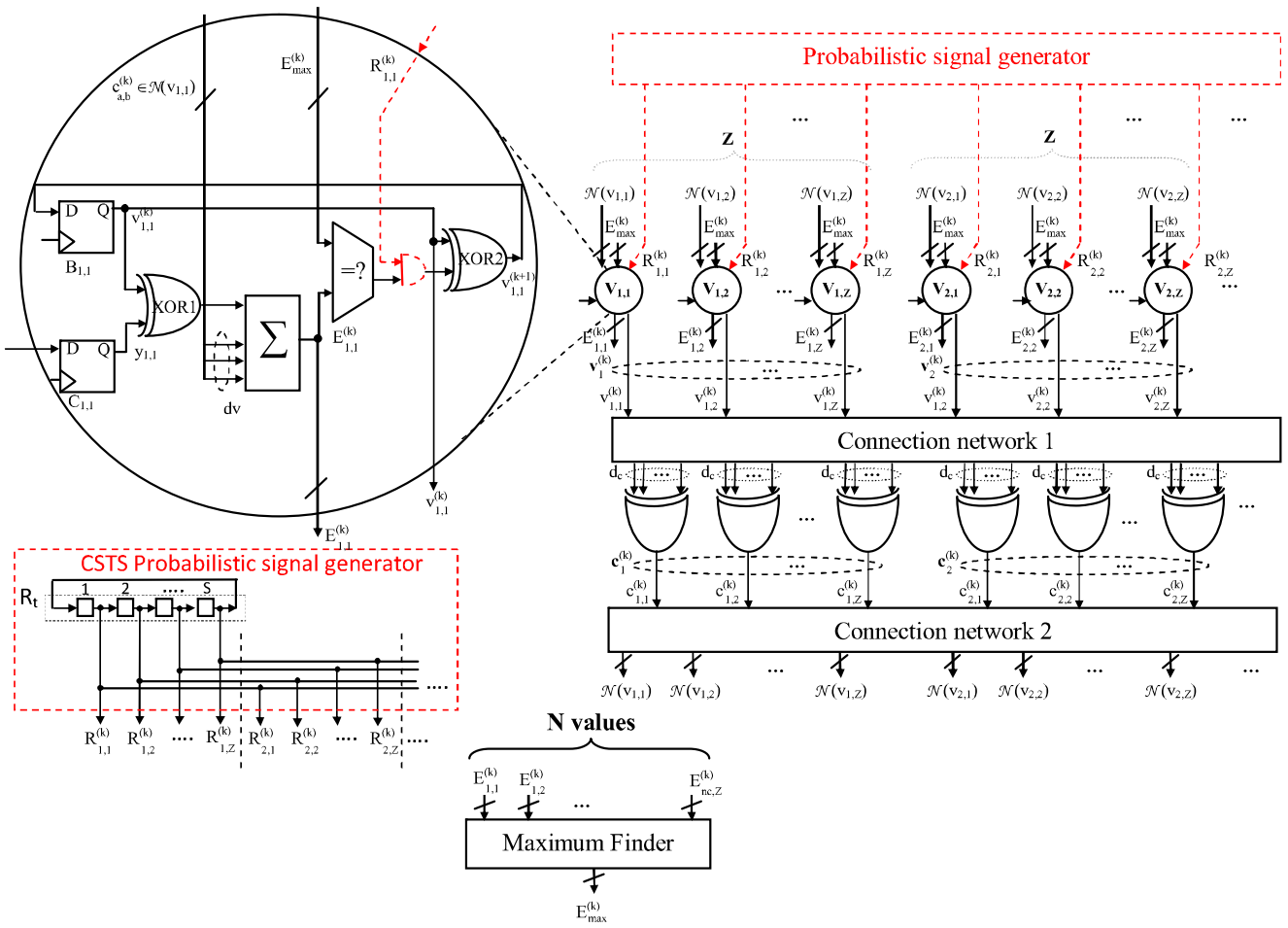



Fig. 1. State of the art implementation of the PGDBF decoder.

are required. It should also be noted that, this implementation does not exploit the QC-LDPC structure.

## III. The Variable-Node-Shift Architecture for QC-LDPC Decoding

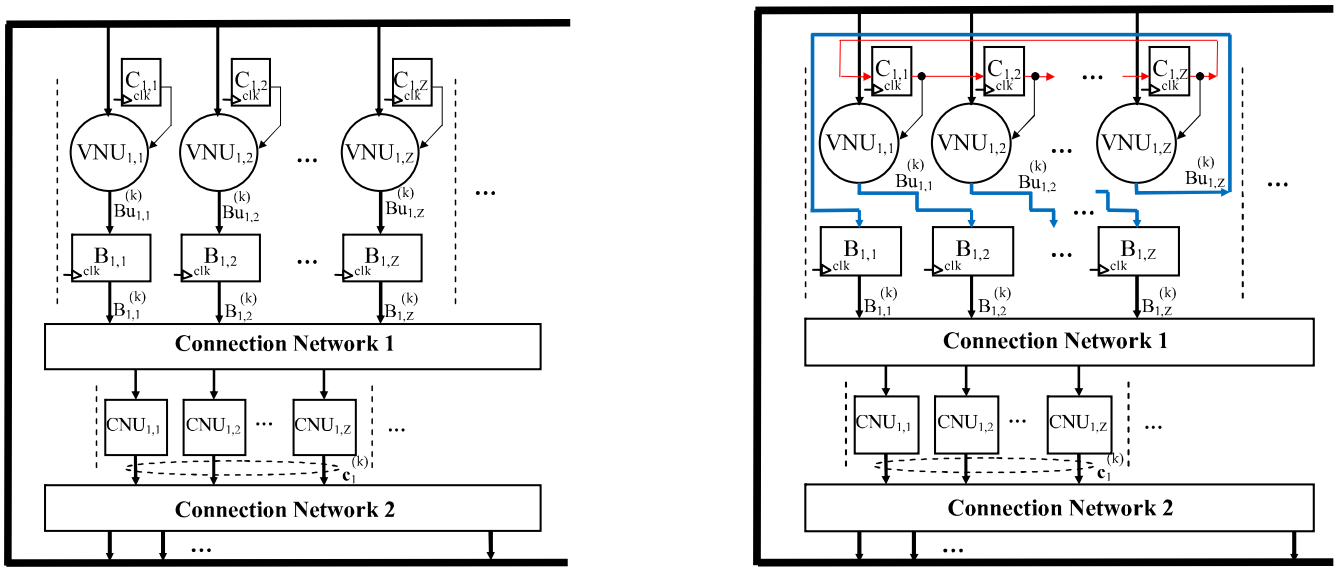
In this section, we present first the generic architecture for LDPC decoding. On top of this generic architecture, the principle and advantages of VNSA are then introduced and discussed. Although the VNSA can be applied to both flooding and layered decoding scheduling [1], we limit the discussion in this section only on the flooding implementation.

### A. The Architecture of LDPC Decoding

LDPC decoders are generally implemented by the connection network blocks connecting two groups of processing units, the variable node units (VNUs - to process the VN operations) and the check node units (CNUs - to process the CN operations). The computed messages are iteratively passed between the VNUs and CNUs through these connection networks during the decoding process.

The generic architecture of LDPC decoders is presented in Fig. 2(a) in which the circles represent the VNUs and squares represent the CNUs. The memory elements are signified by the clock signals *(clk)*. The memory elements $B$ and $C$ are allocated to store the intermediate messages during the decoding process and the channel output values, respectively. We index $B$ and $C$ to follow the indexing of VNs and CNs in QC-LDPC codes above. Also, the superscript $(k)$ indicates the values at iteration $k$. The size of these memory elements depends on the decoding algorithm as well as the channel model. For example, in the hard decision decoding algorithms such as GDBF, PGDBF on the BSC channel [20], [21], $B$ and $C$ in each VNU are two 1-bit registers. For the soft decision decoding algorithms such as MS on the AWGN channel [30], [31], $C$ is a register bank of size $q$ where $q$ is the quantization length [31], to store the channel soft information while $B$ contains multiple register banks of size $q$ to store extrinsic messages.

The operations of LDPC decoding algorithms on this generic architecture are briefly described as following. At the decoding initialization phase, $B$ and $C$ are initialized by the channel values. The decoder starts to decode and an iteration can be divided into 2 steps. In the first step, the messages stored in $B$ are transmitted to the CNUs through the connection network 1. The new messages are produced by these CNUs. In the second step, the messages from CNUs are propagated through the connection network 2 to the




(a) (b)

Fig. 2. The QC-LDPC decoding architectures. (a) The generic architecture. (b) The proposed Variable-Node Shift Architecture (VNSA).

corresponding VNUs in which the new messages, denoted by $Bu$, are produced, based on these CNU messages and the channel value from $C$ memory. These updated messages $Bu$ are stored into $B$ memory when clock event occurs such that $B_{i,j}^{(k+1)} = Bu_{i,j}^{(k)}$. The hard decision is made in each VNU and it is sent to the syndrome check module (this module is not shown in the figure). The decoder stops decoding when all the parity checks are satisfied, otherwise, a new decoding iteration will start.

The main difference between QC-LDPC and random LDPC implementations is at the connection networks. Due to the random constructing process, the connection networks of the random LDPC decoders do not follow any specific structure. On the contrary, those of the QC-LDPC decoders are very constructive. We illustrate in Fig. 3 the connection networks conveying the messages from VNUs to CNUs (connection network 1) and CNUs to VNUs (connection network 2), when implementing the first row of a QC-LDPC example base matrix $H_B$ (in this base matrix example, the white elements of $H_B$ are replaced by all-zero sub-matrices while the shaded elements are replaced by the cyclic shift versions of the main diagonal sub-matrix). It can be seen that, the message flow, conveyed by connection network 1, follows 2 steps. First, the messages are chosen at the level of group of $Z$, corresponding to $Z$ messages from the VNUs in the base columns, *i.e.*, in this example the message groups from base columns 2, 5 and 7 are chosen. Second, the messages in each group are locally cyclic-shifted to form the inputs of CNUs. In the connection network 2, the group of $Z$ computed messages are cyclic-shifted firstly and then propagated to the VNUs corresponding to the base columns as at the input of connection network 1.

It should be noted that, in the generic LDPC decoding architecture, each VN (and also each CN) is processed by a dedicated VNU (CNU) during the decoding process,

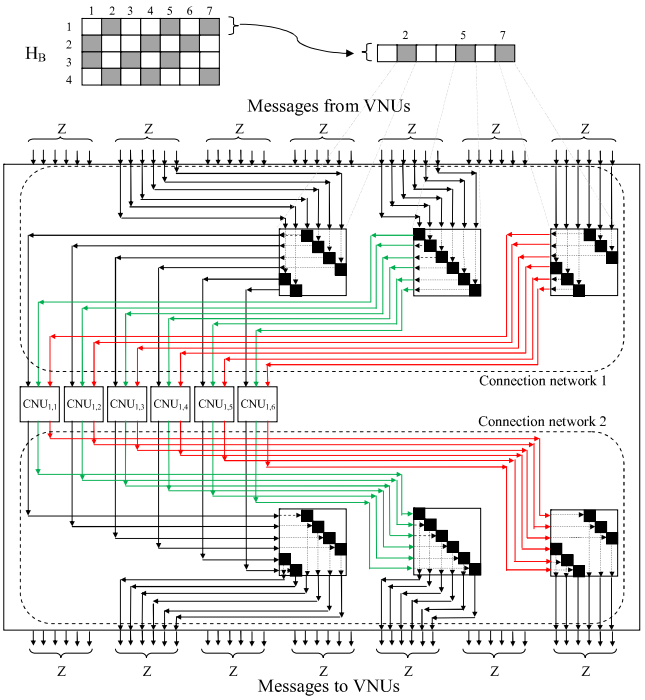



Fig. 3. The connection networks when implementing the first row of the example base matrix $H_B$. Note that, the circulant matrices aim to illustrate the cyclic-shift of the messages and they are the hard-connection wires in the implementation.

*i.e.* $VN_{i,j}$ is always processed by the $VNU_{i,j}$. It is due to the fact that the updated messages, $Bu_{i,j}$, are stored in the same (indexing) memory location $B_{i,j}$ and for the next iterations, $B_{i,j}$ are conveyed to the implemented CNUs and VNUs by the fixed, implemented connection networks. This property of the generic architecture shows less flexibility when the VNs (and/or CNs) require different behaviors over iterations, *e.g.* VNs in PGDBF have 2 different behaviors depending

on the probabilistic signal values. In such a case, the VNUs (and CNUs) of the generic LDPC decoding architecture require unavoidably an additional hardware to perform the additional behaviors.

### *B. The Variable-Node-Shift Architecture for QC-LDPC Decoding*

The VNSA approach is to change the message flow by changing the message storing location from one iteration to another. By doing that, a VN (CN) can be processed by different VNUs (CNUs) during the decoding process. As described in Fig. 2(b), the major change of VNSA compared to the generic architecture, is that the updated messages $Bu$ of a VNU are stored in the memory element $B$ of the consecutive VNU (see definition 1) instead of its own memory. The connection doing $B_{i,j}^{(k+1)} = Bu_{i,j}^{(k)}, 1 \le i \le n_c, 1 \le j \le Z$ in the generic architecture, are replaced by another connection doing $B_{i,(j\%Z+1)}^{(k+1)} = Bu_{i,j}^{(k)}$, $1 \le i \le n_c, 1 \le j \le Z$, *i.e.* after each iteration, all the messages of the VNs of each base column are cyclically shifted by a factor of 1. The same modification is applied on the channel value memory $C$, *i.e.* the memory sequence $C$ is also cyclically shifted after each iteration, $C_{i,(j\%Z+1)}^{(k+1)} = C_{i,j}^{(k)}, 1 \le i \le n_c, 1 \le j \le Z$. In general, one can apply by any value of $\ell$, $1 \le \ell < Z$ in cyclic shifting the $B$ and $C$, keeping the same property of VNSA, such that $B_{i,(j\%Z+\ell)}^{(k+1)} = Bu_{i,j}^{(k)}$ and $C_{i,(j\%Z+\ell)}^{(k+1)} = C_{i,j}^{(k)}$. Also, it could be a left cyclic shift, such that $B_{i,(j\%Z-\ell)}^{(k+1)} = Bu_{i,j}^{(k)}$ and $C_{i,(j\%Z-\ell)}^{(k+1)} = C_{i,j}^{(k)}$. Without loss of generality, we limit $\ell = 1$ and right cyclic shift in this work.

*Definition 1:* Two VNUs $v_{i_1,j_1}$ and $v_{i_2,j_2}$, $1 \le i_1, i_2 \le n_c, 1 \le j_1, j_2 \le Z$ (or two CNUs $c_{a_1,b_1}$ and $c_{a_2,b_2}$, $1 \le a_1, a_2 \le n_r, 1 \le b_1, b_2 \le Z$), are called to be consecutive if they are in the same column (row) of the base matrix $H_B$, $i_1 = i_2$ ($a_1 = a_2$), and are in the positions $j_1$, $j_2$ ($b_1$, $b_2$) such that $j_2 = j_1\%Z+1$ ($b_2 = b_1\%Z+1$) where % is the modulus operation.

In VNSA, the VNs and CNs are processed by different VNUs and CNUs while providing the decoding results identically to those of the generic implementation. This can be clarified by considering the case of two consecutive iterations. In such a case, a VN (CN) of the generic architecture is always operated in one (fixed) VNU (CNU) while in VNSA, it is operated in 2 consecutive VNUs (CNUs). It is possible thanks to the structure of VNSA. Indeed, at the end of the first iteration, the messages are stored in such a way that all messages from VNs belonging to a base column, are cyclically shifted when being read at the beginning of the second iteration. Because of that, the connection network 1 conveys a VN message to the input of the consecutive CNU, instead of the input of the same CNU in the previous iteration. Since the cyclic shift is applied for all VNUs messages of all base columns, the consecutive CNU receives all messages which are supposed to be received by the CNU of previous iteration (in case there is no message cyclic shift). In general, the CNUs (and VNUs) produce identical results given that inputs are identical. The CNU computed results are, therefore, identical to the results in the generic architecture. Note further that, due to the cyclic shift of the input, the CNU output vector, $\mathbf{c_a}, 1 \le a \le n_r$, are cyclically shifted compared to the results of generic architecture.

For VN computation, at the outputs of CNUs, the hard wires in connection network 2 are gathered to form groups of $Z$, conveying $Z$ messages of $Z$ CNUs in the base rows to $Z$ VNUs of the base columns (see Fig. 3). Since the connection network 2 is the fixed hard wires while the CNUs computed messages, $\mathbf{c_a}$, are cyclically shifted, the consecutive VNU will receive all messages which are supposed to be received by VNU of previous iteration (in case there is no message cyclic shift). The consecutive VNU also receives the channel values of the corresponding VN because the channel memories, $C$, are also cyclically shifted at the end of the first iteration. The VN computation results are therefore similar to those of the generic implementation, and differ only from the computing VNUs.

The advantages of the VNSA-based architecture become apparent when the decoding algorithm may use different operations to process the same VN (or CN) at different decoding iterations. PGDBF is an example of such a decoding algorithm. Due to the probabilistic signal input, a VNU may behave in 2 different ways, that is, it may have 2 different results in 2 iterations, even if the it has the same input messages in these 2 iterations. The advantage of applying VNSA in term of decoding complexity comes from the fact that it helps distribute the different behaviors to different VNUs instead of implementing all of them in each and every VNU. In this case, the complexity of the whole system can be significantly reduced. The following example gives more clarifications on the complexity reduction. We consider the case where each VNU may have 2 different behaviors corresponding to the fact that it performs one of 2 different functions (denoted by $f_1$ and $f_2$) in each iteration. In the generic implementation, 2 functions are implemented in every VNU and a module is implemented to be in charge of choosing the function to be executed in each iteration ($f_1$ is chosen with probability $p_0$ and $f_2$ is with probability $1-p_0$), *e.g.*, in PGDBF, this module is the probabilistic signal generator. In implementing this type of decoder by VNSA, 2 types of VNUs are implemented, each one performing only one function, either $f_1$ or $f_2$. A fraction of $p_0$ of the VNUs are of type-1 (implementing function $f_1$), while a fraction of $(1 - p_0)$ are of type-2 (implementing function $f_2$). We denote the complexity of generic VNU as $\mathcal{C}_{12}$ while $\mathcal{C}_1$ ($\mathcal{C}_2$) is the complexity when only function $f_1$ ($f_2$) is implemented in each VNU. The complexity of VNUs of the VNSA implementation ($\mathcal{C}_{VNSA}$) compared to those of the generic implementation ($\mathcal{C}_{CONV}$) is expressed by the complexity efficiency $\xi$ as following:

$$\xi = \frac{\mathcal{C}_{VNSA}}{\mathcal{C}_{CONV}} = \frac{p_0\mathcal{C}_1 + (1-p_0)\mathcal{C}_2}{\mathcal{C}_{12}} \tag{3}$$

Due to the hardware reusing when implementing 2 functions in a VNU, $\mathcal{C}_{12} \le \mathcal{C}_1 + \mathcal{C}_2$ then

$$\xi \ge \frac{p_0\mathcal{C}_1 + (1-p_0)\mathcal{C}_2}{\mathcal{C}_1 + \mathcal{C}_2} = p_0 + \frac{1-2p_0}{1+\frac{\mathcal{C}_1}{\mathcal{C}_2}} \tag{4}$$

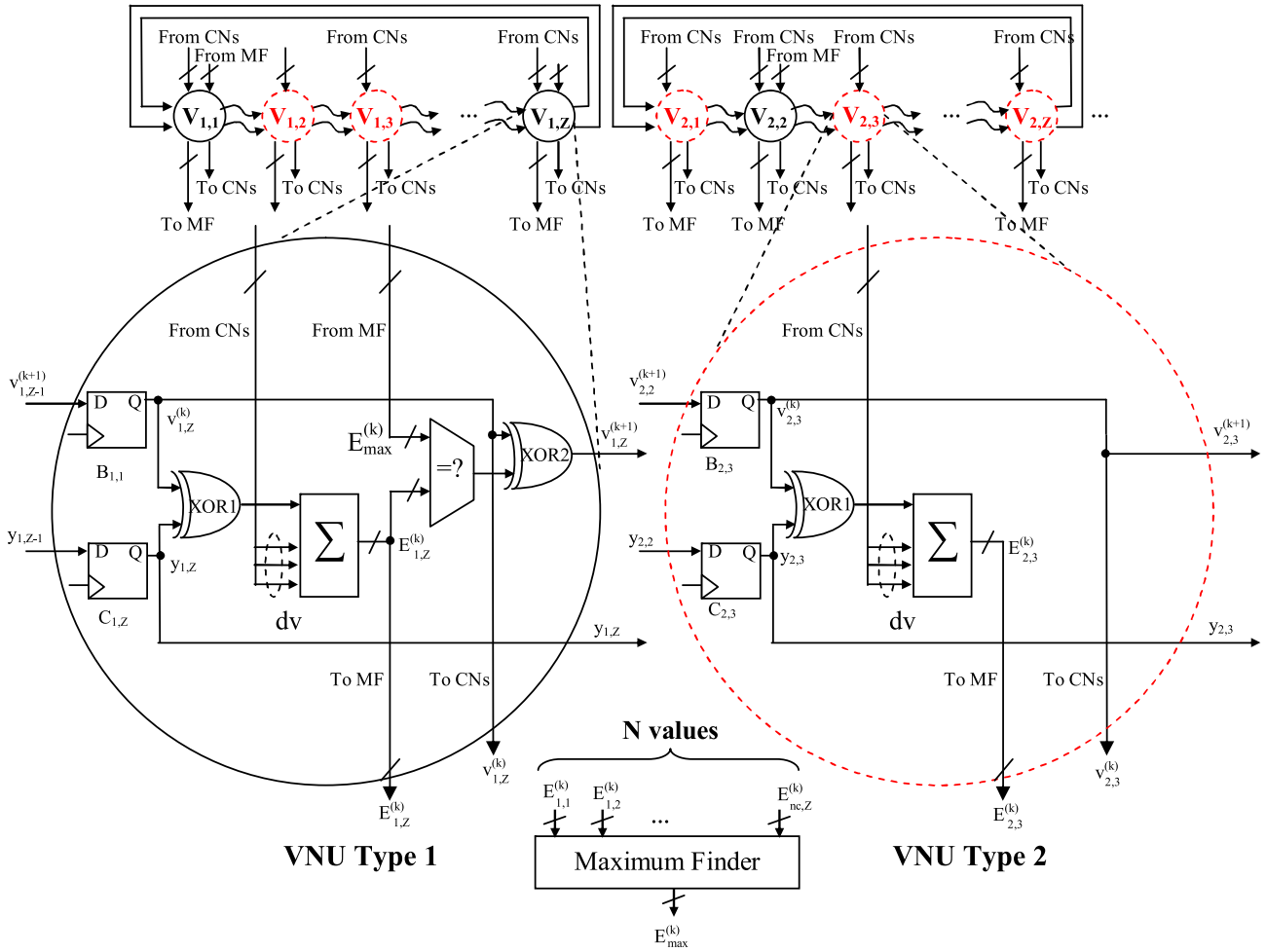



Fig. 4. The implementation of VNSA-PGDBF decoder.

It can be seen that the complexity efficiency $\xi$ generally depends on the complexity of the functions $f_1$ and $f_2$ and could be controlled by $p_0$. For example, when VNSA is applied to implement the PGDBF in the next section, an extreme case is shown where $\mathcal{C}_1 \gg \mathcal{C}_2$ and $p_0 = 0.7$ and from that, $\xi \geq 0.7$. In this case, applying VNSA helps reduce up to 30% the hardware resources in implementing the VNUs.

It should be noted further that in VNSA, the allocation of the VNUs decides the functions and their executing order with which a VN is processed during decoding process. By allocating $p_0 Z$ VNUs type 1 and $(1-p_0)Z$ VNUs type 2 in $Z$ VNUs of each base column, a VN is, by construction, processed by $f_1$ with ratio $p_0$ and $f_2$ with ratio $1-p_0$ after $Z$ iterations. The VNSA has, therefore, lower complexity than the generic implementation since the latter requires an additional module to choose the executing function in each iteration. For example, this additional module in PGDBF is the probabilistic signal generator and it requires a significant hardware overhead as shown in [20], [21]. More interesting, different functions $f_1$ and $f_2$ represent different decoding algorithms. Therefore, VNSA could be easily applied for different types of LDPC decoding algorithms.

## IV. The Implementation of Probabilistic Gradient Descent Bit Flipping Decoder Using Variable Node Shift Architecture

### A. Implementation of PGDBF With VNSA

In this section, we present the implementation of PGDBF using the VNSA, denoted as the VNSA-PGDBF. In VNSA-PGDBF, the message cyclically shift principle of VNSA are applied and 2 types of VNU are introduced. The 2 VNU types in VNSA-PGDBF operate similarly to the VNU in the PGDBF implementation in Section II (called conventional VNU) in which the random input is correspondingly 1 and 0. The architecture of VNSA-PGDBF is presented in Fig. 4.

The type-1 VNU of the VNSA-PGDBF, illustrated by the black solid circles in Fig. 4, mimics the operation of the conventional VNU when the random signal is equal to 1. In this case, the AND gate in the conventional VNU (Fig. 1) passes the equality comparator output directly to the XOR2 input since $X$ **AND** $1 = X$ and the VN value of the next iteration, $v_{i,j}^{(k+1)}$, is computed by $v_{i,j}^{(k+1)} = v_{i,j}^{(k)}$ **XOR** $\mathbb{1}(E_{i,j}^{(k)} = E_{max}^{(k)})$ where $\mathbb{1}(.)$ denotes the indicator function. The type-1 VNU is designed by simply removing the AND gate in the conventional VNU, keeping the same computation behaviors.

It can be seen that, this type-1 VNU is the VNU of GDBF decoder since the flip of the VN (operated by the XOR2) depends only on the equality between this VN energy and the maximum energy value.

The type-2 VNU of the VNSA-PGDBF, illustrated by the red dashed circles in Fig. 4, is designed by reproducing the operation of the conventional VNU when the random signal equals to 0. In the conventional VNU (Fig. 1), when the random input is 0, the AND gate produces 0 at its output regardless the equality comparison result. The VN value for the next iteration, $v_{i,j}^{(k+1)}$, is the one of current iteration, $v_{i,j}^{(k)}$, since $v_{i,j}^{(k+1)} = v_{i,j}^{(k)}$ **XOR** $0 = v_{i,j}^{(k)}$. The type-2 VNU is designed by simply propagating directly the current value $v_{i,j}^{(k)}$ to the output and removing the XOR2 gate in the conventional VNU. More specially, the AND gate and equality comparator can also be eliminated without affecting to the VNU operation. It can be seen that, the type-2 VNU is much simpler than the VNU of GDBF thanks to the elimination of the computation logic.

Although several parts of the conventional VNU are eliminated, leading the VNUs of VNSA-PGDBF to be simpler, the concept of PGDBF decoder is still fulfilled. It is highly interesting since there is no random generator needed. Indeed, it is noted that, the operation of VNSA-PGDBF is identical to the version of Cyclically Shift Truncated Sequence PGDBF (CSTS-PGDBF) decoder in Section II with the truncated sequence $R_t$, $|R_t| = S = Z$ (Fig. 1). 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 this VNU sequence, each VN will see the VNUs types (type-1 and type-2) with the same order of having bit 1 and bit 0 at the random input (in conventional VNU) when cyclically shifting $R_t$. The VNSA-PGDBF could be more advantageous in decoding performance since it has the possibility to distribute the type-1 and type-2 VNUs randomly in $Z$ VNUs of each base column and differently from other base columns. This is illustrated by the decoding performance gain of VNSA-PGDBF over CSTS-PGDBF, $S = Z$ in Section V. For the PGDBF implemented in this paper, we also restrict to have $p_0 Z$ VNUs type 1 and $(1 - p_0)Z$ VNUs type 2 in each base column.

### *B. An Imprecise Implementation of PGDBF With VNSA*

VNSA-PGDBF is shown to have lower complexity than the conventional PGDBF implementation thanks to the simpler VNUs and the elimination of the probabilistic signal generator. We further simplify the VNSA-PGDBF by replacing the type-2 VNU by other VNU type (type-3) as described in Fig. 6 and refer this PGDBF implementation as VNSA-IM-PGDBF. The type-3 VNU behaves similarly to the VNU in conventional PGDBF having 0 at the random input. The speciality comes from the fact that, this VNU type does not compute the energy value and all computing circuitry is eliminated. The behind idea comes from the operation of this type of VNU where it keeps the current value, $v_{i,j}^{(k)}$, for the next iteration regardless the maximum energy value and from that, its energy value may not need to join to the maximum energy finding. This helps further simplify the hardware complexity of VNSA-PGDBF decoder.

The low complexity of VNSA-IM-PGDBF is obtained due to the simplified type-3 VNU and the simpler MF. Indeed, it can be seen that, the type-3 VNU does not cost any hardware resources except the memory elements, ($C_{type-3} \ll C_{type-1}$). The hardware saving, therefore, depends on the number of type-3 VNU in $N$ implemented VNUs, which is controlled by the value of $p_0$. The MF in VNSA-IM-PGDBF needs to find the maximum energy in the list of $p_0 N$ input values instead of $N$. It is obvious that, the hardware cost of this sorting module is reduced when decreasing the number of inputs [32]. We apply the same technique in [21] to implement the MF for VNSA-based decoders in this paper.

A provable issue of VNSA-IM-PGDBF is that, the maximum energy value found in some iterations may not be the true maximum as in the precise PGDBF decoder. This comes from the fact that, this maximum is found in a shorter list of candidates ($p_0 N$ instead of $N$) and the impreciseness appears when only the VNs, which are being processed on type-3 VNU, have the maximum energy. In order to evaluate the effect of this impreciseness on the decoding performance, we have conducted a statistical analysis on the error correction performance (the Frame Error Rate - FER) as a function of $p_0$ and show the results in Fig. 5. The test codes used in this papers are with the parameters $(d_v, d_c) = (3, 6)$, $Z = 54$, $M = 648$, $N = 1296$, code rate $R = 0.5$, denoted as dv3R050N1296 and $(d_v, d_c) = (4, 8)$, $Z = 54$, $M = 648$, $N = 1296$, code rate $R = 0.5$, denoted as dv4R050N1296. It is very interesting that, VNSA-IM-PGDBF outperforms the VNSA-PGDBF despite the impreciseness introduced by the MF. It can be seen that, although the range of $p_0$ maintaining the good decoding performance of VNSA-IM-PGDBF is more selective than the one of VNSA-PGDBF, the decoding performance of VNSA-IM-PGDBF on the range of $p_0 \geq 0.6$ is always better than VNSA-PGDBF on the dv3R050N1296 code (Fig. 5(a)). For the dv4R050N1296 code (Fig. 5(b)), this performance gain of VNSA-IM-PGDBF over VNSA-PGDBF is even more significant (at $p_0 = 0.7$, around 1dB FER gain) and it is true on the wider range of $p_0$. Basing on the statistics, $p_0 \approx 0.7$ for the dv3R050N1296 code is used in the next section to maximize the decoding performance gain and hardware reduction.

Although the theoretical explanation of the VNSA-IM-PGDBF superiority over VNSA-PGDBF requires further study and it is an interesting topic for future work, the fact that using "untrue" maximum value to improve the decoding performance is inline with the approach of the decoder-dynamic shift PGDBF (DDS-PGDBF) [33] in literature. The authors in [33] proposed to use the threshold to flip the VNs which is the maximum energy of the previous iteration instead of the current one. Despite the untrue maximum value threshold used, DDS-PGDBF surpasses the soft decision decoders in performance and approaches the Maximum Likelihood Decoding at the cost of a large number of iterations. The VNSA-IM-PGDBF, with the same spirit, uses the imprecise threshold to flip the VNs and coherently provides an improvement in decoding performance even at acceptable number of iterations.

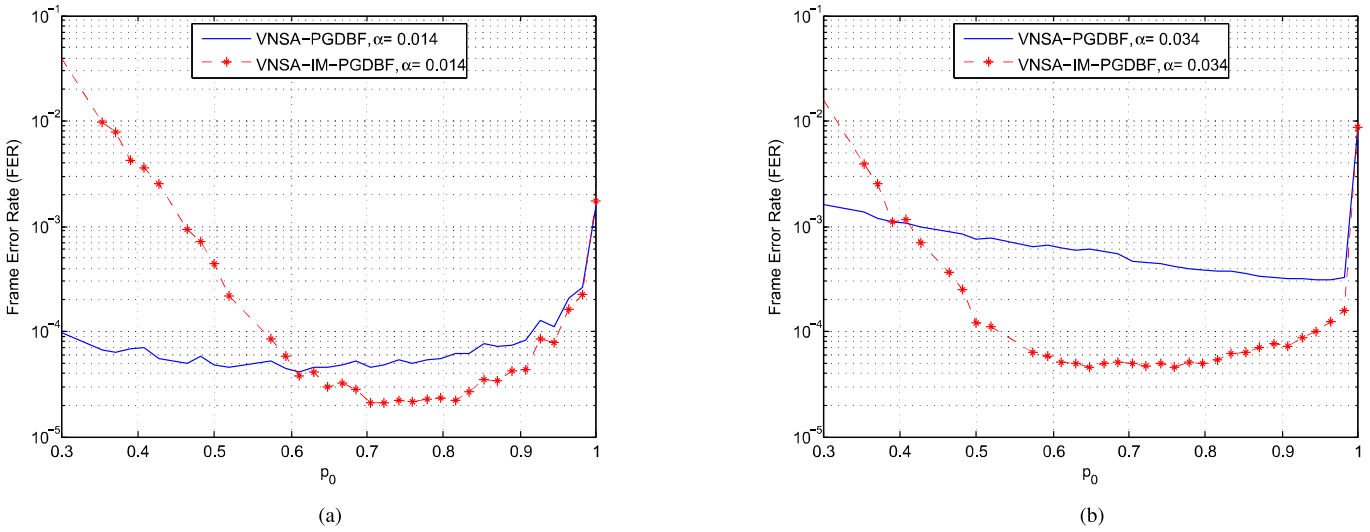



Fig. 5. The statistics on the decoding performance of VNSA-based PGDBF decoders as the function of $p_0$. (a) dv3R050N1296 QC-LDPC code. (b) dv4R050N1296 QC-LDPC code.

## V. Synthesis Results and Decoding Performance

### A. Synthesis Results

In this section, we report the ASIC synthesis results of our decoder implementations. The proposed decoders are implemented using the Digital Standard Cell Library SAED-EDK90-CORE, Process 1P9M 1.2v/2.5v. The hierarchical design flow is followed with the Synopsys Design Platform: Synopsys Design Compiler is used for VHDL synthesis, Synopsys VCS tool for simulation and DVE, a graphical user interface for debugging and viewing waveforms. Synopsys Prime Time tool is used for timing analysis, power and energy estimation, and Synopsys IC Compiler (ICC) is used for floor planning, place and route.

We demonstrate in the first synthesis comparison the achievable area gains when using the VNSA approach. In this comparison, we have set the timing constraint identical for all decoders, fixed to 10 ns, assuming that the decoders operate at the same clock frequency (100Mhz). By doing that, we measure precisely the impact of VNSA on the hardware cost, even if the working frequency is not maximized. The results are reported in Table I for the dv3R050N1296 QC-LDPC code and the values in brackets indicate the percentage of additional (+) or saving (−) cost, compared to the deterministic GDBF implementation. The implementations of GDBF and CSTC-PGDBF in all tables are the ones in [21] (CSTS-PGDBF and all VNSA-based PGDBF decoders are with $p_0 = 0.7$). Since the post place-and-route results are not available in that paper, we have re-synthesized them and obtained those values. As the first remark, it is clear that, the VNSA is an alternative implementation solution since no extra complexity required to implement VNSA-based GDBF decoder compared to the conventional GDBF implementation. The second remark is that, VNSA allows reducing the decoder complexity as expected. The VNSA-PGDBF requires less than the conventional GDBF implementation, around 6.68% while the conventional PGDBF implementation needs 3.77% extra cost. As also expected, the VNSA-IM-PGDBF largely reduces the complexity with 15.1% lower than the GDBF decoder. This complexity gain is significant since VNSA-IM-PGDBF provides an equivalent decoding performance to CSTS-PGDBF, $|R_t| = 4Z$ (shown in Fig. 7(a)) while being lower complexity than GDBF. The similar conclusions have been verified for LDPC codes with different lengths, rates and values of $d_v$. Indeed, it can be seen in the Table II that, while the CSTS-PGDBF decoder always requires an amount of extra resource (from 2% to 14% compared to GDBF), the VNSA-PGDBF decoder reduces 2.5% to 5% and the VNSA-IM-PGDBF decoder reduces 13% to 17% the complexity with respect to GDBF decoder.

TABLE I

Comparison on Hardware Resources to Implement the GDBF and PGDBF Decoders by Using the Conventional and the VNSA Architectures

| **dv3R050N1296 - AREA** $(mm^2)$ | | | |
|---|---|---|---|
| | **Conventional [21]** | **VNSA-based** | **Imprecise VNSA-based** |
| **GDBF** | 0.292 (+0%) | 0.2913 (-0.24%) | N/A |
| **PGDBF** | 0.303 (+3.77%) | 0.2725 (-6.68%) | 0.2479 (-15.1%) |

In the second comparison, the synthesis strategy for each decoder is to look for the smallest timing constraint that the synthesizer can pass. The maximum frequency, $f_{max}$, that the decoders can operate on, is then determined from that smallest timing constraint. We report the frequency and average throughput in Table III. The average throughput ($\theta$) of the decoders is computed as $\theta = \frac{f_{max} * N}{It_{ave} * N_c}$ where $N_c$ denotes the number of clock cycles needed for one decoding iteration and $It_{ave}$ is the average number of iterations. We take also 2 examples of decoder implementations in literature for the comparisons. The first implementation is the Adaptive Threshold Bit Flipping from [27]. Although ATBF was designed and tested on the AWGN channel, it is worth to make the

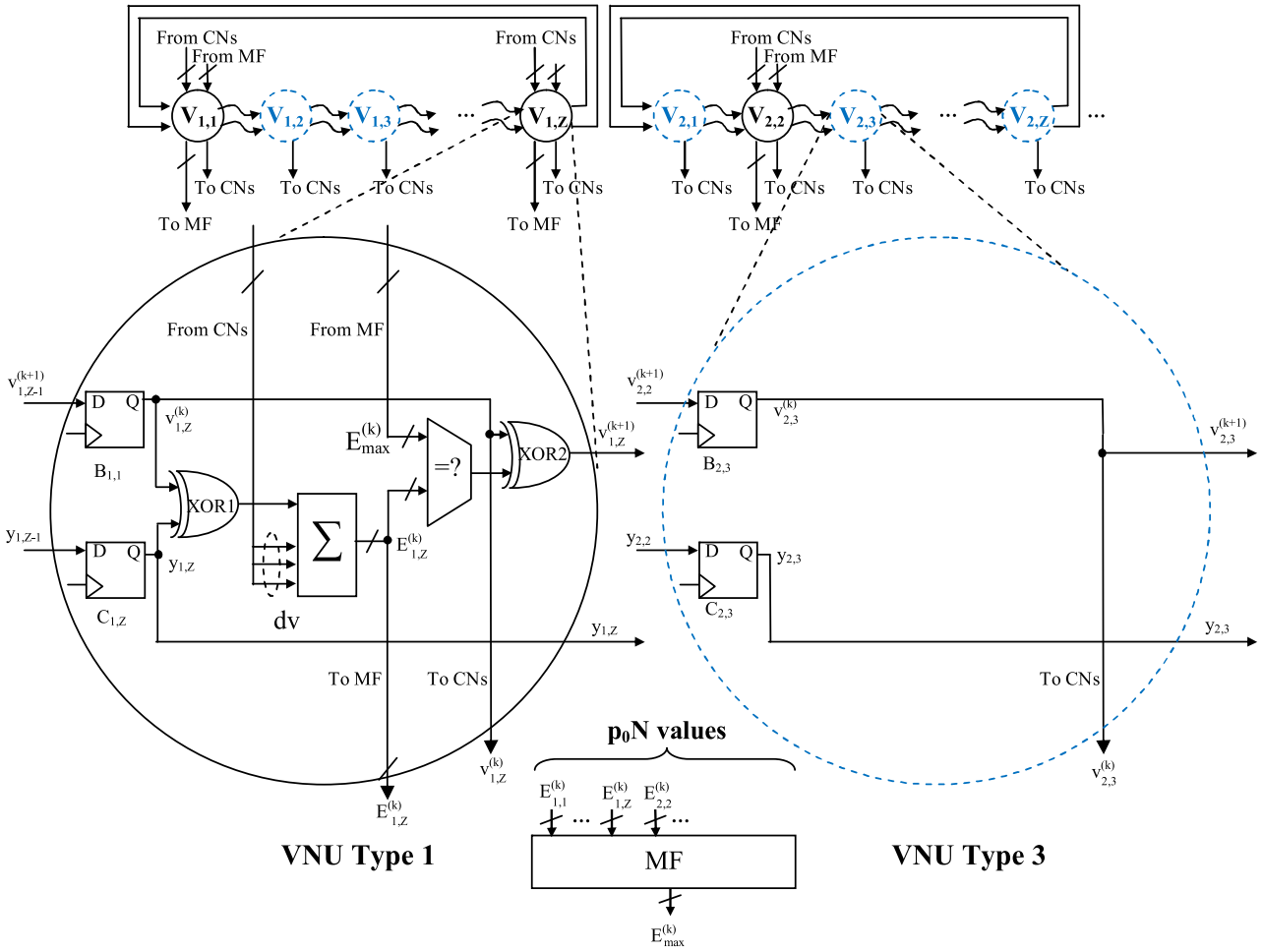



Fig. 6. The imprecise implementation of VNSA-PGDBF decoder, VNSA-IM-PGDBF.

TABLE II

HARDWARE RESOURCE USED TO IMPLEMENT GDBF AND PGDBF DECODERS FOR LDPC CODES WITH DIFFERENT CODEWORD LENGTHS AND CODE RATES. THE VALUES IN BRACKETS ARE THE ADDITIONAL (+) OR SAVING (−) HARDWARE RESOURCES WITH RESPECT TO THE GDBF

| **AREA** ($mm^2$) | | | | | | |
|---|---|---|---|---|---|---|
| | dv= 3 | | dv= 4 | | | |
| | $d_c = 5$ | $d_c = 6$ | $d_c = 8$ | $d_c = 16$ | $d_c = 28$ | $d_c = 34$ |
| **(Code length, code rate)** | (155, 0.4) | (1296, 0.75) | (1296, 0.5) | (1296, 0.75) | (2212, 0.857) | (9520, 0.88) |
| **GDBF** | 0.0364 (+0.0%) | 0.2891 (+0.0%) | 0.3896 (+0.0%) | 0.3967 (+0.0%) | 0.6682 (+0.0%) | 3.2291 (+0.0%) |
| **LFRS-PGDBF (S = M)** | 0.0416 (+14.29%) | 0.3271 (+13.14%) | 0.4172 (+7.08%) | 0.4132 (+4.16%) | 0.6972 (+4.34%) | 3.2967 (+2.09%) |
| **VNSA-PGDBF** | 0.0353 (-3.02%) | 0.2782(-3.77%) | 0.3767 (-3.31%) | 0.3772 (-4.92%) | 0.6516 (-2.48%) | 3.1104 (-3.68%) |
| **VNSA-IM-PGDBF** | 0.0316 (-13.19%) | 0.2516 (-12.97%) | 0.3263 (-16.25%) | 0.3303 (-16.74%) | 0.5630 (-15.74%) | 2.76 (-14.53%) |

comparison since ATBF is a GDBF-based BF decoder and the difference is only from the quantized channel value in energy equation. The reported results were also on the regular $d_v = 3, d_c = 6$ with codeword length, $N = 1008$, being close to our test code. We want to further compare our BF decoders to a soft-decision decoder by using the MS in [34] as the second reference. The MS in [34] is a quantized decoder with 4 bits for messages, 6 bits for APP information and on the same dv3R050N1296 LDPC code. Note that, the hardware results of the MS are extracted from that paper while the average iterations and the error correction performance is based on our simulations on BSC channel.

The conclusions drawn from the first comparison that the VNSA helps reduce significantly the complexity of PGDBF implementations, are maintained in this second comparision. Indeed, VNSA-PGDBF requires only 89% of GDBF complexity to be efficiently implemented. The reduction is more significant in the case of VNSA-IM-PGDBF with around 18.3% complexity reduced with respect to that of the GDBF. Interestingly, the maximum frequency of VNSA-IM-PGDBF

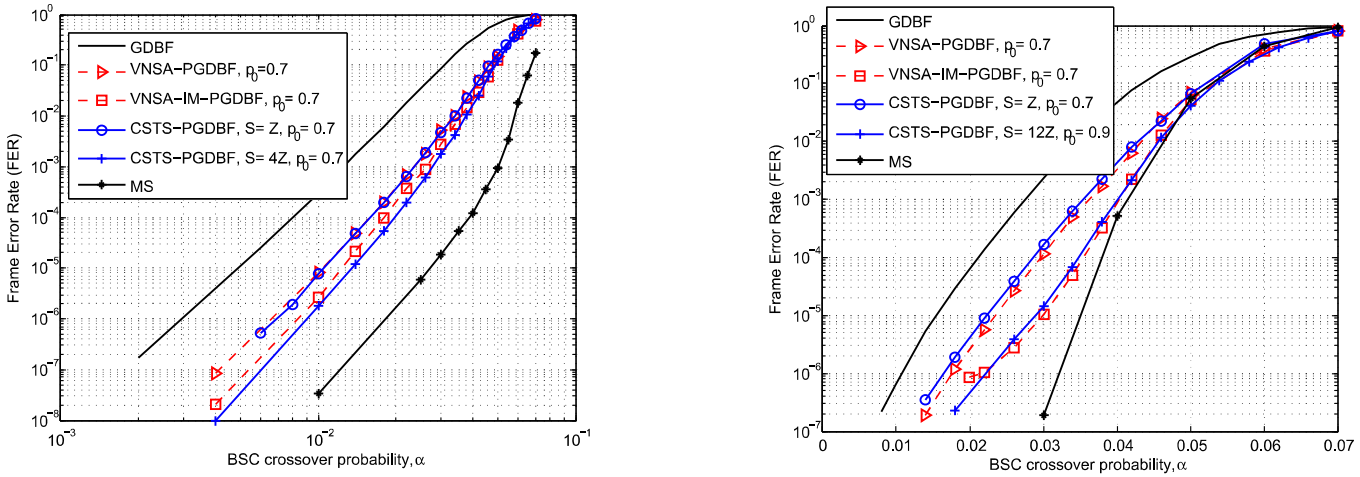



Fig. 7. The decoding performance implemented decoders on the QC-LDPC codes. (a) dv3R050N1296 QC-LDPC code. (b) dv4R050N1296 QC-LDPC code.

TABLE III
THE AREA, POWER AND THROUGHPUT COMPARISON BETWEEN GDBF, PGDBF AND THE DECODERS ON THE LITERATURE

| | Code length | Code rate | Technology | AREA ($mm^2$) | Power (mW) | $f_{max}$ (MHz) | $N_c$ | $FER = 1e^{-5}$ | | $\alpha = 0.01$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | $It_{ave}$ | $\theta$ (Gbit/s) | $It_{ave}$ | $\theta$ (Gbit/s) |
| **GDBF** | 1296 | 1/2 | 90nm | 0.360 (+0.0%) | 61 | 385 | 1 | 2.24 (@$\alpha \approx 0.005$) | 222.8 | 2.95 ($FER = 3e^{-4}$) | 169.1 |
| **CSTS-PGDBF(S = Z = 54)[21],** $p_0 = 0.7$ | 1296 | 1/2 | 90nm | 0.367 (+2.0%) | 51.8 | 357 | 1 | 5.48 (@$\alpha \approx 0.011$) | 84.4 | 5.15 ($FER = 7e^{-6}$) | 89.8 |
| **VNSA-PGDBF**, $p_0 = 0.7$ | 1296 | 1/2 | 90nm | 0.320 (-11.1%) | 82.4 | 370 | 1 | 4.83 (@$\alpha \approx 0.010$) | 99.3 | 4.83 ($FER = 8e^{-6}$) | 99.3 |
| **VNSA-IM-PGDBF**, $p_0 = 0.7$ | 1296 | 1/2 | 90nm | 0.294 (-18.3%) | 76.0 | 400 | 1 | 6.38 (@$\alpha \approx 0.013$) | 81.3 | 5.32 ($FER = 2.6e^{-6}$) | 97.4 |
| **ATBF** [27] | 1008 | 1/2 | 90nm | 0.63 | 33.14 | 250 | - | 25.2Gbps @ 10 iterations | | | |
| **MS** [34] | 1296 | 1/2 | 65nm | 0.72 | - | 250 | 6 | 2.34 (@$\alpha = 0.025$) | 23.1 | 1.29 ($FER = 1e^{-7}$) | 41.9 |

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 [27], 1296 compared to 1008, our BF decoders are, however, less than a half of the ATBF complexity while being higher in operating frequency, $370 - 400$Mhz compared to 250Mhz. The VNSA-based PGDBF decoders are much lower complexity in comparison to MS with the same codeword length. Indeed, 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 synthesis results show that, VNSA-based PGDBF decoders consume more energy than GDBF and CSTS-PGDBF, with 76mW of VNSA-IM-PGDBF, 82.4mW of VNSA-PGDBF compared to 51.8mw and 61mW of GDBF and CSTS-PGDBF, respectively. This comes from the fact that, due to the cyclic shift of the tentative codeword and the channel value for each and every iteration, VNSA-based PGDBF decoders result more switching activities than the decoders on conventional architecture. Also, it is because VNSA-based PGDBF decoders operate at higher frequency.

For decoding throughput comparisons, we compare the decodes in the 2 scenarios in which the throughput values are computed at a target Frame Error Rate, *i.e.* $FER = 1e^{-5}$, and at a certain level of channel noise, *i.e.* $\alpha = 0.01$. Our implementation of BF decoders allow to perform one iteration in $N_c = 1$ clock cycle, which results in a very high decoding throughput and much higher than ATBF and MS. At the same channel noise level, $\alpha = 0.01$, the VNSA-PGDBF is faster 2.4 times than MS decoder and 3.9 times than ATBF, but in 1.9 times slower than the GDBF. GDBF offers the best throughput but with limitation in error correction, $FER = 3e^{-4}$ compared to $2.6e^{-6}$ of VNSA-IM-PGDBF and $1e^{-7}$ of MS. In order to maintain a target performance, *i.e.* $FER = 1e^{-5}$, the VNSA-based PGDBF decoders are $3.5-4.3$ times faster than the MS decoder and 2.7 times slower than GDBF.

### *B. Decoding Performance*

In this section, we illustrate the decoding performance of VNSA-based PGDBF decoders in comparison with GDBF and MS decoders. The MS decoder is set with $It_{max} = 20$ while the GDBF and the PGDBF decoders are set with $It_{max} = 300$. We use the two regular LDPC codes introduced above for the simulations (the dv3R050N1296 and the dv4R050N1296 codes). It can be firstly seen in Fig. 7 that, the VNSA-based PGDBF decoders are much better than GDBF and approaching to the MS performance, especially on the dv4R050N1296 code. VNSA-IM-PGDBF is considerable better than the VNSA-PGDBF for both LDPC codes. Interestingly, VNSA-IM-PGDBF is almost equal to the theoretical PGDBF performance (the CSTS-PGDBF with $S = 4Z$ for dv3R050N1296 and $S = 12Z$ for dv4R050N1296 as presented in [21]). For the dv3R050N1296 code, the FER of

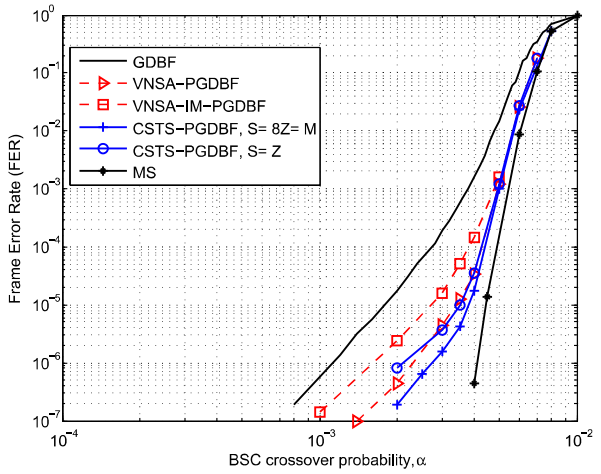



Fig. 8. Decoding performance of VNSA-based PGDBF decoders on the $(d_v, d_c) = (4, 34)$, $Z = 140$, $M = 1120$, $N = 9520$, code rate $R = 0.88$ QC-LDPC code.

VNSA-IM-PGDBF maintains closely to theoretical PGDBF decoder at a very low error rate ($1e^{-8}$) while for the dv4R050N1296, the error floor appears from the $FER = 1e^{-6}$. The MS decoder is the best in error correction at the cost of the increased decoding complexity and reduced throughput as shown in previous section.

Fig. 8 shows another decoding performance comparison between VNSA-based PGDBF decoders with other decoders on a long and high rate code, the $(d_v, d_c) = (4, 34)$, $Z = 140, M = 1120, N = 9520$, code rate $R = 0.88$ QC-LDPC code. All PGDBF decoders are with $p_0 = 0.9$. These performance curves re-confirm the advantage of VNSA in providing the good decoding performance mentioned above. However, VNSA-IM-PGDBF tends to show a high error floor on this code, leading an inferior decoding performance to other implementations.

## VI. Conclusion

In this paper, we introduce a new hardware architecture for QC-LDPC decoding, called Variable Node Shift Architecture (VNSA). The VNSA is used to efficiently implement the Probabilistic Gradient Descent Bit Flipping decoder. The VNSA is shown to be an alternative solution for QC-LDPC decoding implementation and its advantages, in terms of decoding performance and decoder complexity, come when different types of variable node processing units (VNUs) and/or check node processing units (CNUs) are implemented. The first implementation of Probabilistic Gradient Descent Bit Flipping using VNSA is called VNSA-PGDBF. In VNSA-PGDBF, 2 types of VNUs are designed which are simpler than those of the conventional PGDBF implementation. VNSA-PGDBF offers the error correction as good as the optimized PGDBF implementation in the literature, while reducing the decoder complexity up to 11% less than the one of the deterministic Gradient Descent Bit Flipping (GDBF) decoder. We further propose another simplified and imprecise version of VNSA-PGDBF, called VNSA-IM-PGDBF. In VNSA-IM-PGDBF, a new and trivial VNU is introduced in which all computing circuits are eliminated. The impreciseness appears at the level of the maximum finder where the maximum energy is found in a smaller list of candidates. Interestingly, VNSA-IM-PGDBF offers a better error correction performance than the VNSA-PGDBF while further reducing the hardware cost to 18% lower than that of the GDBF. VNSA-based PGDBF decoders, with outstanding decoding capability, low complexity and ultra high decoding throughput, are the competitive solutions for the next communication and storage 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*. Amsterdam, The Netherlands: Elsevier, 2014.

[2] 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.

[3] 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)*, May 2015, pp. 1450–1453.

[4] 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.

[5] *IEEE Standard for Information Technology—Telecommunications and Information Exchange Between Systems—Local and Metropolitan Area Networks—Specific Requirements Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications—Corrigendum 2*, IEEE Standard 802.3-2005 (Corrigendum to IEEE Std 802.3-2005), Aug. 2007.

[6] *IEEE Standard for Information Technology– Local and Metropolitan Area Networks– Specific Requirements– Part 11: Wireless LAN Medium Access Control (MAC)and Physical Layer (PHY) Specifications Amendment 5: Enhancements for Higher Throughput*, IEEE Standard 802.11n-2009, Mar. 2008.

[7] *IEEE Standard for Local and Metropolitan Area Networks—Part 16: Air Interface for Fixed Broadband Wireless Access Systems—Amendment 2: Medium Access Control Modifications and Additional Physical Layer Specifications for 2-11 GHz*, IEEE Standard 802.16a-2003, 2003.

[8] R. G. Gallagher, *Low–Density Parity–Check Codes* (Research Monograph). Cambridge, MA, USA: MIT Press, 1963.

[9] Y. Kou, S. Lin, and M. P. C. Fossorier, "Low-density parity-check codes based on finite geometries: A rediscovery and new results," *IEEE Trans. Inf. Theory*, vol. 47, no. 7, pp. 2711–2736, Nov. 2001.

[10] 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.

[11] 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.

[12] 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.

[13] O. Al 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.

[14] G. Sundararajan, C. Winstead, and E. Boutillon, "Noisy gradient descent bit-flip decoding for LDPC codes," *IEEE Trans. Commun.*, vol. 62, no. 10, pp. 3385–3400, Oct. 2014.

[15] Y. H. Liu, X. L. Niu, and M. L. Zhang, "Multi-threshold bit flipping algorithm for decoding structured LDPC codes," *IEEE Commun. Lett.*, vol. 19, no. 2, pp. 127–130, Feb. 2015.

[16] T. C.-Y. Chang and Y. T. Su, "Dynamic weighted bit-flipping decoding algorithms for LDPC codes," *IEEE Trans. Commun.*, vol. 63, no. 11, pp. 3950–3963, Nov. 2015.

[17] S. Imani, R. Shahbazian, and S. A. Ghorashi, "An iterative bit flipping based decoding algorithm for LDPC codes," in *Proc. Iran Workshop Commun. Inf. Theory (IWCIT)*, May 2015, pp. 1–3.

[18] H. Huang, Y. Wang, and G. Wei, "Mixed modified weighted bit-flipping decoding of low-density parity-check codes," *IET Commun.*, vol. 9, no. 2, pp. 283–290, 2015.

[19] K. Ma, J. Jin, W. Li, and P. Zhang, "Two-staged weighted bit flipping (WBF) decoding algorithm for LDPC codes," in *Proc. IEEE 9th Int. Conf. Anti-Counterfeiting, Secur., Identificat. (ASID)*, Sep. 2015, pp. 141–144.

[20] K. Le *et al.*, "Efficient realization of probabilistic gradient descent bit flipping decoders," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Lisbon, Portugal, May 2015, pp. 1494–1497.

[21] K. Le, F. Ghaffari, D. Declercq, and B. Vasić, "Efficient hardware implementation of probabilistic gradient descent bit-flipping," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 64, no. 4, pp. 906–917, Apr. 2017.

[22] J. Li, S. Lin, K. Abdel-Ghaffar, W. E. Ryan, and D. J. Costello, Jr., *LDPC Code Designs, Constructions, and Unification*. Cambridge, U.K.: Cambridge Univ. Press, 2016.

[23] S. Kim, J. S. No, H. Chung, and D. J. Shin, "Quasi-cyclic low-density parity-check codes with girth larger than 12," *IEEE Trans. Inf. Theory*, vol. 53, no. 8, pp. 2885–2891, Aug. 2007.

[24] N. Bonello, S. Chen, and L. Hanzo, "Construction of regular quasi-cyclic protograph LDPC codes based on vandermonde matrices," *IEEE Trans. Veh. Technol.*, vol. 57, no. 4, pp. 2583–2588, Jul. 2008.

[25] K. Zhang, X. Huang, and Z. Wang, "High-throughput layered decoder implementation for quasi-cyclic LDPC codes," *IEEE J. Sel. Areas Commun.*, vol. 27, no. 6, pp. 985–994, Aug. 2009.

[26] B. Xiang, D. Bao, S. Huang, and X. Zeng, "An 847–955 Mb/s 342–397 mW dual-path fully-overlapped QC-LDPC Decoder for WiMAX system in 0.13 $\mu$m CMOS," *IEEE J. Solid-State Circuits*, vol. 46, no. 6, pp. 1416–1432, Jun. 2011.

[27] M. Ismail, I. Ahmed, and J. Coon, "Low power decoding of LDPC codes," *ISRN Sensor Netw.*, vol. 2013, Dec. 2013, Art. no. 650740.

[28] Y.-L. Ueng, C.-Y. Wang, and M.-R. Li, "An efficient combined bit-flipping and stochastic LDPC decoder using improved probability tracers," *IEEE Trans. Signal Process.*, vol. 65, no. 20, pp. 5368–5380, Oct. 2017.

[29] J. Jung and I.-C. Park, "Multi-bit flipping decoding of LDPC codes for NAND storage systems," *IEEE Commun. Lett.*, vol. 21, no. 5, pp. 979–982, May 2017.

[30] T. Brack *et al.*, "Low complexity LDPC code decoders for next generation standards," in *Proc. Design, Autom. Test Eur. Conf. Exhib.*, Nice, France, 2007, pp. 1–6.

[31] T. Nguyen-Ly *et al.*, "FPGA design of high throughput LDPC decoder based on imprecise Offset Min-Sum decoding," in *Proc. IEEE 13th Int. New Circuits Syst. Conf. (NEWCAS)*, Jun. 2015, pp. 1–4.

[32] 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.

[33] D. Declercq, C. Winstead, B. Vasić, F. Ghaffari, P. Ivanis, and E. Boutillon, "Noise-aided gradient descent bit-flipping decoders approaching maximum likelihood decoding," in *Proc. 9th Int. Symp. Turbo Codes Iterative Inf. Process. (ISTC)*, Sep. 2016, pp. 300–304.

[34] 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 *Proc. Euromicro Conf. Digit. Syst. Design (DSD)*, Aug./Sep. 2016, pp. 230–237.


**Khoa Le** received the B.Sc. and M.Sc. degrees in electronics engineering from the Ho Chi Minh City University of Technology, Vietnam, in 2010 and 2012, respectively, the Ph.D. degree from the University of Cergy-Pontois, France, in 2017.He is currently a Post-Doctoral Researcher with the ETIS Laboratory, ENSEA, France. His research interests are in error correcting code algorithms, analysis and their implementations in FPGA/ASIC.

**David Declercq** (SM'11) was born in 1971. He received the Ph.D. degree in statistical signal processing from the University of Cergy-Pontoise, France, in 1998. He is currently a Full Professor with ENSEA, Cergy-Pontoise. He is a General Secretary of the National GRETSI Association. He has 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 was involved 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 the design of GF(q) LDPC codes for short and moderate lengths, and 2) the simplification of the iterative decoders for GF(q) LDPC codes with complexity/performance tradeoff constraints. He has authored over 40 papers in major journals the IEEE TRANSACTIONS COMMUNICATION, the IEEE TRANSACTIONS INFORMATION THEORETICAL COMMUNICATIONS LETTERS, the EURASIP JWCN, and over 120 papers in major conferences in Information Theory and Signal Processing.

**Fakhreddine Ghaffari** received the Degree in electrical engineering and the master's degree from the National School of Electrical Engineering, Tunisia, in 2001 and 2002, respectively, and the Ph.D. degree in electronics and electrical engineering from the University of Sophia Antipolis, France, in 2006. He is currently an Associate Professor with 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.

**Lounis Kessal** received the Ph.D. degree in microelectronics from Paris-XI University in 1987. He is currently an Associate Professor with the ENSEA Engineering School, and involving in research activities with the ETIS Laboratory. His research interests are essentially on real-time image processing.

His previous research works lead him to study the interest of the dynamic reconfiguration approach in architectures dedicated to signal and image processing. He is also involved in reconfigurable systems on chip and ASIC/FPGA implementations of error correcting code algorithms.

**Oana Boncalo** received the B.Sc. and Ph.D. degrees in computer engineering from University Politehnica Timisoara, Romania, in 2006 and 2009, respectively. She is currently an Associate Professor with University Politehnica Timisoara. She has authored over 50 research papers in topics related to digital design. Her research interests include computer arithmetic, LDPC decoder architectures, digital design, and reliability estimation and evaluation.

**Valentin Savin** received the master's degree in mathematics from the École normale supérieure de Lyon in 1997, and the Ph.D. degree in mathematics from the J. Fourier Institute, Grenoble, in 2001, and the master's degree in cryptography, security and coding theory from the University of Grenoble 1. Since 2005, he has been with the Digital Communications Laboratory, CEA-LETI, first as a two-year Post-Doctoral Fellow, and then as a Research Engineer. Since 2016, he has been an appointed CEA Senior Expert in information and coding theory. Over the last years, he has been involved in the design of low-complexity decoding algorithms for LDPC and Polar codes, and on the analysis and the optimization of LDPC codes for physical and upper-layers applications. He has authored over 80 papers in international journals and conference proceedings, holds ten patents. He is currently participating in or coordinating several French and European research projects in ICT.