Ultra-high-throughput EMS NB-LDPC decoder with full-parallel node processing

Hassan Harb, Ali Chamas Al Ghouwayel, Laura Conde-Canencia, Cédric Marchand, Emmanuel Boutillon

Ultra-high-throughput EMS NB-LDPC decoder with full-parallel node processing

Hassan Harb, Ali Chamas Al Ghouwayel, Laura Conde-Canencia, Cédric Marchand and Emmanuel Boutillon

***Abstract*—This paper presents an ultra-high-throughput decoder architecture for NB-LDPC codes based on the Hybrid Extended Min-Sum algorithm. We introduce a new processing block that updates a check node and its associated variable nodes in a fully pipelined way, thus allowing the decoder to process one row of the parity check matrix per Clock Cycle (CC). The work specifically focuses on a rate 5/6 code of size (*N*, *K*) = (144, 120) symbols over GF(64). The synthesis results on 28-nm technology show that the proposed architecture improves the throughput efficiency of the state of the art by a factor greater than 10. The architecture reaches a throughput above 10 Gbps for SNR values greater than 4 dB. Compared to a 5G binary LDPC code of same size and code rate, the proposed architecture offers a gain of 0.3 dB at a Frame Error Rate of 10^{-3}. One of the unexpected results is that the proposed architecture almost halves the memory bandwidth compared to a classical binary LDPC code.**

***Index Terms*—Channel coding, decoder implementation, ASIC, non-binary LDPC, Min-Sum, parity check.**

I. Introduction

NON-Binary (NB) Low-Density Parity-Check (LDPC) codes allow to close the performance gap with the Shannon limit [1] when using small or moderate frame lengths. They are defined on high order Galois Fields (GF) of order q with $q > 2$ and have been proven to be more robust than convolutional turbo-codes and binary LDPC codes [2]. However, even if they present numerous advantages (see [3] [4]) their main drawback is the complexity, which is challenging at the receiver side. In NB-LDPC decoders, the direct application of the Belief Propagation (BP) [5] algorithm leads to $O(q^2)$ complexity and is thus prohibitive for $q > 16$. A considerable amount of work has then been dedicated to reduce the complexity of decoding algorithms and their associated architectures ([6] [7] [8], among others), with a special focus on the Check Node (CN) processing which is the major bottleneck in NB-LDPC decoders.

This work particularly focuses on the Extended Min-Sum (EMS) algorithm [9] [10] as it currently continues to present one of the most competitive complexity/performance trade-offs [11] [12]. For the EMS CN implementation, the Forward-Backward (FB) approach was introduced in [12] as a serial concatenation of Elementary CNs (ECNs). This structure suffers from high latency and low decoding throughput. Other approaches were proposed (e.g., the Trellis-EMS (T-EMS) [13]) to reduce latency. This complexity was reduced with the one-minimum T-EMS [14] and the Trellis Min-Max (T-MM) [15] [16] algorithms. However, all variant of the T-EMS algorithms present a complexity that increases with the cardinal q of the Galois Field.

In this paper we show some innovative ideas that significantly enhance the throughput and the hardware efficiency of the prior designs [4] [17]. This prior work includes the Syndrome-based algorithm [18] that efficiently performed parallel CN computations for $q \geq 16$ and was initially considered for implementing a GF(256) CN processor with a CN degree $d_c = 4$ [19]. However, its complexity is dominated by the number of computed syndromes which increases quadratically with d_c. This limits its interest for high coding rates (i.e., high d_c values). A solution was then proposed based on sorting the input vectors according to a reliability criteria [20] [21] to significantly reduce the CN hardware complexity without affecting performance. This so-called *presorting* technique was applied to the syndrome-based architecture in [20] and to the FB architecture in [21]. A hybridization of those two architectures was presented in [17] for high q and d_c values.

In this paper we consider the work in [17] to design a decoder that significantly outperforms the state of the art in terms of throughput and efficiency. The design includes the presorting technique and an innovative unit that processes both the CNs and Variable Nodes (VN) in a fully pipeline architecture. This architecture is called "Full Parallel Hybrid Check Node" (FPHCN) in the paper. For the practical description of the decoder, we consider a specific code but the new proposed principles can be easily generalized to any NB-LDPC code, knowing that its benefits are specially interesting for high rates. We synthesized the decoder on 28-nm technology and performed a detailed study of its throughput and efficiency for comparison with the state of the art. The proposed design reaches a decoding throughput above 10 Gbit/s for a CN degree of 12, which represents a significant gain compared to [17]. In terms of throughput efficiency (i.e., throughput per gate ratio), the gain factor is in the order of 3.4 (for low SNR values) and up to 13 (for high SNR values). This improvement in terms of throughput efficiency is also helped by the concurrent processing of two frames at the cost of the duplication of the memory banks.

The paper is organized as follows: Section II introduces notation, NB-LDPC codes, the EMS algorithm principles and the code structure considered in this work. Section III recalls the presorting technique and describes the decoding steps and the CN-VN merging technique. Section IV is dedicated to the global decoder architecture. Simulation results for the proposed code and its binary LDPC counterpart are compared in Section V. Implementation results, throughput analysis and a detailed comparison with the state of the art are presented in section VI. Finally, conclusions and perspectives are discussed in Section VII.

II. NOTATION, NB-LDPC CODES AND EMS ALGORITHM

This Section introduces NB-LDPC codes, describes the calculation of the intrinsic messages and the principles of the EMS algorithm. Table I lists the symbols and acronyms considered throughout the paper, which include the characteristics of the code, the exchanged messages in the decoder and other terms for the description of the global decoder.

A. NB-LDPC codes defined over Galois Fields

A NB-LDPC code is a linear block code defined on a very sparse parity-check matrix H whose non-zero elements belong to a finite field GF(q), where $q > 2$. The elements of GF(q) are $\{0, \alpha^0, \alpha^1, \ldots, \alpha^{q-2}\}$. The dimension of matrix H is $M \times N$, where M is the number of parity-CNs and N is the number of VNs (i.e., the number of GF(q) symbols in a codeword). A codeword is denoted by $\mathbf{C} = (x_0, x_1, \ldots, x_{N-1})$, where x_k, $k = 0, \ldots, N-1$, is a GF(q) symbol represented by $m = \log_2(q)$ bits as $x_k = (x_{k,0}\ x_{k,1} \ldots x_{k,m-1})$. The construction of regular (d_v, d_c) NB-LDPC codes is expressed as a set of M parity-check equations C_j, $j = 0, 1, \ldots, M-1$, over GF(q), where the j^{th} parity check equation C_j is given as

$$C_j : \sum_{i=0}^{d_c-1} h_{j,k(j,i)} x_{k(j,i)} = 0, \tag{1}$$

where $\{k(j,i)\}_{i=0,1,\ldots,d_c-1}$ is the set of the d_c non-null positions in the j^{th} row of the matrix H and $h_{j,k(j,i)}$ its associated GF values. Each variable is connected to exactly d_v non-null elements in a column.

B. Intrinsic messages

The exchanged messages in the EMS decoding algorithm are Log Likelihood Ratio (LLR) values. The intrinsic LLR values are computed from an observed information coming from the channel. If we consider the Additive White Gaussian Noise (AWGN) channel and a Binary Phase-Shift Keying (BPSK) modulation, the GF symbol x will be modulated by m BPSK channels with amplitude $\mathrm{B}(x_p) = (-1)^{x_p}$, $p = 0, \ldots, m-1$. At the receiver side, the received samples r_p are expressed as:

$$r_p = \mathrm{B}(x_p) + w_p,$$

where w_p is a realization of a Gaussian noise of variance σ^2.

Let $Y = (y_0, y_1, \ldots, y_{m-1})$ be the LLR intrinsic vector associated to x. Each value y_p is defined as:

$$y_p = \log\left(\frac{\mathrm{P}(x_p = 0/r_p)}{\mathrm{P}(x_p = 1/r_p)}\right) = \frac{2r_p}{\sigma^2}. \tag{2}$$

Let $\bar{x} = (\bar{x}_0, \bar{x}_1, \ldots, \bar{x}_{m-1})$, where the values $\bar{x}_p$ are defined as hard decisions such that if $\mathrm{sign}(y_p) > 0$, then $\bar{x}_p = 0$, $\bar{x}_p = 1$ otherwise. Considering the hypothesis that all the symbols in the GF(q) alphabet have equal probability, the expression of the LLR $I^+(x)$ of a symbol x knowing y is expressed as:

$$I^+(x) = \sum_{p=0}^{m-1} |y_p| \Delta(x_p, \bar{x}_p), \tag{3}$$

TABLE I
NOTATION

Symbols	
q	Order of GF
$\mathcal{H}$, H	Prototype matrix and Parity-Check Matrix (PCM)
(M, N)	Number of rows (CNs) and columns (or VNs) in H
d_c	Degree of connectivity of the CN
d_v	Degree of connectivity of the VN
h	non-zero GF elements in H
$h_{j,k}$	non-zero element in H that connects j^{th} CN with k^{th} VN
$\{0, \alpha^0, \ldots, \alpha^{q-2}\}$	GF(q) elements
x	a GF symbol
$Y = (y_0, y_1, \ldots, y_{m-1})$	m bit LLR values associated to a symbol
$\Pi = \{\pi(0), \pi(1), \pi(2)\}$	Indexes of the 3 smallest magnitudes of y
$(X^+, X^\oplus)$	LLR and GF values of X
$n_{m_{in}}$	Number of CN input messages
$n_{m_{out}}$	Number of CN output messages
$I = \{I^+[0], I[1], \ldots, I[n_{m_{in}} - 1]\}$	Intrinsic vector of size $n_{m_{in}}$
$\tilde{I} = \{\lvert y_p\rvert_{p=0,\ldots,m-1}, \Pi, I^+[0]\}$	Pre-processed intrinsic vector
$I[j] = (I^+[j], I^\oplus[j])$	j^{th} element of I composed of a couple ($I^+[j]$ = LLR value, $I^\oplus[j]$ = GF value)
$U = \{U[0], \ldots, U[n_{m_{in}} - 1]\}$	CN input vector
$U' = \{U'[0], \ldots, U'[n_{m_{in}} - 1]\}$	Switched CN input vector after presorting
$V = \{V[0], \ldots, V[n_{m_{out}} - 1]\}$	CN output vector
$\hat{x}$	Decision on VN V
$n_{max,it}$	Maximum number of iterations
$n_{av,it}$	Average number of iterations
$\Psi = \{\psi_0, \ldots, \psi_{d_c-1}\}$	Indexes generated by the presorting
b	Number of bits of quantization
Acronyms	
SN	Syndrome Node
VSV	Valid Syndrome Vector
DB	Decorrelation Block
CU	Control Unit
DMU	Decision Making Unit
DMR	Decision Making Reorder
PTB	Parity Test Block
FPHCN	Full Parallel Hybrid CN

where $\Delta(x_p, \bar{x}_p) = 0$ if $x_p = \bar{x}_p$ and $\Delta(x_p, \bar{x}_p) = 1$ otherwise. Note that, by definition, $I^+(\bar{x}) = 0$ is the smallest LLR value. It will be denoted as $I[0] = (I^+[0], I^\oplus[0])$, with the LLR $I^+[0] = I^+(\bar{x}) = 0$ and the associated GF value $I^\oplus[0] = \bar{x}$. Let $\Pi = \{\pi(0), \pi(1), \pi(2)\}$ be respectively the index of the smallest, the second smallest and the third smallest magnitude $|y_p|$ values, $p = 0, 1, \ldots, m-1$. Then, the second smallest LLR value $I^+[1]$ is obtained by flipping the bit of $\bar{x}$ of index $\pi(0)$ to obtain $I^\oplus[1]$, i.e., $I[1] = (I^+[1] = |y_{\pi(0)}|, I^\oplus[1])$. The third smallest value $I^+[2]$ is obtained by flipping the bit of $\bar{x}$ of index $\pi(1)$ to obtain $I^\oplus[2]$. Thus, $I[2] = (I^+[2] = |y_{\pi(1)}|, I^\oplus[2])$. Finally, the fourth smallest LLR value $I^+[3]$ is given by $\min(A, B)$ with $A = |y_{\pi(0)}| + |y_{\pi(1)}|$ and $B = |y_{\pi(2)}|$. If $\min(A, B) = A$, the associated GF value $I^\oplus[3]$ is obtained by flipping the bits of index $\pi(0)$ and $\pi(1)$ of $\bar{x}$, otherwise, if $\min(A, B) = B$, $I^\oplus[3]$ is obtained by flipping the bit of index $\pi(2)$ of $\bar{x}$. This

method and its generalization to compute in parallel the first $n_{m_{in}}$ terms of the intrinsic vector I are described in details in [22]. Finally, for hardware design, the LLR values need to be quantized on a fixed precision. To do so, we use the following expression

$$y_p = \text{sat}(\lfloor \gamma r_p Q + 0.5 \rfloor, Q),$$

where $\text{sat}(x, Q)$ is the clipping function of x in $[-Q, Q]$ ($\text{sat}(x, Q) = x$ if $|x| < Q$, $\text{sign}(x) \times Q$ otherwise), $\lfloor x \rfloor$ indicates the floor function, $Q = 2^{b-1} - 1$ is the saturation value expressed as a function of b, the number of bits of quantization (for $b = 6$, $Q = 31$). The fix scaling factor γ encompasses the scaling factor $2/\sigma^2$ found in the LLR expression of (2). The γ value is set empirically to $\gamma = 1.2$ in order to optimize the decoding performance. In the hardware architecture, we will consider $n_{m_{in}} = 4$. The intrinsic vector I associated to a given received symbol Y is thus composed as

$$I = (I^{\oplus}[0], I[1], I[2], I[3]). \tag{4}$$

Note that, with $b = 6$ and $m = 6$, the size n_I of I is given by $n_I = 3b + 4m = 42$ bits. Associated to the symbol Y, the pre-processed vector $\tilde{I}$ is defined as

$$\tilde{I} = (|y_0|, |y_1|, \ldots, |y_5|, \Pi, I^{\oplus}[0]), \tag{5}$$

and allows to reconstruct easily I with few hardware resources. It also allows to compute $I^+[\alpha]$ for any $\alpha \in$ GF(64) using (3). The size $n_{\tilde{I}}$ for $b = 6$ and $m = 6$ is $n_{\tilde{I}} = 6 \times 5 + 3 \times 3 + 6 = 45$ bits (note: $m - 1 = 5$ bits to encode $|y_p|$ and 3 bits to encode each index π of Π).

C. Extended Min-Sum algorithm for NB-LDPC codes

A detailed description of the different steps and equations in the Min-Sum (MS) algorithm was presented in [4]. We consider in this paper the Extended Min-Sum (EMS) [11] with the following characteristics: VN degree $d_v = 2$, CN input messages truncated to a size $n_{m_{in}} \ll q$ and CN output messages to $n_{m_{out}} \ll q$. This leads to computation and storage reduction without necessarily performance loss [9] [10].

For the EMS algorithm description, we define the following:

- $I = \{I[0], \ldots, I[n_{m_{in}} - 1]\}$ as an intrinsic LLR vector (see previous section),
- $\{h_0, \ldots, h_{d_c-1}\}$ are the d_c non-zero elements in H associated to a CN,
- $\{U_0, \ldots, U_{d_c-1}\}$ is the group of messages sent to a CN from the d_c connected VNs,
- $\{V_0, \ldots, V_{d_c-1}\}$ is the group of messages sent to d_c VNs from a CN.

Each U_i message can be written as $U_i = \{U_i[0], \ldots, U_i[n_{m_{in}} - 1]\}$. Each element $U_i[j]$ corresponds to a couple $U_i[j] = (U_i^+[j], U_i^{\oplus}[j])$ where $U_i^{\oplus}[j]$ is the GF(q) symbol and $U_i^+[j]$ its corresponding LLR value, $i = 0, \ldots, d_c - 1$ and $j = 0, \ldots, n_{m_{in}} - 1$. The elements in these messages are sorted related to their LLR values. The four steps of the EMS algorithm considering $d_v = 2$ are:

1) LLR calculation: The LLR calculation is defined in section II.B. It generates the intrinsic vector I of size $n_{m_{in}} = 4$. Each U message is initialized with the intrinsic value I of the connected VN.

2) CN update: The CN update is processed as

$$V_i^+(x) = \min\left\{ \sum_{i'=0, i' \neq i}^{d_c-1} U_{i'}^+[j_{i'}] \middle| \bigoplus_{i'=0, i' \neq i}^{d_c-1} U_{i'}^{\oplus}[j_{i'}] = x \right\}, \tag{6}$$

where $j_{i'} \in \{0, 1, \ldots, n_{m_{in}} - 1\}$ for $i' = 0, 1, \ldots, d_c - 1$, $i' \neq i$ and $\oplus$ refers to GF addition (i.e., XOR gate).

The final stage is to partially sort in increasing order the set of values of $V_i^+(x)$ indexed by $x \in$ GF(q) to obtain an ordered set $V_i^{\oplus} = \{x_0, x_1, \ldots, x_{n_{m_{out}}-1}\}$ that verifies

$$\forall (j, k),\ j < k < n_{m_{out}} \Rightarrow V_i^+(x_j) \leq V_i^+(x_k),$$

and

$$\forall x \in \text{GF}(q),\ x \notin V_i^{\oplus} \Rightarrow V_i^+(x_{n_{m_{out}}-1}) \leq V_i^+(x).$$

The i^{th} output message is thus given as $V_i = \{V_i[0], V_i[1], \ldots, V_i[n_{m_{out}-1}]\}$, where $V_i[j] = (V_i^+[j] = V_i^+(x_j), V_i^{\oplus}[j] = x_j)$, $j = 0, 1, \ldots n_{m_{out}-1}$.

In the state of the art, the GF values outside V_i are associated with a default LLR value D_i, with $D_i = V_i^+[n_{m_{out}} - 1] + O$, i.e., the default value is equal to the highest LLR value of the V message added with O, a positive offset value (see [10] for more details on the definition of the offset value). In section III.A, we propose a new method to determine the default value D_i to facilitate the hardware implementation of the full parallel CN architecture.

3) VN update: After processing CN a, the required inputs of a VN are: the intrinsic vector I of size $n_{m_{in}}$, the m values of Y to be able to compute $I^+(x)$ for any $x \in$ GF(q) using (3), the received message V^a of size $n_{m_{out}}$ coming from CN a, the default value D^a associated to message V^a and the message U^a sent to the CN a (U^a encompasses both I information and the updated message V^b coming from CN b. Note that U^a is used only for the decision process). The two outputs of the VN are the current message U^b of size $n_{m_{in}}$ to be sent to CN b and the VN decision $\hat{x}$ obtained by combining V^a and U^a.

The first step of the VN processing is the addition of the intrinsic LLR values on the incoming V^a message to generate the message $\bar{V}^a$ defined as:

$$\bar{V}^{a,+}[j] = V^{a,+}[j] + I^+(V^{a,\oplus}[j]),\ j = 0, 1, \ldots n_{m_{out}} - 1. \tag{7}$$

Since $\bar{V}^a$ associates LLR values for only a subset of GF values, in parallel, a second message $\bar{I}$ is generated as

$$\bar{I}^+[j] = I^+[j] + D^{a,+},\ j = 0, 1, \ldots n_{m_{in}} - 1. \tag{8}$$

Then, the $n_{m_{in}}$ smallest values of set $\bar{V}^a \cup \bar{I}$ in terms of LLR value are extracted along with their associated GF symbols to generate the vector messages $\bar{U}^b$. Note that by construction, $\bar{V}^{a,\oplus} \cap \bar{I}^{\oplus}$ may not be empty. In that case, the corresponding LLR element in $\bar{I}$ is saturated so that $\bar{U}^b$ contains the first $n_{m_{in}}$ smallest LLR values with distinct GF values. The last step to generate the final message is the normalization process that keeps the first LLR of the message equal to zero, i.e.,

$$\begin{aligned} U^{b,+}[j] &= \bar{U}^{b,+}[j] - \bar{U}^{b,+}[0], \\ U^{b,\oplus}[j] &= \bar{U}^{b,\oplus}[j],\ j = 0, \ldots, n_{m_{out}} - 1. \end{aligned} \tag{9}$$

4) Decision making: in the MS algorithm, the decision is done by adding all the incoming information, i.e.,

$$\hat{x} = \arg \min_{x \in \mathrm{GF}(q)} \left(V^{a,+}(x) + V^{b,+}(x) + I^{+}(x) \right). \quad (10)$$

In the EMS algorithm [4], this process is simplified first by considering that U^a already contains the summation $V^b + I$, then by pruning the $n_{m_{in}}$ elements of vector U^a to its first 3 elements. The merging of the $n_{m_{out}}$ elements of V^a and the first three elements of U^a gives V^T as $V^T[j] = (V^{T,+}[j], V^{T,\oplus}[j] = U^{a,\oplus}[j])$, $j = 0, 1, \ldots n_{m_{out}} - 1$, where

$$V^{T,+}[j] = V^{a,+}[j] + \begin{cases} U^{a,+}[0] & \text{if } V^{a,\oplus}[j] = U^{a,\oplus}[0] \\ U^{a,+}[1] & \text{if } V^{a,\oplus}[j] = U^{a,\oplus}[1] \\ U^{a,+}[2] + O & \text{otherwise.} \end{cases} \quad (11)$$

Note that $U^{a,+}[0] = 0$ and that, compared to [4], the new version omits the case where $V^{a,\oplus}[j] = U^{a,\oplus}[2]$.

Finally, the decision is made $\hat{x} = V^{T,\oplus}[k]$ with $k = \arg\min_j \{V^{T,+}[j]\}$. In practice, the offset value O is equal to 1.

Before describing the flooding algorithm, we present the NB-LDPC code implemented in the paper.

D. Code Structure

The code considered in this work is a $(N, K) = (144, 120)$ NB-LDPC defined over GF(64) with $d_v = 2$, $d_c = 12$ and code rate $r = 5/6$. This code is a Quasi-Cyclic LDPC code constructed from the complete 2×12 base matrix $\mathcal{H}$ defined as

$$\mathcal{H} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \end{bmatrix} \quad (12)$$

with an expansion factor of 12. During the lifting process of $\mathcal{H}$, every element $\mathcal{H}(i,j)$, $i = 0, 1$ and $j = 0, 1, \ldots, 11$, is replaced by the 12×12 identity matrix with a right shift rotation equal to $\mathcal{H}(i,j)$. Based on this definition, the resulting matrix H is of size $(M, N) = (24, 144)$ in GF(64). The equivalent size in binary is thus $6(M, N) = (144, 864)$.

Let us define layer one (L_1) as the set of CNs of index 0 to 11 and layer two (L_2) as the set of CNs of index 12 to 23. Then, any variable is connected to a unique CN in L_1 and a unique CN in L_2. According to the parity check equation given in (1), the 12 indexes $k(j,i)$ of the j^{th} parity check are given by $k(j,i) = j + 12i$ when $j = 0, 1, \ldots, 11$ and $k(j,i) = \mathrm{mod}\ (j + i, 12) + 12i$ when $j = 12, 13, \ldots 23$.

The GF coefficients $\{h_{j,k(j,i)}\}_{i=0,1,\ldots 11}$ of the first layer L_1 and second layer L_2 are $\{\alpha^{\mu_i}\}$ and $\{\alpha^{\beta_i}\}$ respectively, where $i = 0, \ldots, 11$, $\mu_i \in \{43, 0, 31, 4, 37, 9, 59, 14, 49, 20, 55, 25\}$ and $\beta_i \in \{0, 31, 4, 37, 9, 59, 14, 49, 20, 55, 25, 43\}$.

E. Flooding scheduling

The decoding process iterates until a maximum number of iterations ($n_{max,it}$) is reached or the M parity equations are satisfied. In each iteration, M CN and $M \times d_c$ VN updates are performed. At the end of every iteration, a decision is taken on the N VNs.

Let $l = 0, \ldots, n_{max,it} - 1$ be the iteration number. In the following, every vector message that is being processed at iteration l is appended by (l) at its exponent. The decoding process is described in Algorithm 1.

Input: Received message composed of $N = 144$ received GF symbols $Y_k = \{y_{k,0}, y_{k,1}, \ldots, y_{k,5}\}, k = 0, 1, \ldots, 143$.

Output: Decoded message $\hat{\mathbf{C}} = \{\hat{x}_0, \ldots, \hat{x}_{N-1}\}$.

Notations $U_k^{(l),u}$ refers to the message sent at iteration l from the VN k to the CN of layer u, $u = 0, 1$.

Initialization:
  $l = 0$; decoded = false.
  for $k \leftarrow 0$ **to** 143
    Compute $I_k = (I_k[i])_{i=0,1,2,3}$, and Π_i (section II-B).
    $U_k^{(0),0} = U_k^{(0),1} = I_k$
  end for

Iterative decoding:
  while $l < n_{max,it}$ **and** not(decoded)
    $l = l + 1$;
    for $j \leftarrow 0$ **to** 11 (processing of layer L_1)
      Parallel computation with $i = 0, 1, \ldots, 11$
      $\{U_{k(j,i)}^{(l-1),0}\} \xrightarrow{\text{see II.C.2}} \{V_{k(j,i)}^{(l),0}\}$
      $\left(\{V_{k(j,i)}^{(l),0}\}, I_{k(j,i)}, |Y_{k(j,i)}|, \Pi_{k(j,i)}\right) \xrightarrow{\text{see II.C.3}} \{U_{k(j,i)}^{(l),1}\}$
      $\left(\{U_{k(j,i)}^{(l),0}\}, \{V_{k(j,i)}^{(l),0}\}, I_{k(j,i)}, |Y_{k(j,i)}|, \Pi_{k(j,i)}\right) \xrightarrow{\text{see II.C.4}} \hat{x}_{k(j,i)}$
    end for
    for $j \leftarrow 12$ **to** 23 (processing of layer L_2)
      Parallel computation with $i = 0, 1, \ldots, 11$
      $\{U_{k(j,i)}^{(l-1),1}\} \xrightarrow{\text{see III.A}} \{V_{k(j,i)}^{(l),1}\}$
      $\left(\{V_{k(j,i)}^{(l),1}\}, I_{k(j,i)}, |Y_{k(j,i)}|, \Pi_{k(j,i)}\right) \xrightarrow{\text{see III.B}} \{U_{k(j,i)}^{(l),0}\}$
    end for
    decoded= true; (is $\hat{x}$ a codeword?)
    for $j \leftarrow 0$ **to** 23
      if Parity check C_j (see equation (1)) not satisfied with $\hat{x}$.
        decoded = false
      end if
    end for
  end while

Algorithm 1: Flooding scheduling of matrix H.

III. Pipelined CN-VN unit

This section presents a pipelined architecture able to perform a CN of degree $d_c = 12$ and its 12 associated VNs every CC. To do this parallel architecture, we first remind the principle of the hybrid CN architecture [17]. Then, based on this formalism, we describe the result of the optimization process of the CN architecture for the 5/6-rate $N = 144$ code in the paper. Then, an innovative method to merge VN and CN processing is presented.

A. Principle of the hybrid CN architecture

In [17], the authors present the principle of the hybrid CN architecture. The three main functions performed by this CN are recalled: the presorting, the ECN processing and the decorrelation using the Valid Syndrome Vector (VSV).

1) Presorting: The preliminary step of CN processing is the presorting block that leads to a significant reduction of the CN computations. In [21] the authors proposed the sorting of the CN input vectors based on the LLR value of the second GF element. This sorting polarizes the reliability of the input vectors and classifies them into two sets: high reliability and low reliability. As described in Section II-B, the first element (i.e., the most reliable symbol) is always having a zero LLR value. The sorting criteria is the following: the higher the LLR value of the second element, the higher the reliability of the vector. In other terms, a *big* difference between the first and second LLR values indicates that the competition between the GF symbols in the vector will clearly favor the first one and the rest will *rarely* contribute to the final decision on the codeword. Discarding (or eliminating) them leads to computational reduction without any performance loss. As a consequence, presorting *helps* the CN to concentrate its processing effort on low-reliability vector messages.

Fig. 1 shows the presorting principle for $d_c = 4$, $n_{min} = 4$ and GF(64). The second most reliable LLR values $\{2, 5, 1, 3\}$ of each message (U_0 to U_3) are considered as inputs to a sorter block. Then, the d_c input messages are switched based on the indexes ψ. The high reliability messages are concentrated in one region and dashed elements are discarded prior to the CN processing. More details on the presorting technique are presented in [20], [21] and [23].

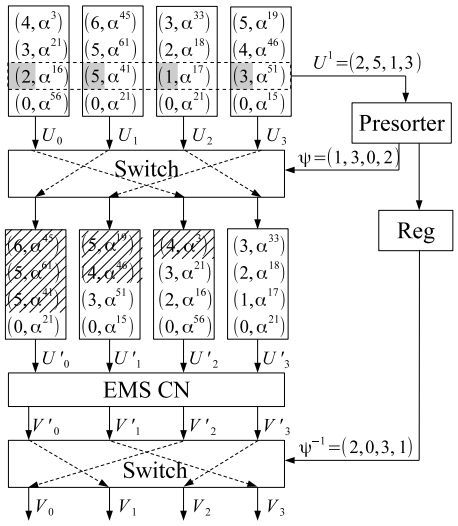

Fig. 1. Presorting principle of the EMS-based CN.

2) ECN: For the implementation of the CN processing, equation (6) is implemented in a simplified way using the hybrid CN architecture defined in [17]. The whole CN architecture is characterized graphically in Fig. 2.a). Let us describe, from top to bottom, the graphical elements used in this design and the corresponding processing.

First, the number of elements of U'_i, $i = 0, \ldots, 11$, that enter the CN is indicated by the number of circles below it. For example, only $U'_0[0] = (U'^{+}_0[0], U'^{\oplus}_0[0])$ out of $n_{m_{in}} = 4$ elements of message U'_0 enters the CN and the first 3 elements $\{U'_8[0], U'_8[1], U'_8[2]\}$ of message U'_8 enter the CN. The line of multipliers on the top indicates that each GF value of U'_k are multiplied by the GF coefficient h'_k. The outputs of the multipliers enter a datapath composed of a network of ECNs. Each ECN performs the bubble check algorithm. Let us give the key to understand the processing performed by the generic ECN given in Fig. 2.b). An ECN receives two input vectors A and B of size n_a and n_b given by the number of circles (or bubbles) respectively in the first column and in the first row ($n_a = 4$ and $n_b = 3$ in Fig. 2.b)). It generates an output message C of size n_c. Note that in Fig.2.a), the output size is implicitly defined as the number of inputs (i.e., number of vertical bubbles) of the next ECN. A circle in position (t_0, t_1), $t_0 = 0, \ldots, n_a - 1$ and $t_1 = 0, \ldots, n_b - 1$, means that $U_a[t_0]$ and $U_b[t_1]$ are added to generate a couple $(U^+_a[t_0] + U^+_b[t_1], U^{\oplus}_a[t_0] \bigoplus U^{\oplus}_b[t_1])$. The n_c bubbles of minimum LLR sorted in increasing order constitute the output vector of the ECN. The VSV is appended with a boolean value that indicates whether $U_c[t_2]$, $t_2 = 0, \ldots, n_c - 1$, has been generated with $U_b[0]$ or with $U_b[t_1]$, $t_1 > 0$. Bubbles in dark color append a false Boolean value to the corresponding position in the VSV vector.

The dashed line labeled T+6 at the output of ECN9 and ECN10 indicates that three pipeline stages are inserted in these blocks. This pipeline labeling starts at T+0, where the first pipeline stage that represents the input registers storing the input of the pre-sorting architecture is inserted, followed by three pipeline stages inserted in this pre-sorting architecture (see Fig. 4). This labeling T+i will continue through the CN-VN architecture indicating the position at which each pipeline stage is inserted. The three last ECNs (ECN11, ECN12 and ECN13) are slightly simplified compared to the other ECNs because all the bubbles are output without any sorting. In fact, one of the main ideas in the architecture is to save hardware complexity by postponing the sorting operation in the VN processing. Since no sorting is performed, the default value D of the check to variable message cannot be determined. Thus, we propose to empirically set it to the LLR of a fixed bubble position indicated by D_g, D_{10} and D_{11} in ECN11, ECN12 and ECN13, respectively. Note that the size of the output message S of ECN11 is $n_S = 20$ while the size of the message of ECN12 and ECN13 is $n_{FB} = 16$. Any element of S is given by the summation of all the incoming messages, a decorrelation process is thus required [17]. It suppresses the GF symbol $U'^{\oplus}_i[0]$ from $S^{\oplus}[t]$ if $U'^{\oplus}_i[0]$ contributes in computing $S^{\oplus}[t]$ to generate the i^{th} output thanks to the GF adder (addition and subtraction are equivalent in the GF domain), where $i = 0, \ldots, 11$, $t = 0, \ldots, n-1$ and $n \in \{n_S, n_{FB}\}$. Otherwise, if $U'^{\oplus}_i[0]$ does not contribute in computing $S^{\oplus}[t]$, the Decorrelation block (DB) associated to U'_i saturates the LLR value $S^+[t]$. This is done thanks to the VSV vector that is being checked by the DB. The final multiplication is applied on the GF value to compute the output message V'_i. In more details, $V'^{\oplus}_i[t] = (S^{\oplus}[t] - U'^{\oplus}_i[0]).h'^{-1}_i$ and $V'^{+}_i[j] = S^+[t]$. These operations are performed in one CC to have three CCs latency in total to perform the CN.

B. CN-VN Processing

Fig. 3 shows the architecture of the proposed parallel pipelined CN-VN. The inputs of CN-VN are the

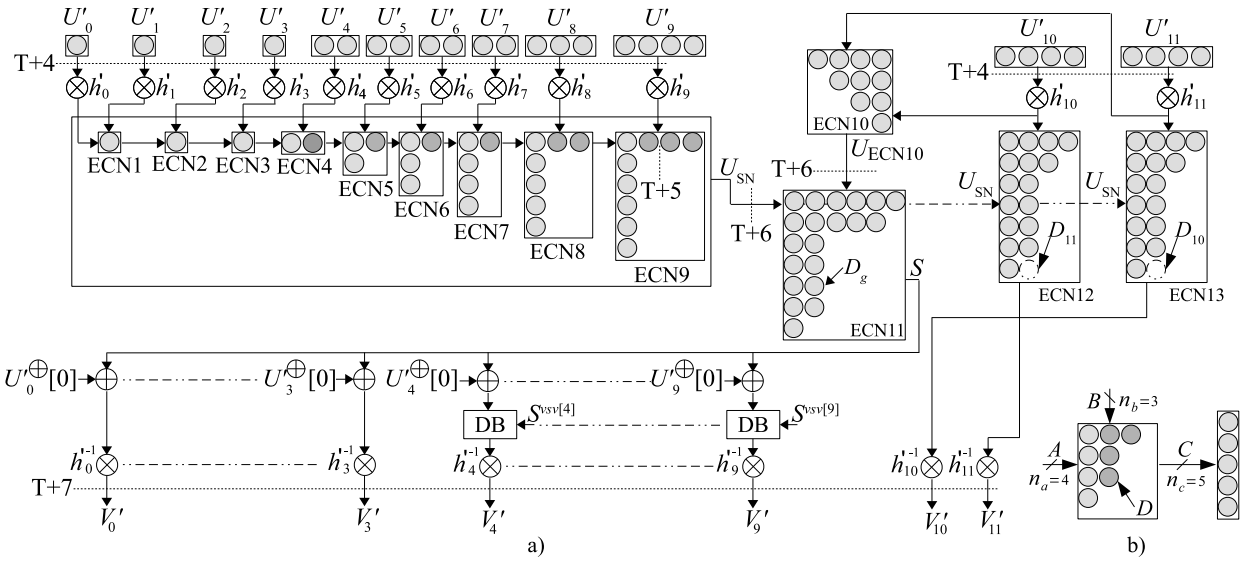

Fig. 2. a) High level CN architecture and b) ECN in case of $n_a = 4$, $n_b = 3$ and $n_c = 5$.

intrinsic information $\tilde{I} = \{I, \tilde{I}_e\}$ ($I = \{I_0, \ldots, I_{11}\}$ and $\tilde{I}_e = \{\{I_0[0], \ldots, I_{11}[0]\}, \{|Y_0|, \ldots, |Y_{11}|\}, \{\Pi_0, \ldots, \Pi_{11}\}\}$), the GF coefficients and their inverse $(h, h^{-1}) = \{(h_0, h_0^{-1}), \ldots, (h_{11}, h_{11}^{-1})\}$ and the extrinsic messages $U^a = \{U_0^a, \ldots, U_{11}^a\}$. The first stage of the proposed joint CN-VN unit starts in a similar way than the hybrid architecture, except that all messages are received in parallel: first, all the inputs are permuted using the indexes Ψ obtained by the presorting block, then the hybrid CN is performed.

The presorting receives $\{U_0^+[1], \ldots, U_{11}^+[1]\}$ to generate the indexes $\Psi = \{\psi[0], \ldots, \psi[11]\}$ for $d_c = 12$ based on the presorting principle shown in Fig. 1, thus $U^+_{\psi[0]}[1] \leq U^+_{\psi[1]}[1] \leq \cdots \leq U^+_{\psi[11]}[1]$. The architecture of the parallel pipelined presorting block is shown in Fig. 4. This architecture is inspired from [24]. Every comparator-swap receives two inputs where the one that is having minimum LLR value will be positioned at the lower output and the one with maximum LLR value will be positioned at the higher output. The indexes of the inputs are also shifted to provide at the output of the presorting block, the indexes of sorted minimum values $\psi[i]$, $i = 0, \ldots, d_c - 1$.

The presorting block consists of 42 comparator swap components and three pipeline stages are inserted in the presorting as shown in Fig. 4. Based on the Ψ values, the inputs of the CN-VN are switched using the permutation (Perm. Ψ) block. Every $\psi[i]$ is coded on 4 bits since there are $d_c = 12$ positions. Thus, the size of Ψ is $4 \times 12 = 48$ bits. The input messages are permuted in one CC. After that, the CN is performed. The specification of the CN was defined in section III.A.

After the CN processing, the VN and DM blocks operate in parallel to make the VN update and the decision on every input. There are 12 VN and 12 DM blocks associated to a

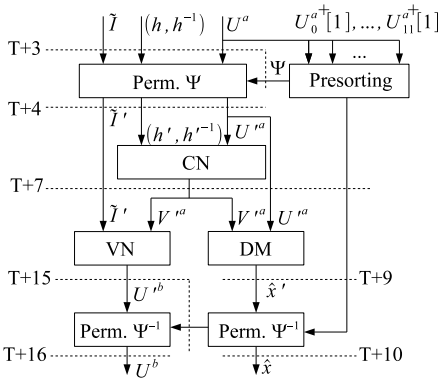

Fig. 3. CN-VN architecture.

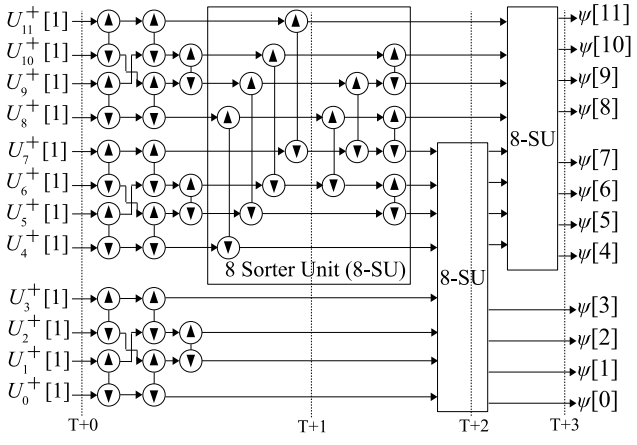

Fig. 4. Presorting architecture.

CN. The VN and DM processing were described in section II.C. Figure 5 illustrates the parallel architecture of the VN. The eLLR block generates the intrinsic LLR value $I^+(V_i'^{a\oplus})$, thanks to $|Y_i'|$ and $I_i'^{\oplus}[0]$, then the value is added to $V_i'^{a+}$ to

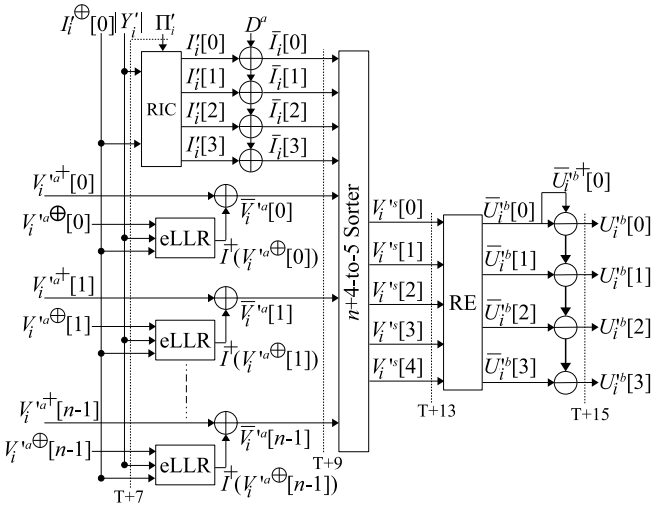

Fig. 5. VN architecture

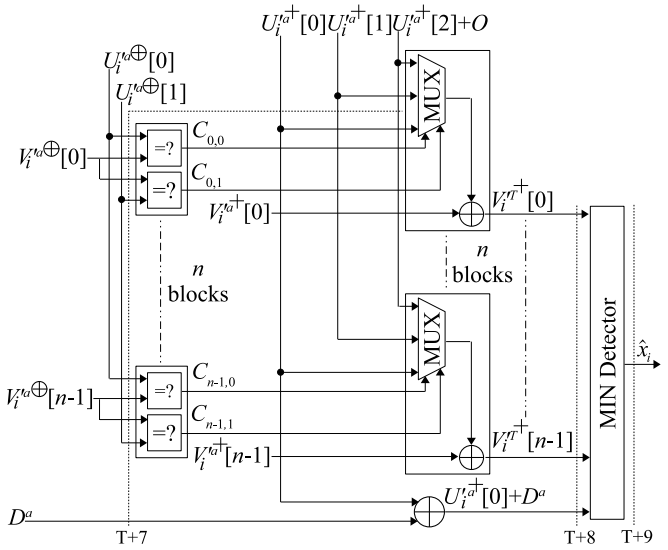

Fig. 6. DM architecture

generate $\bar{V}_i'^{a+}$. The Regeneration of Intrinsic Candidates (RIC) block regenerates the intrinsic candidates $\{I_i'[0], \ldots, I_i'[3]\}$. Then, the offset value $D^a \in \{D_g, D_{10}, D_{11}\}$ associated to V_i' is added on $\{I_i'[0], \ldots, I_i'[3]\}$ to generate $\bar{I}_i^+$. In section II.C, we showed that the output is generated by detecting the most reliable not redundant GF symbols from $\{V_i'^a[0], \ldots, V_i'^a[n-1], I_i'[0], \ldots, I_i'[3]\}$. To reduce the complexity, the sorting and the Redundant Elimination (RE) operations are separated. First, the vector $V_i'^s$ of $n_{m_{in}} + n_\delta$ couples having the lowest LLR values are detected from $\{V_i'^a[0], \ldots, V_i'^a[n-1], I_i'[0], \ldots, I_i'[3]\}$, then the outputs are generated by detecting, from $V_i'^s$, the $n_{m_{in}}$ couples without redundant GF symbols. In this work $n_\delta = 1$. Thus, the $n+4$-to-5 Sorter block, $n \in \{n_{FB}, n_s\} = \{16, 20\}$, generates $V_i'^s$ that is having the $4+1=5$ couples of lowest LLR values among $\{V_i'^a[0], \ldots, V_i'^a[n-1], I_i'[0], \ldots, I_i'[3]\}$. The RE block generates $U_i'^b$ by detecting the 4 couples from V'^s that are different in terms of GF values and having lowest LLR values. Finally, the normalization of $\bar{U}^{b,+}$ (9) is performed. Eight pipeline stages are inserted in the VN architecture.

Figure 6 shows the parallel architecture of the DM block. It contains $2n$ comparators operating in parallel to check whether $V_i'^{\oplus}[t_0] = U_i'^{\oplus}[t_1]$ or not, where $t_0 = 0, \ldots, n-1$ and $t_1 = 0, 1$. We compare $V_i'^{\oplus}[t_0]$ to $U_i'^{\oplus}[0]$ and $U_i'^{\oplus}[1]$ as described in equation (11). The 3-to-1 MUXs operate as follows: if $V_i'^{\oplus}[t_0] = U_i'^{\oplus}[t_1]$ then the output is $U_i'^{+}[t_1]$, otherwise, the output is $U_i'^{+}[2]+O$. The output of every 3-to-1 MUX is added to its associated $V_i'^{+}[t_0]$. Only the couple $U_i'[0]$ is considered from the set $\{U_i'^{\oplus}[0], U_i'^{\oplus}[1], U_i'^{\oplus}[2]\}$ where its LLR value is added to the default value D^a. Finally, the MIN Detector block selects from $\{V_i'[0], \ldots, V_i'[n-1], U_i'[0]\}$ the decided symbol $\hat{x}_i'$ having the lowest LLR value.

The last operation in CN-VN is the inverse permutation performed using Ψ^{-1} to reorder $U'^b = \{U_0'^b, \ldots, U_{11}'^b\}$ and $\hat{X}' = \{\hat{x}_0', \ldots, \hat{x}_{11}'\}$ to their original order. This block is having one pipeline stage and hence 16 CCs and 10 CCs are the total latency to generate U^b and $\hat{X}$ respectively.

IV. Proposed parallel and pipelined decoder

This section describes the global architecture of the decoder as well as the inputs/outputs of each block. The memory system and the timing diagram of the decoding process are discussed. The global decoder is based on the CN-VN unit described in Section III which has been customized to offer the best performance-complexity trade-off for the considered code. This CN-VN unit can be modified to meet the specifications of any other NB-LDPC code and thus design the associated decoder. In the following, we use the index $k(j,i)$, $i = 0, \ldots, 11$, $j = 0, \ldots, M-1$ and $k(j,i) \in \{0, \ldots, 143\}$, to refer to a quantified symbol $Y_{k(j,i)}$ for a specific $\text{VN}_{k(j,i)}$. For instance, when CN_0 is being processed after the extension of the prototype matrix $\mathcal{H}$, the set $\{\text{VN}_0, \text{VN}_{12}, \text{VN}_{24}, \text{VN}_{36}, \text{VN}_{48}, \text{VN}_{60}, \text{VN}_{72}, \text{VN}_{84}, \text{VN}_{96}, \text{VN}_{108}, \text{VN}_{120}, \text{VN}_{132}\}$ is considered and hence $k(0,0) = 0$, $k(0,1) = 12$, $\ldots$, $k(0,11) = 132$ are the indexes associated to VN_0, VN_{12}, $\ldots$, VN_{132} respectively. These indices along with the associated GF symbols h_i are indicated by the PCM of the code.

A. Architecture overview

The architecture of the global decoder is shown in Fig. 7. The 144 symbols of a received frame are input in 18 CCs by group of 8 symbols. The input order is given by the layer L_1 order of the parity check matrix when it is read line by line by block of 8 symbols (see Fig. 8). Thus, when the input *start* is set to one to indicate the arrival of a new frame, the first input Y is equal to $\{Y_0, Y_{12}, Y_{24}, Y_{36}, Y_{48}, Y_{60}, Y_{72}, Y_{84}\}$ (all of them belong to CN_0). The second CC, Y is equal to $\{Y_{96}, Y_{108}, Y_{120}, Y_{132}, Y_1, Y_{13}, Y_{25}, Y_{37}\}$ (the first 4 symbols belong to CN_0 and the last 4 symbols belong to CN_1) and so on. The size of the input *Y* is 288 bits (8 symbols, each symbol composed of $m = 6$ LLR values quantified on $b = 6$ bits). The outputs of the global architecture are the signal *Ready* that

indicates both the end of the decoding process (output $\hat{C}$ valid) and the availability of the decoder to receive a new frame. The decoded frame $\hat{C}$ is composed of the $K = 120$ GF(64) information symbols of the transmitted message, which gives a total size of $120 \times m = 720$ bits. The output signal $decod\_ok$ is set to one when the decoding process is succeeded, i.e., when all parity checks are satisfied.

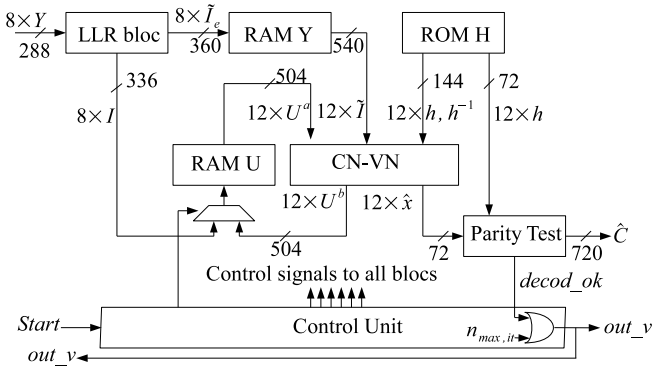

Fig. 7. Global decoder architecture.

The internal structure of the decoder is composed of 6 blocks. The LLR block performs the processing described in section II.B. From the 8 input symbols, the LLR block generates the 8 associated intrinsic vector I (total size $8n_I = 336$ bits) that are directly stored in RAM U and the 8 associated pre-processed intrinsic vector $\tilde{I}_e$ (total size of $8n_{\tilde{I}} = 360$ bits) that are directly stored in RAM Y. The architecture of the LLR block is composed of 8 parallel symbol LLR generators, each symbol generator receiving the 6 binary LLRs of a symbol and generating vector I and $\tilde{I}_e$. The reader can refer to [22] for more details about the LLR generator architecture. From RAM Y, RAM U and ROM H (ROM H contains only the GF coefficients of the parity checks), the 12 information vectors related to a given CN are sent to the CN-VN component. After processing, vector U^b is stored back into ROM U while decision vector is sent to the parity test block. This block verifies whether the current decoded frame is a codeword or not based on (1). Finally, the control unit synchronises the components and generates the read/write instructions of memory blocks.

Let us describe in more details the internal structure of the memory blocks (RAM U, RAM Y and RAM H), the Parity Test block and the overall control block among with the timing diagram. Note that the CN-VN is already detailed in section III.

B. Memory blocks

Recalling Section II.D, after the extension of the prototype matrix $\mathcal{H}$ (see (12)), the obtained PCM H is of size $(M, N) = (24, 144)$ with two layers L_1 and L_2. There are three types of memories in the decoder: Extrinsic RAM (RAM U), Intrinsic RAM (RAM Y) and the ROM (ROM H) that stores the h coefficients of the PCM. Fig. 8 shows the structure of the Extrinsic RAMs. The RAMs are structured according to the PCM matrix, i.e., according to the connections of the CNs with the VNs. A value $j = 0, \ldots, 143$ in a cell represents the index of the VN_j. Every RAM_i, $i = 0, \ldots, 11$, stores 24 extrinsic messages of 12 successive VNs, with each VN connected to a CN in L_1 and a CN in L_2. For instance, RAM_2 stores the extrinsic messages associated to $VN_{24}, VN_{25}, \ldots, VN_{35}$ that are connected to $CN_0, CN_1, \ldots, CN_{11}$, respectively from the first layer L_1, and to $CN_{22}, CN_{23}, \ldots, CN_{21}$, respectively from the second layer L_2. When processing CN_i, $i = 0, \ldots, 23$, the messages are read in parallel from RAMs as $\{RAM_0[i], RAM_1[i], \ldots, RAM_{11}[i]\}$. The read address A_r is a counter varying from 0 up to 23 periodically. The set of inputs $\{U_0^b, \ldots, U_{11}^b\}$ coming from the CN-VN is stored in their appropriate positions in the RAMs. For example, let $CN_0 \in L_1$ be the CN that is being processed. We have $A_r = 0$ and hence $\{U_0^a, \ldots, U_{11}^a\} = \{RAM_0[0], \ldots, RAM_{11}[0]\}$. In other words, the VNs $\{VN_0, VN_{12}, \ldots, VN_{132}\}$ are being processed. Once processed, the results $\{U_0^b, \ldots, U_{11}^b\}$ are written into the associated VNs: U_0^b is associated to VN_0 in the second layer and hence it will be stored in $RAM_0[12]$; U_1^b is associated to VN_{12} and hence it will be stored in $RAM_1[23]$, ...; U_{11}^b is associated to VN_{132} and it will be stored in $RAM_{11}[13]$. Therefore, each RAM_i requires its own write address A_{w_i}, $i = 0, \ldots, 11$. Every cell in a RAM stores 42 bits: 4 GF symbols (each of 6 bits) and 3 non-zero LLR values (each of 6 bits). Furthermore, since the latency of the CN is 16 CCs, some updated message VNs in L_1 are directly used in L_2. These messages are highlighted in grey color in Fig. 8. In other words, the decoding process is not completely flooding (recall section II.E).

The intrinsic RAMs store the information related to the intrinsic LLR messages of the $N = 144$ VNs. These VNs are organized in RAM blocks similarly to RAM L_1 part shown in Fig. 8. For instance, the intrinsic messages of VN_0 are stored in the first cell of the first RAM block ($RAM_0[0]$), while the intrinsic messages of VN_{50} are stored in the third cell of the fifth RAM block ($RAM_4[2]$), and so on. The required information is concatenated to be stored in each cell of length $n_{\tilde{I}} = 45$ bits. Every intrinsic RAM has its own read address and write address.

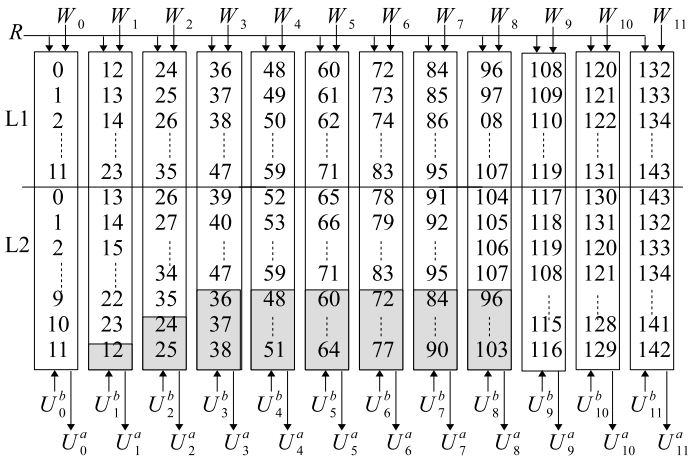

Fig. 8. Extrinsic RAM Banks in RAM ROM Banks block.

The non-zero elements of the PCM and their inverse are

stored in a ROM block. Due to the specific code construction (Section II.D.), the ROM has only 2 words, one for each layer, where each word is of size equal to $(6 \times 2) \times 12 = 144$ bits since every non-zero GF value h_i and its inverse h_i^{-1} consists of 6-bit words, and $i = 0, \ldots, 11$.

It is interesting to evaluate the memory bandwidth of the proposed architecture per iteration and per symbol, then, per bit. In an iteration, a VN is implied in two CNs. For each CN, it reads $\tilde{I}$ in the intrinsic RAM (thus $n_{\tilde{I}} = 45$ bits) and reads the U^a message in the extrinsic RAM (42 bits from 4 GF symbols and 3 non-zero LLRs) and write back the U^b message (thus 42 bits) in the extrinsic RAM. Thus, the total number of read/write operations to process a symbol during an iteration is $2(45+2\times42) = 258$ bits. Since a symbol contains 6 bits of information, it gives in average 43 bits of read/write memory access per codeword bit per decoding iteration. This number should be compared to a binary LDPC decoder. Assuming a $d_v = 3$ and a soft-output based CN architecture [25], with the soft-output coded on 8 bits and the extrinsic on 6 bits, then each iteration will require $d_v \times 2(8+6) = 84$ bits of read/write memory access per message bit per decoding iteration. The natural conclusion that may goes against the common belief is that **NB-LDPC code can decrease the memory bandwidth by almost 50% compared to binary LDPC code**. The size of the memory is also reduced from $(8 + 3 \times 6) = 26$ bits per message bits for LDPC down to $(45 + 2 \times 42)/6 = 21.5$ bits in average per message bits for the NB-LDPC.

C. Parity Test block

The Parity Test block performs the test of all the $M = 24$ parity check equations based on equation (1). During the first 12 CCs when the CN-VN outputs the decision taken during the processing of layer L_1, the parity checks are tested on the fly while the 144 decoded symbols of the decoded codeword $\hat{C}$ are stored in a register bank. During the next 6 CCs, the 12 parity checks of the second layer are tested thanks to two parity checks working in parallel. If the $M = 24$ equations are satisfied, the decoding process ends with the decoded codeword. The Boolean *decod_ok* is sent to the control unit to stop the decoding process of the current frame and starts a new frame. The output vector $\hat{C}$ is also available at the output of the block for the case where the maximum number of iterations is reached and the decoding process ends.

D. Control unit and decoding scheduling

The control unit block controls the read/write operations from/to the RAM ROM Banks. A *start* signal indicates the arrival of the observed symbols and hence the control signals of the RAM ROM Banks are generated based on a counter in the Control Unit (CU).

The control of the decoder works with a periodicity of $2 \times 24 = 48$ CCs as shown in Fig. 9. In this figure, four different frames in different phases are presented. Frame $k-2$ (white color) that is decoded at cycle -1, frame $k-1$ (blue color) that is still being processed, frame k (grey color) that is being received at cycle 0 just after decoding frame $k-2$ and frame $k+1$ (cyan color) that is being received after decoding frame $k-1$. The $N = 144$ received symbols Y of frame k are received in 18 CCs from cycle numbers 0 up to 17 by a group of 8 symbols and sent directly to the LLR block. After two CCs of latency, the LLR block generates all the side information related to the received symbol ($\{I, \tilde{I}_e\}$). The data is stored in their appropriate location in the intrinsic memory RAMs and in the extrinsic memory RAMs. At cycle index 18, all the intrinsic information of the VNs connected to CN_0 are stored in memory. The processing of the layers L_1 and L_2 for frame k starts at cycle number 19, taking 24 cycles to complete at cycle number $19 + 24 - 1 = 42$.
Then, 10 cycles after the beginning of the processing of the first CN of frame k, i.e., at CC number $18+9 = 27$ the decision on the VNs associated to the first CN are output ($\hat{X}$). After 18 CCs (see Parity Test block description), a codeword is said to be decoded ($decod\_ok = 1$) if all the decisions generated by layer L_1 verify all the M parity CNs (at CC number 47), just in time to start again the loading of a new codeword at cycle number 48. As seen in Fig. 9, the processing of a given frame requires the utilization of a given component at most 24 CCs. Among the 48 cycles of processing of a given frame, there are at least 24 cycles during which some components of the CN-VN unit are in idle mode and can be used to process another frame. Thus, two frames will be always present in the CN-VN unit to be processed in parallel. Note that, since the number of iterations may differ from one frame to another, the order of the decoded frames generated at the output of the decoder may hence differ from the order they entered the decoder with.

The number of CCs to decode a frame is thus $48 \times n_{it,f}$, where $0 < n_{it,f} \leq n_{max,it}$ is the number of iterations to decode a frame. Since two frames are decoded in parallel, the average number of CC to decode a codeword is $24 \times n_{av,it}$ CCs.
This parallelism in the simultaneous processing of two consecutive frames requires the duplication of the intrinsic and extrinsic RAMs to store the data of two frames. To summarize, looking at the global execution of the decoder, the 19 CCs latency for preparing the data (shown in Fig. 9) and the 16 CCs latency of the CN are not considered when evaluating the execution time of the decoder (which has a direct impact on the throughput rate). We also note that without the duplication of the RAMs, that allowed the parallel processing of two consecutive frames, the 16 CCs latency of the CN has to be considered as a part of the execution time at each iteration. This is due to the fact that CN_{23} and CN_0 share the same variable VN_{12}, which prevents the start of the second iteration before the processing of CN_{23} is ended. Therefore, $M = 24$ CCs is the latency of one iteration.

V. Simulation results

As described in Section III.B, some bubbles from the Hybrid CN are eliminated to reduce the computational complexity but at the cost of a slight performance loss. In order to compensate this performance degradation, the maximum number of iterations is increased to 30 iterations. Fig. 10 shows simulation results for the BP decoder [5], the well-known FB-

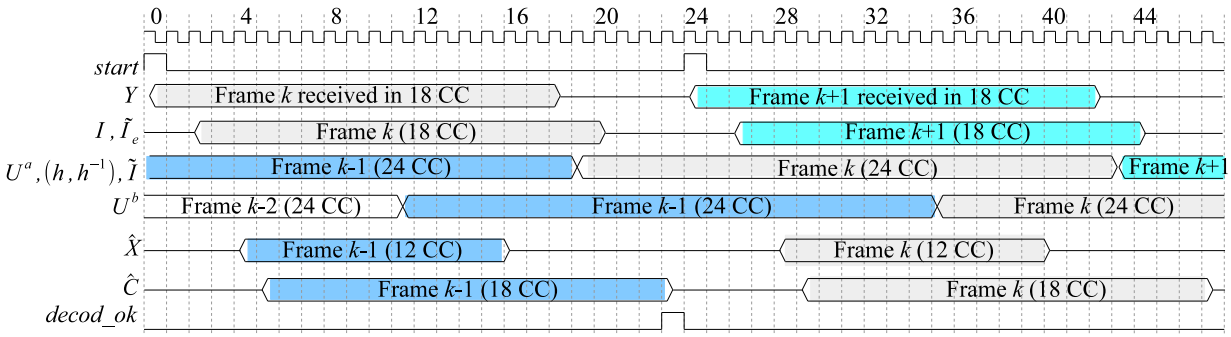

Fig. 9. Timing diagram of the overlapping phase.

CN EMS decoder [26], proposed decoder, the FB-CN Min-Max decoder [27], and the binary Sum-Product (SP)-based decoder. The BP, the FB-CN EMS and the FB-CN Min-Max NB-LDPC code have the same parameters: $K = 120$ GF(64) symbols, $N = 144$ GF(64) symbols and CR $= 5/6$ (equivalently, $K = 720$ bits and $N = 864$ bits). The SP-based B-LDPC code is of length $N = 864$ bits, $K = 720$ bits and CR $= 5/6$ but designed over GF(2). The BP, the FB-CN EMS and the FB-CN Min-Max decoders are simulated using layered scheduling while the proposed decoder, in its hardware version, result in a "partially layered" scheduling. This partially layered scheduling is due to the fact that the new parallel decoder starts a new CN processing at each clock cycle, which leads to reach the second layer of CNs without having fully updated the VNs with the check node messages of the first layer. Note that the addition of idle clock cycles could solve the issue but would reduced the decoding throughput. In the proposed architecture, the CN are directly computed without waiting for updated data. Thus, some entries of the 12 CN of the second layer take benefit of the updated data of the current iteration (shown in grey in Fig. 8) while the others take benefit of updated data from the previous iteration, as it is the case for flooding scheduling. In summary, the decoder has a convergence speed between the convergence speed of the layered scheduling and the convergence speed of the flooding scheduling.

We consider Monte Carlo simulations under the AWGN channel, BPSK modulation and the LLR values quantized on $b = 6$ bits. A performance loss of 0.08 dB is observed between the proposed decoder with 30 iterations and the references floating point BP and fixed point FB-CN decoding algorithms with 8 iterations, $n_m = 16$ and $n_{op} = 18$. Although the proposed decoder is implemented with a maximum number of iterations equal to 30, it is the average number of iterations that will be taken into consideration to determine the average decoding throughput. This will be discussed in more details in the next section. Comparing the proposed decoder with the FB-CN Min-Max layered decoding for $n_{max,it} = 8$, the proposed decoder shows slightly better performance than the Min-Max algorithm in the waterfall region ($10^{-1} \leq$ FER $\leq 10^{-6}$). When compared to its binary LDPC counterpart, the proposed decoder presents a gain of 0.3 dB at a FER of 10^{-3}. It is worth mentioning that the NB-LDPC code offers an important advantage in terms of spectrum efficiency since high order modulations are suitable to be used with NB-LDPC codes designed over GF($q > 2$), where there is no need for iterative demodulation. To evaluate performance in a short time, the complete digital communication chain is implemented on an FPGA device. The source, encoder, channel and decoder are implemented using VHDL. The source generates random bits that are encoded, BPSK modulated, affected by an AWGN, then demodulated and decoded. A hardware discrete channel emulator is implemented to emulate the AWGN channel. We used the Xilinx KC705 FPGA DevKit containing a Kintex 7 where the simulation and emulation results are matched.

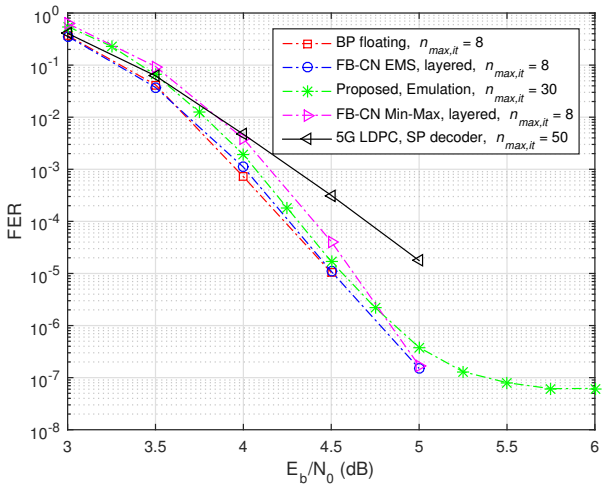

Fig. 10. FER performance for a (144, 120) NB-LDPC code over GF(64): Proposed decoder vs FB CN-based decoder and (864, 720) B-LDPC code over GF(2) SP decoder.

As Fig. 10 shows, the error floor of the proposed decoder starts from $E_b/N_0 = 5.25$ db where FER $= 10^{-7}$. This is due to the significant simplifications that have been done on the EMS algorithm in order to reduce its complexity. These simplifications are mainly: 1) the predefined offset value; 2) the new redundant elimination process in the VN block where it is split up into two phases (sorting then redundant suppression); 3) the reduction of the considered intrinsic symbols in the DM block from 3 down to 2 symbols; 4) the significant reduction of the number of bubbles in the ECN units.

VI. Implementation results

This section discusses the throughput calculation and the post-synthesis results on 28-nm FDSOI technologies. Since the decoding throughput is highly dominated by the average number of iterations $n_{av,it}$, we study its variation versus the signal-to-noise ratio (i.e., E_b/N_0). The output decoding throughput T calculation is expressed in Giga bits per second (Gbps) as

$$T = \frac{\log_2(q) \times K}{L_{CN} \times n_{av,it} \times M} \times F_{clk} \times 10^{-3} \text{ (Gbps)}, \qquad (13)$$

where L_{CN} is the latency of the CN-VN and F_{clk} the clock frequency of the design expressed in MHz.

For FPHCN, the synthesis results gives a maximum clock frequency $F_{clk} = 900$ MHz with a latency $L_{CN} = 1$. For the serial hybrid architecture, the maximum clock frequency was given at 800 MHz in [17]. A new synthesis of the hybrid architecture performed by the authors increases the maximum clock frequency up to 1000 MHz. This updated value is considered in the comparison. The latency L_{CN} of the hybrid architecture remains unchanged to the value $L_{CN} = 41$.

Table II compares the average decoding throughput between [17] and FPHCN at different E_b/N_0. Compared to [17], FHPD decoding througput can be increased by a factor between 12.3 up to 20.

TABLE II
AVERAGE NUMBER OF ITERATIONS AND THROUGHPUT RATE COMPARISON OF [17] AND THE PROPOSED FPDH.

E_b/N_0 (dB)		2	3	3.5	4	4.5	5
[17]	$n_{av,it}$	10	6.77	3.1	1.74	1.26	1.05
	T (Gbps)	0.073	0.1	0.24	0.42	0.58	0.7
This work	$n_{av,it}$	30	18.3	7.25	3.4	2.45	1.93
	T (Gbps)	0.9	1.4	3.7	7.9	11	14
Factor gain		12.3	14	15.4	18.8	18.9	20

A comparison of the FPHCN implementation and three state of the art decoders [7], [16], [17] is presented in Table III. In order to take into account the transistor size reduction between the θ-nm technology and the 28-nm, the clock frequency (and thus, the decoding throughput) is scaled by a factor $\theta/28$. Note that, in practice, the maximum frequency of an ASIC design is limited. For example, a maximum clock frequency of 1000 MHz is considered in the European Project H2020 EPIC [28] to compare error correcting decoder architectures. Nevertheless, this limitation is not considered in this work. The hardware complexity C is expressed in millions of NAND gates, the input decoding throughput T (in Gbits/s or Gbps) and the hardware efficiency E defined as the ratio $E = T/C$ in Gbps per million NAND gates.

All these decoders achieve ultra-high throughput rate thanks to parallelism. Let us first compare the FPHCN to the architecture proposed in [7]. The 1.22 Gbps throughput rate shown in [7] is obtained at $E_b/N_0 = 5.0$ db where $n_{av,it} = 11.71$. However, The FPHCN architecture provides higher throughput rate starting from $E_b/N_0 > 3.3$ dB and shows a better hardware efficiency in a factor ranging from 1.05 up to 16.9.

TABLE III
COMPARISON OF STATE OF THE ART NB-LDPC DECODERS (ASICs).

	[7]	[16]	[17]	FPHCN
Techno (nm)	65	90	28	28
N (symbols)	160	1512	144	144
CR	1/2	7/8	5/6	5/6
Decoding Algorithm	EMS Flooding	BS-TMM Layered	EMS Layered	EMS partially layered
Iterations	10-30	8	1-10	1-30
F_{clk} (MHz)	700	360	1000	900
C (NAND)	2.78 M	2.09 M	0.21 M	0.79 M
T (Gbps)	1.22	1.4	0.073→0.7	0.9→14
T (Gbps) 28 nm	2.9	4.5	0.073→0.7	0.9→14
E (T/C)	1.04	2.15	0.35→3.5	1.1→17.7

Since the proposed FPHCN architecture and [16] does not correspond to the same code, the result of table III shows roughly the same order of hardware efficiency: in the worst case, [16] has a double hardware efficiency than FPHCN. If average number of iterations is considered, then highest hardware efficiency of [16] is obtained when considering only one decoding iteration instead of 8, i.e., an hardware efficiency of $2.15 \times 8 = 17.2$ Gbits/s per million NAND gates, which is lower that the proposed hardware efficiency of FPHCN with 1.93 iterations in average for $E_b/N_0 = 5$ dB (see table II) that gives an hardware efficiency of 17.7 Gbits/s per million NAND gates.

Finally, comparing FPHCN with its serial counterpart shown in [17], the FPHCN provides much higher throughput rate where the factor gain varies from 12.3 up to 20 thanks to the high order of parallelism. The significant improvement in terms of throughput rate is reflected on the hardware efficiency in favor of the FPHCN approach as shown in Table III.
Table IV shows the number of processed CNs per second $T_{CN} = \frac{F_{clk} \times 10^6}{L_{CN} \times n_{av,it}}$ (CNs/s) and the hardware efficiency $E_{CN} = T_{CN}/C$ (CNs/s/Mgate) of the CN-VN block in case of FPHCN and [17] ($d_c = 12$). The parallel CN-VN block provides much higher T_{CN} that varies between 33333 CNs/s and 518135 CNs/s, while T_{CN} obtained with the serial CN-VN is varying between 2439 CNs/s and 24390 CNs/s. Even though the area consumption of the serial CN-VN is about 10 times lower than the parallel CN-VN, the hardware efficiency of the parallel CN-VN is higher than the serial approach presented in [17] as shown in Table IV.

TABLE IV
COMPARISON OF THE PARALLEL AND SERIAL CN-VN BLOCK (ASICs).

	CN-VN FPHCN	CN-VN [17]
Iterations	1-30	1-10
C (NAND)	0.38 M	0.032 M
T_{CN} (CNs/s)	33333 → 518135	2439 → 24390
E_{CN} (CNs/s/Mgate)	87718 → 1363513	76218 → 762187

It is worthy to mention that the CN-VN in case of FPHCN constitutes 48.1% of the total complexity of the decoder. The remaining 51.9% are related to the LLR generator blocks, the DM blocks, the parity test blocks, the RAM/ROM blocks and the control unit.

VII. Conclusion and perspectives

This paper was dedicated to an ultra-high-throughput EMS NB-LDPC decoder implementation based on a Fully Parallel Hybrid Check Node architectures. We particularly focused on a GF(64) $(144, 120)$ code with high rate (CR $= 5/6$). A number of architectural strategies made possible the 10 Gbps throughput for high SNR, which represents a throughput efficiency gain in the order of 6 to 50 (depending on the SNR) compared to [17]. Beside the careful optimization of the number of bubbles in each ECN, several original ideas have been presented in the paper to optimize the prior hybrid architecture. The main idea is to merge the CN and the VN processing with the suppression of the sorting operation after the CN processing thanks to the use of a predefined bubble position to get the default check to variable LLR value for the VN processing. This leads to both a reduced hardware complexity and a reduced memory bandwidth (almost 50% of reduction compared to a binary LDPC code). The two-step generation of the variable-to-check messages, i.e., the selection of the $n_{m_{in}} + n_\delta$ messages with smallest LLR, then the extraction of $n_{m_{in}}$ smallest LLR with distinct GF values, is also a new contribution.

As a proof of concept, the full design of a small code has been performed. Two codewords are decoded in parallel to avoid idle cycles in the hardware. Simulation results showed that the proposed decoder (partially layered scheduling and $n_{max,it} = 30$) outperforms the (860, 720) binary-LDPC SP used in the 5G standard by 0.3 dB at FER of 10^{-3}. Emulation results on FPGA show that the proposed decoder introduces only a 0.08 dB penalty loss in the waterfall region compared to the reference floating point BP layered decoder with $n_{max,it} = 8$. A drawback of the proposed architecture is the apparition of an error floor around a FER of 10^{-7}.

There are many possible extensions of this work. The first one is to find a way to mitigate the error floor. Then, to determine the optimal sets of parameters of the hybrid CN-VN architecture in the general case (different code length, code rate and Galois Field order). From this study, it would be possible to design a flexible hardware parallel architecture able to decode a set of codes with different coding rates and lengths. In terms of hardware, the advantages of the proposed CN-VN unit should be even greater when the code length is high enough to fully perform the layered decoding algorithm.

References

[1] M. C. Davey and D. J. C. MacKay, "Low density parity check codes over GF(q)," *IEEE Communications Letters*, vol. 2, no. 6, pp. 159–166, June 1998.

[2] S. Pfletschinger, A. Mourad, E. Lopez, D. Declercq, and G. Bacci, "Performance evaluation of non-binary LDPC codes on wireless channels," in *Proceedings of ICT Mobile Summit*. Santander, Spain, June 2009.

[3] D. Declercq, M. Colas, and G. Gelle, "Regular GF(2^q)-LDPC coded modulations for higher order QAM-AWGN channel," in *Proc. ISITA*. Parma, Italy, Oct. 2004.

[4] E. Boutillon, L. Conde-Canencia, and A. A. Ghouwayel, "Design of a GF(64)-LDPC decoder based on the EMS algorithm," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, no. 10, pp. 2644–2656, Oct 2013.

[5] M. Davey and D. MacKay, "Low-density parity check codes over GF(q)," *Communications Letters, IEEE*, vol. 2, no. 6, pp. 165–167, June 1998.

[6] D. J. C. MacKay and M. Davey, "Evaluation of Gallager codes for short block length and high rate applications," in *Proc. IMA Workshop Codes, Syst., Graphical Models*, 1999.

[7] Y. S. Park, Y. Tao, and Z. Zhang, "A fully parallel nonbinary LDPC decoder with fine-grained dynamic clock gating," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 2, pp. 464–475, Feb 2015.

[8] F. Cai and X. Zhang, "Relaxed min-max decoder architectures for nonbinary low-density parity-check codes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 11, pp. 2010–2023, Nov 2013.

[9] D. Declercq and M. Fossorier, "Decoding algorithms for nonbinary LDPC codes over GF(q)," *IEEE Trans. Comm.*, vol. 55, no. 4, pp. 633–643, April 2007.

[10] A. Voicila, D. Declercq, F. Verdier, M. Fossorier, and P. Urard, "Low complexity, low memory EMS algorithm for non-binary LDPC codes," in *IEEE Intern. Conf. on Commun., ICC'2007*. Glasgow, England, June 2007.

[11] D. Declercq and M. Fossorier, "Decoding algorithms for nonbinary LDPC codes over GF(q)," *IEEE Transactions on Communications*, vol. 55, no. 4, pp. 633–643, April 2007.

[12] A. Voicila, D. Declercq, F. Verdier, M. Fossorier, and P. Urard, "Low-complexity decoding for non-binary LDPC codes in high order fields," *IEEE Transactions on Communications*, vol. 58, no. 5, pp. 1365–1375, May 2010.

[13] E. Li, D. Declercq, and K. Gunnam, "Trellis-based extended Min-Sum algorithm for non-binary LDPC codes and its hardware structure," *IEEE Transactions on Communications*, vol. 61, no. 7, pp. 2600–2611, July 2013.

[14] J. O. Lacruz, F. García-Herrero, J. Valls, and D. Declercq, "One minimum only trellis decoder for non-binary low-density parity-check codes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 62, no. 1, pp. 177–184, Jan 2015.

[15] J. O. Lacruz, F. García-Herrero, M. J. Canet, and J. Valls, "Reduced-complexity nonbinary LDPC decoder for high-order Galois fields based on trellis Min-Max algorithm," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 8, pp. 2643–2653, Aug 2016.

[16] H. P. Thi and H. Lee, "Basic-set trellis Min-Max decoder architecture for nonbinary LDPC codes with high-order Galois fields," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 3, pp. 496–507, March 2018.

[17] C. Marchand, E. Boutillon, H. Harb, L. Conde-Canencia, and A. Al Ghouwayel, "Hybrid check node architectures for NB-LDPC decoders," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 2, pp. 869–880, Feb 2019.

[18] P. Schlafer, N. Wehn, M. Alles, T. Lehnigk-Emden, and E. Boutillon, "Syndrome based check node processing of high order NB-LDPC decoders," in *Telecommunications (ICT), 2015 22nd International Conference on*, April 2015, pp. 156–162.

[19] V. Rybalkin, P. Schläfer, and N. Wehn, "A new architecture for high speed, low latency NB-LDPC check node processing for GF(256)," in *2016 IEEE 83rd Vehicular Technology Conference (VTC Spring)*, May 2016, pp. 1–5.

[20] C. Marchand and E. Boutillon, "NB-LDPC check node with pre-sorted input," in *2016 9th International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*, Sept 2016, pp. 196–200.

[21] H. Harb, C. Marchand, A. Ghouwayel, A. Conde-Canencia, L., and E. Boutillon, "Pre-sorted forward-backward NB-LDPC check node architecture," in *SIPS*, 2016.

[22] H. Harb, A. C. Al Ghouwayel, and E. Boutillon, "Parallel generation of most reliable LLRs of a non-binary symbol," *IEEE Communications Letters*, vol. 23, no. 10, pp. 1761–1764, Oct 2019.

[23] C. Lin, S. Tu, C. Chen, H. Chang, and C. Lee, "An efficient decoder architecture for nonbinary LDPC codes with extended min-sum algorithm," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 63, no. 9, Sept 2016.

[24] M. J. S. Amin Farmahini-Farahani, Henry J. Duwe III and K. Compton, "Modular design of high-throughput, low-latency sorting units," *IEEE TRANSACTIONS ON COMPUTERS*, vol. 62, no. 7, pp. 1389–1402, July 2013.

[25] M. Rovini, F. Rossi, P. Ciao, N. L'Insalata, and L. Fanucci, "Layered decoding of non-layered LDPC codes," in *9th EUROMICRO Conference on Digital System Design (DSD'06)*, Aug 2006, pp. 537–544.

[26] H. Wymeersch, H. Steendam, and M. Moeneclaey, "Log-domain decoding of LDPC codes over GF(q)," in *Communications, 2004 IEEE International Conference on*, vol. 2, June 2004, pp. 772–776.

[27] V. Savin, "Min-max decoding for non binary LDPC codes," in *Information Theory, 2008. ISIT 2008. IEEE International Symposium on*, July 2008, pp. 960–964.

[28] Enabling Practical Wireless Tb/s Communications with Next Generation Channel Coding, H2020 Eureopean grant n 760150, (2017-2020) [website: https://epic-h2020.eu/].