

Design of a GF(64)-LDPC Decoder Based on the EMS Algorithm

Emmanuel Boutillon, Laura Conde-Canencia, Ali Al Ghouwayel

▶ **To cite this version:**

Emmanuel Boutillon, Laura Conde-Canencia, Ali Al Ghouwayel. Design of a GF(64)-LDPC Decoder Based on the EMS Algorithm. IEEE Transactions on Circuits and Systems Part 1 Fundamental Theory and Applications, 2013, 60 (10), pp.2644 - 2656. 10.1109/TCSI.2013.2279186 . hal-00777131

HAL Id: hal-00777131
https://hal.science/hal-00777131

Submitted on 31 Jan 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Design of a GF(64)-LDPC Decoder Based on the EMS Algorithm

Emmanuel Boutillon, *Senior Member, IEEE,* Laura Conde-Canencia, *Member, IEEE,* and Ali Al Ghouwayel

***Abstract*—This paper presents the architecture, performance and implementation results of a serial GF(64)-LDPC decoder based on a reduced-complexity version of the Extended Min-Sum algorithm. The main contributions of this work correspond to the variable node processing, the codeword decision and the elementary check node processing. Post-synthesis area results show that the decoder area is less than 20% of a Virtex 4 FPGA for a decoding throughput of 2.95 Mbps. The implemented decoder presents performance at less than 0.7 dB from the Belief Propagation algorithm for different code lengths and rates. Moreover, the proposed architecture can be easily adapted to decode very high Galois Field orders, such as GF(4096) or higher, by slightly modifying a marginal part of the design.**

***Index Terms*—Non-Binary low-density parity-check decoders, low-complexity architecture, FPGA synthesis, Extended Min Sum algorithm.**

I. Introduction

THE extension of binary Low-Density Parity-Check (LDPC) codes to high-order Galois Fields (GF(q), with $q > 2$), aims at further close the gap of performance with the Shannon limit when using small or moderate codeword lengths [1]. In [2], it has been shown that this family of codes, named Non-Binary (NB) LDPC, outperforms convolutional turbo-codes (CTC) and binary LDPC codes because it retains the benefits of steep waterfall region for short codewords (typical of CTC) and low error floor (typical of binary LDPC). Compared to binary LDPC, NB-LDPC generally present higher girths, which leads to better decoding performance. Moreover, since NB-LDPC are defined on high-order fields, it is possible to identify a closer connection between NB-LDPC and high-order modulation schemes. When associating binary LDPC to M-ary modulation, the demapper generates likelihoods that are correlated at the binary level, initializing the decoder with messages that are already correlated. The use of iterative demapping partially mitigates this effect but increases the whole decoder complexity. Conversely, in the NB case, the symbol likelihoods are uncorrelated, which automatically improves the performance of the decoding algorithms [3] [4]. Moreover, a better performance of the q-ary receiver processing has been observed in MIMO systems [5] [6]. Finally, NB-LDPC codes also outperform binary LDPC codes in the presence of burst errors [7] [8]. Further research on NB-LDPC considers their definition over finite groups G(q), which is a more general framework than finite Galois fields GF(q) [9]. This leads to hybrid [10] and split or cluster NB-LDPC codes [11], increasing the degree of freedom in terms of code construction while keeping the same decoding complexity.

From an implementation point of view, NB-LDPC codes highly increase complexity compared to binary LDPC, especially at the reception side. The direct application of the Belief Propagation (BP) algorithm to GF(q)-LDPC leads to a computational complexity dominated by $O(q^2)$ and considering values of $q > 16$ results in prohibitive complexity. Therefore, an important effort has been dedicated to design reduced-complexity decoding algorithms for NB-LDPC codes. In [12] and [13], the authors present an FFT-Based BP decoding that reduces complexity to the order of $O(d_c \times q \times \log q)$, where d_c is the check node degree. This algorithm is also described in the logarithm domain [14], leading to the so-called log-BP-FFT. In [15] [16], the authors introduce the Extended Min-Sum (EMS), which is based on a generalization of the Min-Sum algorithm used for binary LDPC codes ([17], [18] and [19]). Its principle is the truncation of the vector messages from q to n_m values ($n_m << q$), introducing a performance degradation compared to the BP algorithm. However, with an appropriate estimation of the truncated values, the EMS algorithm can approach, or even in some cases slightly outperform, the BP-FFT decoder. Moreover, the complexity/performance trade-off can be adjusted with the value of the n_m parameter, making the EMS decoder architecture easily adaptable to both implementation and performance constraints. A complexity comparison of the different iterative decoding algorithms applied to NB-LDPC is presented in [20]. Finally, the Min-Max algorithm and its selective-input version are presented in [21].

In the last years several hardware implementations of NB-LDPC decoding algorithms have been proposed. In [22] and [23], the authors consider the implementation of the FFT-BP on an FPGA device. In [24] the authors evaluate implementation costs for various values of q by the extension of the layered decoder to the NB case. An architecture for a parallel or serial implementation of the EMS decoder is proposed in [16]. Also, the implementation of the Min-Max decoder is considered in [25], [26] and optimized in [27] for GF(32). Finally, a recent paper [1] presents an implementation of a NB-LDPC decoder based on the Bubble-Check algorithm and a low-latency variable node processing [28].

Even if the theoretical complexity of the EMS is in the order of $O(n_m \times \log n_m)$, for a practical implementation, the parallel insertion needed to reorder the vector messages at the

E. Boutillon and L. Conde-Canencia are with the Lab-STICC laboratory, Lorient, CNRS, Université de Bretagne Sud

A. Al Ghouwayel is with the Lebanese International University.

Copyright (c) 2012 IEEE. Personal use of this material is permitted. However, permission to use this material for any other purposes must be obtained from the IEEE by sending an email to pubs-permissions@ieee.org.

[1]Paper published during the reviewing process of our manuscript.

TABLE I
NOTATION

	Code parameters
q	order of the Galois Field
m	number of bits in a GF(q) symbol, $m = \log_2 q$
H	parity-check matrix
M	number of rows in H
N	number of columns in H or number of symbols in a codeword
d_c	check node degree
d_v	variable node degree
$h_{j,k}$	an element of the H matrix
	Notation for the decoding algorithm
$\mathbf{X}$	a codeword
$\mathbf{x}_k$	a GF(q) symbol in a codeword
$x_{k,i}$	the i^{th} bit of the binary representation of $\mathbf{x}_k$
$\mathbf{Y}$	received codeword (channel information)
$\mathbf{y}_k$	a GF(q) symbol in a received codeword
$y_{k,i}$	the i^{th} noisy channel sample in $\mathbf{y}_k$
n_m	size of the truncated message in the EMS algorithm
$L^k(\mathbf{x})$	LLR value of the k^{th} symbol
$\tilde{\mathbf{x}}_k$	symbol of GF(q) that maximizes $P(\mathbf{y}_k\|\mathbf{x})$
$\hat{\mathbf{c}}_k$	a decoded symbol
$\hat{\mathbf{C}}$	the decoded codeword
$\{L^k(x)\}$	the intrinsic message, ($x \in$ GF(q))
$C2V_j^k$	check to variable message associated to edge $h_{j,k}$
$V2C_j^k$	variable to check message associated to edge $h_{j,k}$
λ^k	EMS message associated to symbol $\mathbf{x}_k$
$\lambda^k(l)^{GF}$	GF(q) value of the l^{th} element in the EMS message
$\lambda^k(l)^L$	LLR value of the l^{th} element in the EMS message
	Architecture parameters
n_b	number of quantization bits for an intrinsic message
n_y	number of quantization bits for the representation of $y_{k,i}$
n_{it}	number of decoding iterations
n_{op}	number of operations in an elementary check node processing
L_{dec}	latency of the decoding process (in number of clock cycles)
L_{VN}	latency of the variable node processing
L_{CN}	latency of the check node processing
n_{bub}	number of bubbles
S_{C2V}	subset of GF(q), $S_{C2V} = \{C2V^{GF}(l)\}_{l=1\ldots n_m}$
$\bar{S}_{C2V}$	subset of GF(q) that contains the symbols not in S_{C2V}

Elementary Check Node (ECN) increases the complexity to the order of $O(n_m^2)$. An algorithm to reduce the EMS ECN complexity is introduced in [29] for a complexity reduction in the order of $O(n_m\sqrt{n_m})$. The complexity of this architecture was further reduced without sacrifying performance with the L-Bubble-Check algorithm [30].

As the EMS decoder considers Log-Likelihood Ratios (LLR) for the reliability messages, a key component in the NB decoder is the circuit that generates the *a priori* LLRs from the binary channel values. An LLR generator circuit is proposed in [31], but this algorithm is software oriented rather than hardware oriented, since it builds the LLR list dynamically. In [32], an original circuit is proposed as well as the accompanying sorter which provides the NB LLR values to the processing nodes of the EMS decoder.

In this paper, we present a design and a reduced-complexity implementation of the L-Bubble Check EMS NB-LDPC decoder focusing our attention on the following points: the Variable Node (VN) update, the Check Node (CN) processing as a systolic array of ECNs and the codeword decision-making. Table I summarizes the notation used in the paper.

The paper is organized as follows: section II introduces ultra-sparse quasi-cyclic NB-LDPC codes, which are the one considered by the decoder architecture. This section also reviews NB-LDPC decoding with particular attention to the Min-Sum and the EMS algorithms. Section III is dedicated to the global decoder architecture and its scheduling. The VN architecture is detailed in section IV. The CN processor and the L-Bubble Check ECN architecture are presented in section V. Section VI is dedicated to performance and complexity issues and, finally, conclusions and perspectives are discussed in section VII.

II. NB-LDPC CODES AND EMS DECODING

This section provides a review of NB-LDPC codes and the associated decoding algorithms. In particular, the Min-Sum and the EMS algorithms are described in detail.

A. Definition of NB-LDPC codes

An NB-LDPC code is a linear block code defined on a very sparse parity-check matrix H whose nonzero elements belong to a finite field GF(q), where $q > 2$. The construction of these codes is expressed as a set of parity-check equations over GF(q), where a single parity equation involving d_c codeword symbols is: $\sum_{k=1}^{d_c} h_{j,k}\mathbf{x}_k = 0$, where $h_{j,k}$ are the nonzero values of the j-th row of H and the elements of GF(q) are $\{0, \alpha^0, \alpha^1, \ldots, \alpha^{q-2}\}$. The dimension of the matrix H is $M \times N$, where M is the number of parity-Check Nodes (CN) and N is the number of Variable Nodes (VN), i.e. the number of GF(q) symbols in a codeword. A codeword is denoted by $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)$, where $(\mathbf{x}_k)$, $k = 1 \ldots N$ is a GF(q) symbol represented by $m = \log_2(q)$ bits as follows: $\mathbf{x}_k = (x_{k,1}\ x_{k,2} \ldots x_{k,m})$.

The Tanner graph of an NB-LDPC code is usually much more sparse than the one of its homologous binary counterpart for the same rate and binary code length ([33], [34]). Also, best error correcting performance is obtained with the lowest possible VN degree, $d_v = 2$. These so-called *ultra-sparse* codes [33] reduce the effect of stopping and trapping sets, and thus, the message passing algorithms become closer to the optimal Maximum Likelihood decoding. For this reason, all the codes considered in this paper are ultra-sparse. To obtain both good error correcting performance and hardware friendly LDPC decoder, we consider the optimized non-binary protograph-based codes [35] [36] with $d_v = 2$ proposed by D. Declercq *et al.* [37]. These matrices are designed to maximize the girth of the associated bi-partite graph, and minimize the multiplicity of the cycles with minimum length [38]. This NB-LDPC matrix structure is similar to that of most binary LDPC standards (DVB-S2, DVB-T2, WiMax,...), and allows different decoder schedulings: parallel or serial node processors[2]. Finally, the nonzero values of H are limited to only d_c distinct values and each parity check uses exactly those d_c distinct GF(q) values. This limitation in the choice of the $h_{j,k}$ values reduces the storage requirements.

B. Min-Sum algorithm for NB-LDPC decoding

The EMS algorithm [15] is an extension of the Min-Sum ([39] [40]) algorithm from binary to NB LDPC codes. In this

[2]The final choice will be determined by the latency and surface constraints.

section we review the principles of the Min-Sum algorithm, starting with the definition of the NB LLR values and the exchanged messages in the Tanner graph.

1) Definition of NB LLR values: Considering a BPSK modulation and an Additive White Gaussian Noise (AWGN) channel, the received noisy codeword $\mathbf{Y}$ consists of $N \times m$ binary symbols independently affected by noise: $\mathbf{Y} = (y_{1,1}\ y_{1,2} \dots y_{1,m}\ y_{2,1} \dots\ y_{N,m})$, where $y_{k,i} = B(x_{k,i}) + w_{k,i}$, $k \in \{1, 2, \dots, N\}$, $i \in \{1, \dots, m\}$, $w_{k,i}$ is the realization of an AWGN of variance σ^2 and $B(x) = 2x - 1$ represents the BPSK modulation that associates symbol '-1' to bit 0 and symbol '+1' to bit 1.

The first step of the Min-Sum algorithm is the computation of the LLR value for each symbol of the codeword. With the hypothesis that the GF(q) symbols are equiprobable, the LLR value $L^k(\mathbf{x})$ of the k^{th} symbol is given by [21]:

$$L^k(\mathbf{x}) = \ln\Big(\frac{P(\mathbf{y}_k|\tilde{\mathbf{x}}_k)}{P(\mathbf{y}_k|\mathbf{x})}\Big) \tag{1}$$

where $\tilde{\mathbf{x}}_k$ is the symbol of GF(q) that maximizes $P(\mathbf{y}_k|\mathbf{x})$, i.e. $\tilde{\mathbf{x}}_k = \{\arg\max_{\mathbf{x}\in \mathrm{GF}(q)}, P(\mathbf{y}_k|\mathbf{x})\}$.

Note that $L^k(\tilde{\mathbf{x}}_k) = 0$ and, for all $\mathbf{x} \in$ GF(q), $L^k(\mathbf{x}) \geq 0$. Thus, when the LLR of a symbol increases, its reliability decreases. This LLR definition avoids the need to re-normalize the messages after each node update computation and permits to reduce the effect of quantization when considering finite precision representation of the LLR values.

As developed in [32], $L^k(\mathbf{x})$ can be expressed as:

$$L^k(\mathbf{x}) = \sum_{i=1}^{m}\Big(\frac{(y_{k,i} - B(x_i))^2}{2\sigma^2} + \frac{y_{k,i} - B(\tilde{x}_{k,i})^2}{2\sigma^2}\Big) \tag{2}$$

$$= \frac{1}{2\sigma^2}\sum_{i=1}^{m}\Big(2y_{k,i}(B(\tilde{x}_{k,i}) - B(x_i))\Big). \tag{3}$$

Using (3), $L^k(\mathbf{x})$ can be written as:

$$L^k(\mathbf{x}) = \sum_{i=1}^{m}|LLR(y_{k,i})|\Delta_{k,i}, \tag{4}$$

where $\Delta_{k,i} = x_i$ XOR $\tilde{x}_{k,i}$, i.e. $\Delta_{k,i} = 0$ if x_i and $\tilde{x}_{k,i}$ have the same sign, 1 otherwise and $LLR(y_{k,i}) = \frac{2}{\sigma^2}y_{k,i}$ is the LLR of the received bit $y_{k,i}$.

2) Definition of the edge messages: The Check to Variable (C2V) and the Variable to Check (V2C) messages associated to edge $h_{j,k}$ are denoted $C2V_j^k$ and $V2C_j^k$, respectively. Since the degree of the VN is equal to 2, we denote the two C2V (respectively V2C) messages associated to the variable node k ($k = 1 \dots N$) $C2V_{j_k(1)}^k$ and $C2V_{j_k(2)}^k$ (respectively $V2C_{j_k(1)}^k$ and $V2C_{j_k(2)}^k$) where $j_k(1)$ and $j_k(2)$ indicate the position of the two nonzero values of the k^{th} column of matrix H. Similarly, the d_c C2V (respectively V2C) messages associated to CN j ($j = 1 \dots M$) are denoted $C2V_j^{k_j(v)}$ (respectively $V2C_j^{k_j(v)}$), $v = 1 \dots d_c$, where $k_j(v)$ indicates the position of the v^{th} nonzero value in the j^{th} row of H.

3) The Min-Sum decoding process: The Min-Sum algorithm is performed on the Tanner bi-partite graph. At high level, this algorithm does not differ from the classical binary decoding algorithms that use the horizontal shuffle scheduling [41] or the layered decoder [42] principle.

The decoding process iterates n_{it} times and for each iteration M CN updates and $M \times d_c$ VN updates are performed. During the last iteration a decision is taken on each symbol, the decoded symbol is denoted by $\hat{\mathbf{c}}_k$ and the decided codeword by $\hat{\mathbf{C}}$. The codeword decision performed in the VN processors concludes the decoding process and the decoder then sequentially outputs $\hat{\mathbf{C}}$ to the next block of the communication chain.

The steps of the algorithm can be described as:

Initialisation: generate the intrinsic message $\{L^k(x)\}_{x\in\mathrm{GF}(q)}$, $k = 1 \dots N$ and set $V2C_{j_k(v)}^k = L^k$ for $k = 1 \dots N$ and $v = 1, 2$.

Decoding iterations: for 1 to the maximum number of iterations

for $(j = 1 \dots M)$ **do**

1) Retrieve in parallel from memory $V2C_j^{k_j(v)}, v = 1 \dots d_c$ messages associated to CN j.
2) Perform CN processing to generate d_c new $C2V_j^{k_j(v)}, v = 1 \dots d_c$ messages [3].
3) For each variable node $k_j(v)$ connected to CN j, update the second $V2C$ message using the new $C2V$ message and the L^k intrinsic message.

Final decision For each variable node, make a decision $\hat{\mathbf{c}}_k$ using the $C2V_{j_k(1)}^k$, $C2V_{j_k(2)}^k$ messages and the intrinsic message.

4) VN equations in the Min-Sum algorithm: Let $L(\mathbf{x})$, $V2C(\mathbf{x})$ and $C2V(\mathbf{x})$ be respectively the intrinsic, V2C and C2V LLR values associated to symbol $\mathbf{x}$. The decoding equations are:

Step 1: VN computation : for all $\mathbf{x} \in$ GF(q)

$$V2C(\mathbf{x}) = C2V(\mathbf{x}) + L(\mathbf{x}) \tag{5}$$

Step 2: Determination of the minimum V2C LLR value

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}\in\mathrm{GF}(q)}\{V2C(\mathbf{x})\} \tag{6}$$

Step 3: Normalization

$$V2C(\mathbf{x}) = V2C(\mathbf{x}) - V2C(\hat{\mathbf{x}}) \tag{7}$$

5) CN equations in the Min-Sum algorithm: With the forward-backward algorithm [43] a CN of degree d_c can be decomposed into $3(d_c-2)$ ECNs, where an ECN has two input messages U and V and one output message E (see Figure 7).

$$E(\mathbf{x}) = \min_{x_u, x_v \in \mathrm{GF}(q)^2}\{U(\mathbf{x}_u) + V(\mathbf{x}_v)\}_{x_u \oplus x_v = \mathbf{x}} \tag{8}$$

where $\oplus$ is the addition in GF(q).

[3]Note that the multiplicative coefficients associated to the edge of the Tanner graph are included in the CN processor.

6) Decision-making equations in the Min-Sum algorithm: The decision $\hat{\mathbf{c}}_k, k = 1 \dots N$ is expressed as:

$$\hat{\mathbf{c}}_k = \arg \min_{\mathbf{x} \in \mathrm{GF}(q)} \{C2V^k_{j_k(1)}(\mathbf{x}) + C2V^k_{j_k(2)}(\mathbf{x}) + L^k(\mathbf{x})\} \quad (9)$$

C. The EMS algorithm

The main characteristic of the EMS is to reduce the size of the edge messages from q to n_m ($n_m << q$) by considering the sorted list of the first smallest LLR values (i.e. the set of the n_m most probable symbols) and by giving a default LLR value to the others.

Let λ^k be the EMS message associated to the k^{th} symbol $\mathbf{x}_k$ knowing $\mathbf{y}_k$ (the so-called intrinsic message). λ^k is composed of n_m couples $(\lambda^k(l)^L, \lambda^k(l)^{GF})_{l=1\dots n_m}$, where $\lambda^k(l)^{GF}$ is a GF(q) element and $\lambda^k(l)^L$ is its associated LLR: $L^k(\lambda^k(l)^{GF}) = \lambda^k(l)^L$. The LLR verifies $\lambda^k(1)^L \leq \lambda^k(2)^L \leq \dots \leq \lambda^k(n_m)^L$. Moreover, $\lambda^k(1)^L = 0$. In the EMS, a default LLR value $\lambda^k(n_m)^L + O$ is associated to each symbol of GF(q) that does not belong to the set $\{\lambda^k(l)^{GF}\}_{l=1\dots n_m}$, where O is a positive offset whose value is determined to maximize the decoding performance [15].

The structure of the V2C and the C2V messages is identical to the structure of the intrinsic message λ^k. The output message of the VN should contain only, in sorted order, the first n_m smallest LLR values $V2C(l)^L, l = 1 \dots n_m$ and their associated GF symbols $V2C(l)^{GF}, l = 1 \dots n_m$. Similarly, the output message of the CN contains only the first n_m smallest LLR values $C2V(l)^L, l = 1 \dots n_m$ (sorted in increasing order), their associated GF symbols $C2V(l)^{GF}, l = 1 \dots d_c$ and the default LLR value $C2V(n_m)^L + O$.

Except for the approximation of the exchanged messages, the EMS algorithm does not differ from the Min-Sum algorithm, i.e., it corresponds to equations (5) to (9).

III. Architecture and decoding scheduling

This section presents the architecture of the decoder and its characteristics in terms of parallelism, throughput and latency.

A. Level of parallelism

We propose a serial architecture that implements a horizontal shuffled scheduling with a single CN processor and d_c VN processors. The choice of a serial architecture is motivated by the surface constraints as our final objective is to include the decoder in an existing wireless demonstrator platform [44]) (see section VI). The horizontal shuffled scheduling provides faster convergence because during one iteration a CN processor already benefits from the processing of a former CN processor. This simple serial design constitutes a first FPGA implementation to be considered as a reference for future parallel or partial-parallel enhanced architecture designs.

B. The overall decoder architecture

The overall view of the decoder architecture is presented in Figure 1. A single CN processor is connected to d_c VN processors and d_c RAM V2C memory banks. The CN processor receives in parallel d_c V2C messages and provides, after computation, d_c C2V messages. The C2V messages are then sent to the VN processors to compute the V2C messages of their second edge.

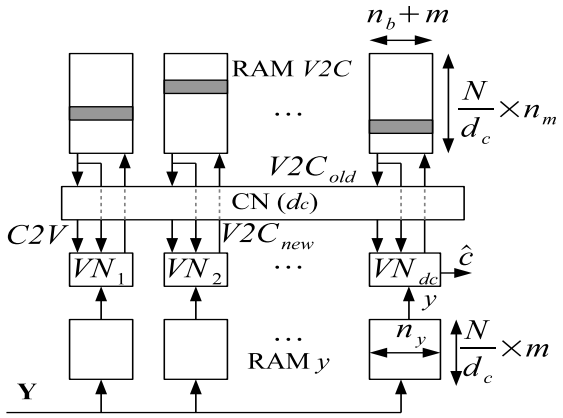


Fig. 1. Overall decoder architecture

Note that, for the sake of simplicity, we have omitted the description of the permutation nodes that implement the GF(q) multiplications. The effect of this multiplication is to replace the GF(q) value $V2C^{GF}(l)$ by $V2C^{GF}(l) \times h_{j,k}$ where the GF multiplication requires only a few XOR operations.

1) Structure of the RAMs: The channel information $\mathbf{Y}$ and the $V2C$ message associated to the N variables are stored in d_c memory banks RAMy and RAM V2C respectively [4]. Each memory bank contains information related to N/d_c variables. In the case of RAMy, the $(y_{k,i})_{i=1\dots m}$ received values associated to the variable $\mathbf{x}_k$ are stored in m consecutive memory addresses, each of size n_y bits, where n_y is the number of bits of the fixed-point representation of $y_{k,i}$ (i.e. the size of RAMy is $(N/d_c \times m)$ words of n_y bits). Similarly, each RAM V2C is also associated to N/d_c variables. The information $V2C_k$ related to $\mathbf{x}_k$ is stored in n_m consecutive memory addresses, each location containing a couple $(V2C^L(l), V2C^{GF}(l))$, i.e., two binary words of size (n_b, m), where n_b is the number of bits to encode the $V2C^L(l)$ values. To reduce memory requirements, for each symbol $\mathbf{x}_k$, only the channel samples $y_{k,i}$ and the extrinsic messages are stored in the RAM blocks. The intrinsic LLR are stored after their computation but they are overwritten by the V2C messages during the first decoding iteration. Each time an intrinsic LLR is required for the VN update, it is re-computed in the VN processor by the LLR generator circuit. Such approach avoids the memorisation of all the LLR of the input message (q messages) and thus, saves significant area when considering high-order Galois Fields ($q \geq 64$).

The partition of the N variables in the d_c memories is a coloring problem: the d_c variables associated to a given CN should be stored each in a different memory bank to avoid memory access conflicts (i.e. each memory bank must have a different color). A general solution to this problem has been

[4] In this paper, we represent two separate RAMs for the sake of clarity. However, in the implementation, RAMy and RAM V2C are merged into a single RAM.

studied in [45]. Since the NB-LDPC matrices considered in our study are highly structured (see [37]), the problem of partitioning is solved by the structure of the code.

2) Wormhole layer scheduling: The proposed architecture considers a wormhole scheduling. The decoding process starts reading the stored $\mathbf{Y}$ and V2C information sequentially and sends, in $m + n_m$ clock cycles, the whole $V2C$ message to the CN. After a maximum delay L_{CN}, the CN starts to send the $C2V$ messages to the VN processors, again with a value $C2V(l)$, $l = 1 \dots n_m$ at each clock cycle[5].

After a delay of L_{VN} (see section IV-B), the VNs send the new $V2C$ messages to the memory. The process is pipelined, i.e, every $\Delta = (m + L_{CN} + n_m)$ clock cycles, a new CN processing is started. The total time to process n_{it} decoding iterations is:

$$L_{dec} = n_{it} \times M \times \Delta + L_{VN} + n_m \tag{10}$$

where L_{dec} is given in clock cycles. Figure 2 illustrates the scheduling of the decoding process.

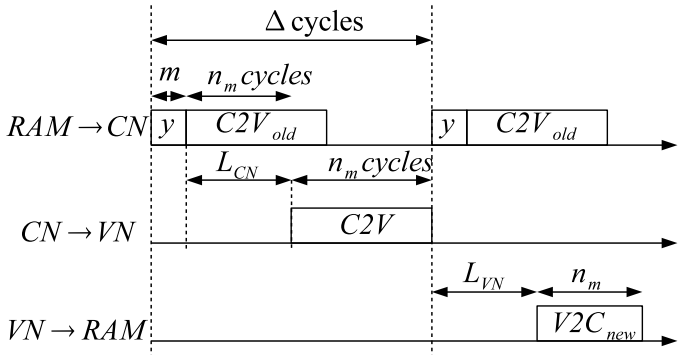


Fig. 2. Scheduling of the global architecture

3) The decoding steps: The decoding process iterates n_{it} times performing M CN updates and $M \times d_c$ VN updates at each iteration. During the last iteration a decision is taken on each symbol. The codeword decision is performed in the VN processors. This concludes the decoding process and the decoder then sequentially outputs $\hat{\mathbf{C}}$ to the next block of the communication chain. Note that the interface of the decoder is then rather simple:

1) Load $\mathbf{y}_k$ and store them in RAMy ($N \times m$ clock cycles).
2) Compute intrinsic information from $\mathbf{y}_k$ to initialize the $V2C$ messages.
3) Perform the n_{it} decoding iterations.
4) During the second edge processing of the last iteration, use the decision process to determine $\hat{\mathbf{c}}$.
5) Output the decoded message (N clock cycles) and wait for the new input codeword to decode.

IV. Variable node architecture

Although most papers on NB-LDPC decoder architectures focus on the CN, the implementation of the VN architecture is almost as complex, if not more, than the implementation of the CN in terms of control. In the proposed decoder, the VN processor works in three different steps: 1) the intrinsic generation; 2) the VN update and 3) the codeword decision. During the first step, prior to the decoding iterations, the Intrinsic Generation Module (**IGM**) circuit is active and generates the intrinsic message $(\lambda^k)_{k=1\dots N}$ from the received $\mathbf{y}_k$ samples. During the VN update, all the blocks of the VN processor, except the **Decision** block, are active. Finally, during the last decoding iteration, the **Decision** block is active (see Figure 3).

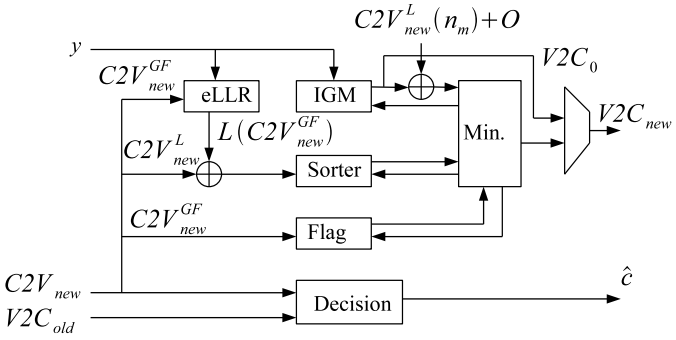


Fig. 3. Variable node architecture of the EMS NB-LDPC decoder

A. The Intrinsic Generator Module (IGM)

The role of the IGM is to compute the λ^k intrinsic messages. In [32], the authors propose an efficient systolic architecture to perform this task. The purpose is to iteratively construct the intrinsic LLR list considering, at the beginning, only the first coordinate, then the first two coordinates and so on, up to the complete computation of the intrinsic vector. The systolic architecture works as a FIFO that can be fed when needed. Once the input symbols $\mathbf{y}_{k,i}$ are received, and after a delay of $m + 2$ clock cycles ($m = log_2(q)$), the IGM generates a new output $\lambda^k(l)$ at every clock cycle. When pipelined, this module generates a new intrinsic vector every $n_m + 1$ clock cycles. Each intrinsic message is stored in the corresponding V2C memory location in order to be used during the first step of the iterative decoding process.

In the present design, in order to minimize the amount of memory, the intrinsic messages are not stored but re-generated when needed, *i.e.*, during each VN update of the iterative decoding process. This choice was dictated by the limited memory resources of the existing FPGA platform. In another context, it could be preferable to generate only once the intrinsic messages, store them in a specific memory and retrieve them when needed.

B. The VN update

In the VN processor, the blocks involved in the VN update are the following: the elementary LLR generator (**eLLR**), the **Sorter**, the **IGM**, the **Flag** memory and the **Min** block.

The task of the VN update is simple: it extracts in sorted order the n_m smallest values, and their associated GF(q) symbols, from the set $S = \{C2V^L(\mathbf{x}) + L(\mathbf{x})\}$ indexed by $\mathbf{x} \in \text{GF}(q)$ to generate the new V2C message.

[5]The time scheduling of the C2V message generation is not fully regular (see section V-C), but we consider a global latency L_{CN} so that the last element $C2V(n_m)$ arrives after $L_{CN} + n_m$ clock cycles

The set of GF(q) values can be divided into two disjoint subsets S_{C2V} and $\bar{S}_{C2V}$, with S_{C2V} the subset of GF(q) defined as $S_{C2V} = \{C2V^{GF}(l)\}_{l=1...n_m}$. In this set, $C2V^L(\mathbf{x}) = C2V^L(l)$, with l such that $C2V^{GF}(l) = \mathbf{x}$. The second set, $\bar{S}_{C2V}$ contains the symbols not in S_{C2V}. If $\mathbf{x} \in \bar{S}_{C2V}$, then $C2V^L(\mathbf{x})$ takes the default value $C2V^L(n_m)+O$ (see section II-C). The generation of S_{C2V} is done serially in 3 steps:

1) $C2V^{GF}(l)$ is sent to the **eLLR** module to compute $L(C2V^{GF}(l))$ according to (4). The value of $C2V^{GF}(l)$ is also used to put a flag from 0 to 1 in the **Flag** memory of size $q = 2^m$ to indicate that this GF(q) value now belongs to S_{C2V}. To be specific, the **Flag** memory is implemented as two memory blocks in parallel, working in ping-pong mode to allow the pipeline of two consecutive C2V messages without conflicts.
2) $L(C2V^{GF}(l))$ is added to $C2V^L(l)$ to generate $S_{C2V}(l)$. Note that S_{C2V} is no more sorted in increasing order.
3) The **Sorter** reorders serially the values in S_{C2V} in increasing order. The architecture of this **Sorter** is described in section IV-C.

The **IGM** is used to generate the second set $\bar{S}_{C2V}$. Each output value $\lambda(l)^L$ of the **IGM** is first added to $C2V^L(n_m)+O$. Then, if $\lambda(l)^{GF}$ belongs to S_{C2V} (i.e. the flag value at address $\lambda(l)^{GF}$ in the flag memory equals '1'), the value is discarded and a new value $\lambda(l+1)^L$ is provided by the **IGM** component to the **Min** component.

The **Min** component serially selects the input with the minimum LLR value from S_{C2V} and $\bar{S}_{C2V}$. Each time it retrieves a value from a set, it triggers the production of a new value of this set until all the n_m values of $V2C$ are generated.

C. The architecture of the Sorter block in the VN

The **Sorter** block in the VN processor is composed of $\lceil \log_2(n_m) \rceil$ stages, where $\lceil x \rceil$ is the smallest interger greater than or equal to x (see Figure 4). The i^{th} ($i = 1, \ldots, \lceil \log_2(n_m) \rceil$) stage serially receives two sorted lists of size 2^{i-1}, and provides a sorted list of size 2^i. The first received list goes into **FIFO_H** and the second list goes into **FIFO_L**. Then, the **Min_Select** block compares the first values of the two FIFOs, pulls the minimum one from the corresponding FIFO and outputs it. In practice, a stage starts to output the sorted list as soon as the first element of the second list is received. The latency of a stage is then $2^{i-1}+1$ clock cycles, plus one cycle for the pipeline, i.e. $2^{i-1}+2$ clock cycles. The size of **FIFO_H** is double (i.e. $2 \times 2^{i-1}$) in order to allow receiving a new input list while outputting the current sorted list.

As an example, to order a list of $n_m = 16$ values, the **Sorter** consists of 4 stages. The first stage receives 16 sequences of size $2^0 = 1$ and outputs 8 sorted lists of size $2^1 = 2$ (i.e. the elements are ordered by couples). The second stage outputs 4 lists of size $2^2 = 4$, the third stage outputs 2 lists of size 8 and, finally, the last stage outputs the whole sorted list of size $2^4 = 16$. The global latency of the **Sorter** is then expressed as:

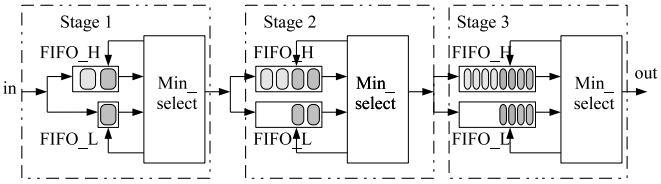


Fig. 4. Architecture of the Sorter block in the VN processor

$$L_{sorter}(n_m) = \sum_{i=1}^{\lceil \log_2(n_m) \rceil} (2^{i-1}+2) \quad (11)$$

Note that the sorter is able to process continuously blocks of size power of two, i.e., for $n_m = 12$, it is able to process a new block every 16 clock cycles and the latency is $L_{sorter}(n_m) = 23$.

D. Decision circuit architecture

The architecture of the simplified codeword decision circuit is presented in Figure 5. The optimal decoding is given by:

$$\hat{\mathbf{c}}_k = \arg \min_{\mathbf{x} \in \mathrm{GF}(q)} \{C2V^k_{j_k(1)}(\mathbf{x})^L + C2V^k_{j_k(2)}(\mathbf{x})^L + L(\mathbf{x})\} \quad (12)$$

Since the decision is done during the second branch update, we can replace in equation (12) $C2V^k_{j_k(1)}(\mathbf{x})^L + L(\mathbf{x})$ by $V2C^k_{j_k(2)}(\mathbf{x})^L$ (see equation (5)). Thus, we can write:

$$\hat{\mathbf{c}}_k = \arg \min_{\mathbf{x} \in \mathrm{GF}(q)} \{V2C^k_{j_k(2)}(\mathbf{x})^L + C2V^k_{j_k(2)}(\mathbf{x})^L\} \quad (13)$$

The processing of this equation is rather complex, since it requires either an exhaustive search for all values of **x**, or a complex Content Addressable Memory (CAM) to search for the common GF(q) values in the V2C and C2V messages. At this point, any method leading to a hardware simplification without significant performance degradation can be accepted. In a very pragmatic way, we tried several methods and we propose to replace, , in equation (13), $\mathbf{x} \in$ GF(q) by $\mathbf{x} \in \{V2C^k_{j_k(2)}(m)^{GF}\}_{m=1,2,3}$ in order to reduce the size of the CAM from n_m to 3.

Let S_0 be the set of the common values between the C2V and V2C messages, indexed by m:

$$S_0 = \{\{C2V^k_{j_k(2)}(l)\}^{GF}_{l=1...n_m}\} \cap \{\{V2C^k_{j_k(2)}(m)\}^{GF}_{l=1,2}\} \quad (14)$$

The decided symbol $\hat{c}_k$ is defined as:

$$\hat{c}_k = \arg\min\{V2C^k_{j_k(2)}(3)^L; C2V^k_{j_k(2)}(l)^L + V2C^k_{j_k(2)}(m)^L\} \quad (15)$$

where $\arg\min$ refers to the associated GF(q) value.

Figure 5 presents the architecture of the Decision circuit and Figure 6 shows performance simulation of the decision circuit comparing CAM sizes 3 and 12 for 8 and 20 decoding iterations. Note that reducing the CAM size from 12 to 3 does not introduce any performance loss when considering 20 decoding iterations.

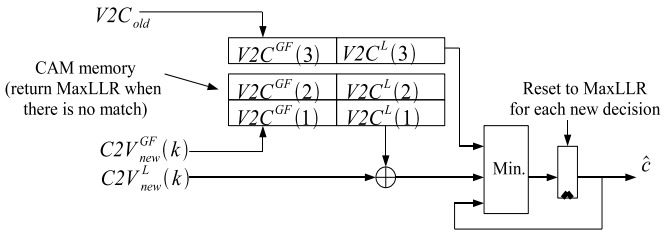


Fig. 5. Architecture of the codeword decision circuit

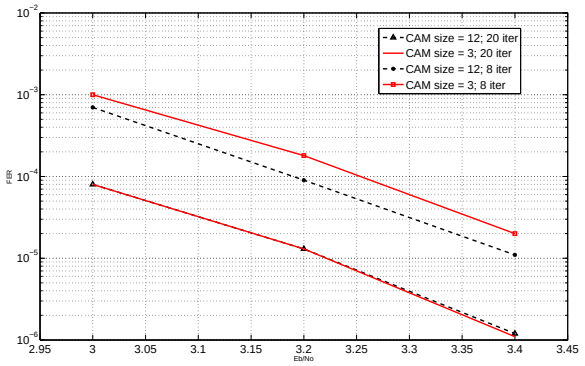


Fig. 6. Simulation of the decoder performance for different CAM sizes in the decision circuit

E. The latency of the VN

The critical path in the VN is the one containing the **Sorter** block, because this block waits for the arrival of the last C2V message to start its processing. The latency L_{VN} is then determined by the latency of the **Sorter**, i.e. L_{sorter}, plus a clock cycle for the adder and another one for the **Min** block.

$$L_{VN} = L_{sorter}(n_m) + 2 \tag{16}$$

V. The Check Node Processor

The CN processor receives d_c messages $V2C_j^{k_j(v)}$, performs its update based on the parity test described in equation (8), and generates d_c messages $C2V_j^{k_j(v)}$ to be sent to the corresponding d_c VNs. The processing of the received messages is executed according to the Forward-Backward algorithm [43] which splits the data processing into 3 layers of $d_c - 2$ ECNs, as shown in Figure 7. The main advantage of this architecture is that it can be easily modified to implement different values of d_c (i.e., to support different code rates).

Each ECN receives two vector messages U and V, each one composed of n_m (LLR,GF) couples, and outputs a vector message E whose elements are defined by equation (8) [15] [16]. This equation corresponds to extracting the n_m minimum values of a matrix $\mathbf{T}_\Sigma$, defined as $\mathbf{T}_\Sigma(i,j) = U(i) + V(j)$, for $(i,j) \in [1, n_m]^2$. In [16], the authors propose the use of a sorter of size n_m which gives a $O(n_m^2)$ computational complexity and constitutes the bottleneck of the EMS algorithm. In order to reduce this computational complexity, two simplified algorithms were proposed [29] [30]. In [29] the Bubble-Check algorithm simplifies the ECN processing by exploiting the properties of the matrix $\mathbf{T}_\Sigma$ and by considering a two-dimensional solution of the problem. This results in a reduction of the size of the sorter, theoretically in the order of $\sqrt{n_m}$. It is also shown in [29] that no performance loss is introduced when considering a size of the sorter smaller than the theoretical one.

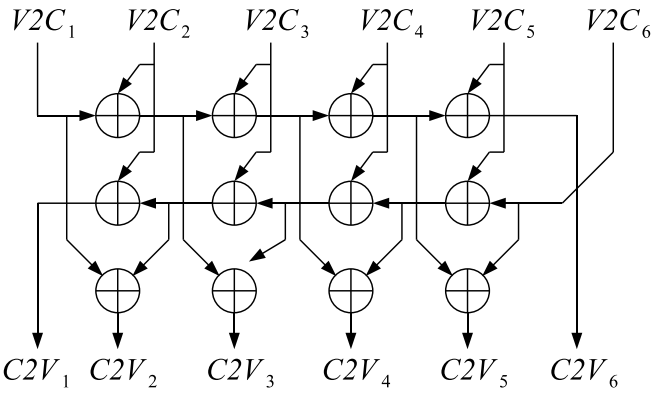


Fig. 7. Architecture scheme of a forward/backward CN processor with $d_c = 6$. The number of ECNs is $3 \times (d_c - 2)$

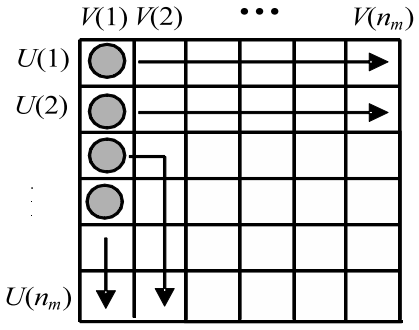


Fig. 8. L-Bubble Check exploration of matrix $\mathbf{T}_\Sigma$. The n_{bub} = 4 values in the sorter are initialized with the matrix values $\mathbf{T}_\Sigma(i,1)$, for $i = 1, \ldots, 4$, and only a maximum of $4 \times n_m - 4$ values in $\mathbf{T}_\Sigma$ are considered in the ECN processing. $\mathbf{T}_\Sigma(i,j) = U(i) + V(j)$

In [30], the authors suppose that the most reliable symbols are mainly distributed in the first two rows and two columns of matrix $\mathbf{T}_\Sigma$ and propose to use the so called L-Bubble Check which presents an interesting performance/complexity tradeoff for the EMS ECN processing. As depicted in Figure 8, the n_{bub} = 4 values in the sorter are initialized with the matrix values $\mathbf{T}_\Sigma(i,1)$, $i = 1, \ldots, 4$, and only a maximum of $4 \times n_m - 4$ values in $\mathbf{T}_\Sigma$ are considered in the ECN processing. Simulation results provided in [30] showed that the complexity reduction introduced by the L-Bubble Check algorithm does not introduce any significant performance loss. For this reason, we adopt the L-Bubble Check algorithm for the implementation of the present NB-LDPC decoder.

A. The L-Bubble ECN Architecture

The L-Bubble ECN architecture is depicted in Figure 9. The input values are stored in two RAMs *U* and *V* to be read during the ECN processing. At each clock cycle, each RAM

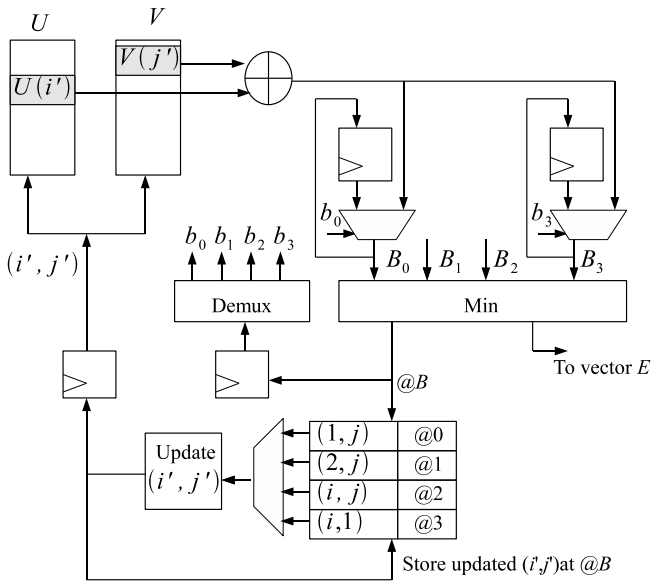


Fig. 9. Architecture scheme of the L-Bubble Check, $n_{bub} = 4$

receives a new (LLR, GF) couple and outputs a couple from a predetermined address. The LLR values of the couples read from the RAMs are added and the associated GF symbols are Xored (added modulo 2) to generate an element $\mathbf{T}_\Sigma(i', j')$ that feeds the sorter. This sorter is composed of four registers (B@ind) with **@ind** $\in \{0, 1, 2, 3\}$ (from left to right), four multiplexers and one **Min** operator that outputs the (LLR, GF) couple having the minimum LLR value.

The values fetched from the memories are denoted by $U(i')$ and $V(j')$, the values $U(i') + V(j')$ are named *bubbles* and feed the registers. The bubbles are tagged as follows: @0 : $(1, j)$, @1 : $(2, j)$, @2 : (i, j), @3 : $(i, 1)$. This addressing scheme is based on the position of the bubbles in the $\mathbf{T}_\Sigma$ matrix.

The complete ECN operation can be summarized as:

1) Read $U(i')$ and $V(j')$ from memories U and V.
2) Compute $\mathbf{T}_\Sigma(i', j') = U(i') + V(j')$. This bubble feeds the sorter to replace the bubble extracted in the preceding cycle. The corresponding register is thus bypassed.
3) Using the **Min** operator, determine the minimum bubble in the sorter and its associated index **@ind** $= \arg\min\{B_i, i = 0, \ldots, 3\}$.
4) From **@ind**, update the address of the i^{th} bubble and store it for the next cycle. The replacing rule is:
   a) if **@ind** = 0 or 1, then $(i', j') = (i, j + 1)$
   b) elsif (**@ind** = 3 & $j = 1$) then $(i', j') = (3, 2)$
   c) else $(i', j') = (i + 1, j')$

This architecture garanties the generation of the ordered list $U^L(i) + V^L(j)$. However, redundant associated GF symbols may appear, which are deleted at the output of the ECN [16]. In order to compensate this redundancy, n_{op} operations are performed in the ECN. Simulation results showed that the best performance/complexity trade-off is obtained for $n_{op} = n_m + 1$.

The critical path of the CN processor is then imposed by the ECN computation composed of RAM access, an adder, two serial comparators and an index update operation.

B. Multiplication and division in GF(q)

As described in section II, the messages crossing the edges between VNs and CNs are multiplied by predetermined GF(q) coefficients $h_{j,k} = \alpha^{a_{j,k}}$ when entering the CN and divided by the same coefficients (i.e. multiplied by $h_{j,k}^{-1} = \alpha^{q-1-a_{j,k}}$) when leaving the CN towards the VN. In order to perform these multiplications in GF(q), we have designed two wired multipliers dedicated to perform the multiplication over $GF(2^6)$. Each multiplier implemented on Virtex IV consumes 14 slices and operates at 900 MHz. The operands of the multiplier are the $V2C^{GF}$ (respectively, the $C2V^{GF}$) and the predefined coefficients stored in Read Only Memory (ROM) called ROM_{mul} (respectively ROM_{div}). Each ROM contains a $M \times 6m$ binary matrix, where each entry contains the six GF(q) coefficients.

C. Timing Specifications

This section describes the timing and scheduling details of the CN processor in the NB-LDPC EMS decoder. We first consider the scheduling at the ECN level and then at the CN processor, which is composed of three layers of serially concatenated ECNs.

1) ECN timing specifications: Figure 10 depicts the operations executed in the ECN at each Clock Cycle (CC). In this Figure, WM stands for Write Memory, RM for Read Memory, Ind upd for Index Update and NV for Non Valid output. The input data is represented by **D** and corresponds to two (LLR, GF) incoming couples. Finally, **E** represents the output (LLR, GF) couple.

The **Sorter** is represented by a vertical rectangle where a blank case represents an empty register and a dark one a filled one. At CC0, the vectors U and V receive their first inputs to be stored in the RAMs at CC1. At CC2, the stored messages are read, fed to the adder and then to the sorter. As shown in Figure 10, the first register is filled (dark case) with the adder output and this (LLR, GF) couple directly goes to the output (E1) as it corresponds to the minimum LLR value [6].

The latency of the ECN is 2 cycles. During the next three CCs, the ECN receives three new data couples and outputs three NV outputs. This 3-CC latency is denoted as Sorter Filling Latency (SFL). After the SFL, at CC4, the four registers in the sorter are filled and the second valid data couple is output.

The number of cycles needed to generate n_m valid outputs is then $n_m + 3$. However, due to the redundant GF(q) symbols that may appear when adding two input messages in U and V, some extra cycles are allowed in order to guarantee the generation of n_m different GF(q) symbols. To be specific, we consider $n_{op} = n_m + 1$, as detailed in section III-B2.

2) CN timing specifications: The Forward-Backward implementation of the CN processor consists of three layers of $d_c - 2$ serially concatenated ECNs (see Figure 7). Let $\text{ECN}e_{L_l}$ denote the e^{th} ECN of layer l, where the numeration is

[6]Let us recall that vectors U and V are sorted in increasing order.

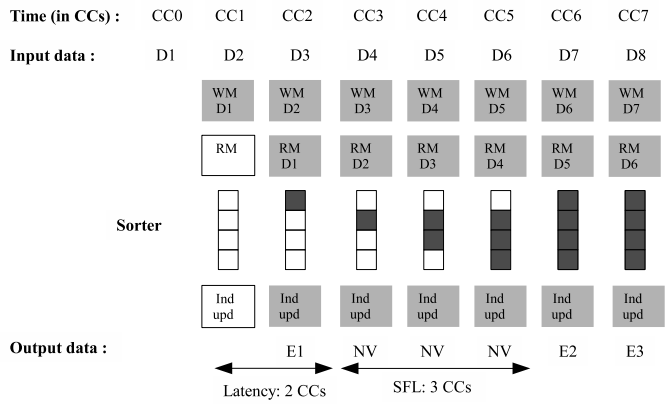


Fig. 10. ECN execution in the first CCs. **D** (resp. **E**) represents the input (resp. output) data corresponding to a (LLR, GF) couple; $n_{bub} = 4$

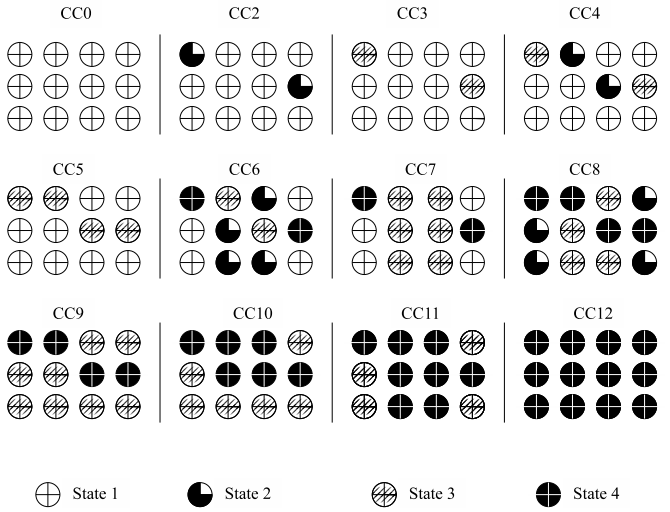


Fig. 11. Global CN execution

considered from left to right and top to bottom. The execution progress for each CC is depicted in Figure 11. The inputs $U_0(0)$ and $U_1(0)$ (resp. $U_4(0)$ and $U_5(0)$) feed ECN1$_{L_1}$ (resp. ECN4$_{L_1}$). Note that only these two ECNs have both inputs directly connected to the RAMs. All the other ECNs have at least one input generated by an adjacent ECN. Because of the latency contraints of the ECN, ECN1$_{L_1}$ and ECN4$_{L_2}$ provide their first output at CC2. These outputs activate ECN2$_{L_1}$ and ECN3$_{L_2}$, that deliver their first output at CC4.

Note that each ECN is in SFL after the generation of its first output. This means that at each of the following three CCs, an NV output is delivered. Four different states are then possible for an ECN:

State 1: Non active.
State 2: Generating first output. The sorter is not filled.
State 3: Generating a NV output. The sorter is not completely filled yet.
State 4: Generating a valid output and the sorter is filled. At this state, all the generated outputs are valid.

The global CN execution is represented in Figure 11. At each CC, the state of each ECN in the Forward/Backward architecture is indicated. For example, at CC0, no ECN is active (State 1). As the ECN latency for the first valid output is 2 CCs, ECN1$_{L_1}$ and ECN4$_{L_2}$ are in State 2 at CC2; ECN2$_{L_1}$ and ECN3$_{L_2}$ are in State 2 at CC4; at CC6, ECN3$_{L_1}$, ECN2$_{L_2}$, ECN2$_{L_3}$ and ECN3$_{L_3}$ are in State 2; finally, at CC8, ECN1$_{L_3}$ and ECN4$_{L_3}$ are in State 2, as well as ECN1$_{L_2}$ and ECN4$_{L_1}$. From CC12, all the outputs are valid, as all the ECNs are in State 4.

The decoding process of the whole CN is constrained by ECN1$_{L_3}$ and ECN4$_{L_3}$. For these ECN, the latency to output the first value is $2(d_c - 2)$. The SFL then follows (i.e. 3 CCs) and during the next $n_{op} - 1$ CCs, the rest of the message is output. The latency L_{CN} of the CN is then given by:

$$L_{CN} = 2 \times (d_c - 2) + 3 + n_{op} - n_m \tag{17}$$

VI. Performance and complexity

A. Decoding throughput

We consider a GF order of $q = 64$ for the implementation of the NB-LDPC decoder. The following code lengths and rates are chosen for the decoder synthesis:

- N = 192 symbols, R = 2/3, $d_c = 6$
- N = 48 symbols, R = 1/2, $d_c = 4$
- N = 72 symbols, R = 1/2, $d_c = 4$

The decoding throughput of the architecture (in bits per second) is

$$D = \frac{N \times R \times m}{L_{dec}} \times F_{clock}$$

where L_{dec} is the number of cycles to decode a frame (see equation (10)) and F_{clock} is the clock frequency. For example, for N = 192 symbols, R = 2/3 and $d_c = 6$ with $n_m = 12$, $n_{op} = 13$, $m = 6$ and $d_c = 6$, the latency values for the CN and VN processing are $L_{CN} = 12$ and $L_{VN} = 25$ clock cycles. The delay is $\Delta = 31$ clock cycles, which constitutes a maximum decoding latency of $L_{dec} = n_{it} \times M \times 31 + 37$ clock cycles to decode a frame and $D = 2.95$ Mbps. Note that D is the maximum decoding throughput assuming that there is a ping-pong input and output RAM to avoid idle times between the input loading of a new codeword and the output of a decoded one.

The serial architecture has been synthetized on a Xilinx Virtex4 XC4VLX200 FPGA. Table II presents the synthesis results [7] for three different frame lengths and code rates considering 8 decoding iterations and 6-bit quantization for input data (intrinsic LLR) as well as for the check-to-variable and variable-to-check messages. The proposed architeture can be easily adapted for any quasi-cyclic ultra-sparse (i.e., $d_v = 2$) GF(q)-LDPC code.

B. Emulation results

To obtain performance curves in record time we have implemented the complete digital communication chain on an FPGA device. For this, the hardware description of the

[7] these synthesis results do not include the ping-pong input and output RAM

TABLE II
POST-SYNTHESIS RESULTS OF THE SERIAL DECODER ARCHITECTURE FOR DIFFERENT CODE LENGTHS AND RATES ON THE XILINX VIRTEX 4 FPGA

	N = 48, R = 1/2	N = 72, R = 1/2	N = 192, R = 2/3
Slices	8727 (9%)	9277 (10%)	18758 (10%)
Slices Flip Flops	6330	6530	11712
Slices LUT	15906 (8%)	16894 (9%)	34846 (19%)
FIFO16/RAMB16s	4 (1%)	4 (1%)	6 (1%)
Maximum frequency (MHz)	64.15	62.53	61.33
Throughput (Mbps)	1.77	1.73	2.95

TABLE III
POST-SYNTHESIS AREA RESULTS FOR THE ENTIRE DIGITAL COMMUNICATION CHAIN IN THE HARDWARE EMULATOR PLATFORM

Resources	Slice Registers	Slice LUTS
Virtex5 FX70T	44800 (100%)	44800 (100%)
PowerPC 440 Virtex-5	2 (0%)	3 (0%)
PowerPC 440 DDR2 Memory Controller	2300 (5%)	1755 (4%)
LDPC-IP	8615 (19%)	14134 (32%)

different parts of the digital communication chain is required, namely the source, the encoder, the channel and the decoder. The source generates random bits that are encoded, BPSK modulated, affected by a an Additive White Gaussian Noise (AWGN), then demodulated and decoded. To emulate the effect of AWGN in the baseband channel, we consider the Hardware Discrete Channel Emulator as in [46]. We use the Xilinx ML507 FPGA DevKit which contains a Virtex5. The PowerPC processor is available as hardcore IP in the FPGA and can be used for software development. For practical purposes, we developped a Human Machine Interface (HMI) for the control of the emulation chain and the generation of performance curves. This HMI consists of a web server/FTP and its main advantage is being multiplatform, i.e. all the control can be done through a web server. More details about the emulator platform can be found in [47].

Table III summarises the post-synthesis area results. LDPC-IP stands for the digital communication chain including the NB-LDPC decoder. The PowerPC is mainly implemented as hardcore IP, which explains that its cells requirement is negligible. The digital chain is a multi-cadenced system, where the LDPC-IP block is cadenced at a frequency of 50 MHz [8].

We compared emulation and software throughputs for different scenarios (i.e. different code rates and frame lengths). The speedup factor between software simulation [9] and hardware emulation was greater than 100 for all cases. The performance results obtained with the hardware emulator platform were compared to the EMS and BP simulation results. The number of iterations for the BP was fixed to 100. Figure 12 considers a frame length of $N = 192$ symbols and a code rate $R = 2/3$.

[8]Note that the maximum frequency of the LDPC-IP block is of 65MHz. However, we select a frequency of 50 MHz because it is faster for design tools to find a place-and-route solution for a system with lower frequency constraints

[9]performed on an Intel Bi-Quad 8× 2 GHz processor with 24 Go RAM and 6144 Mo Cache

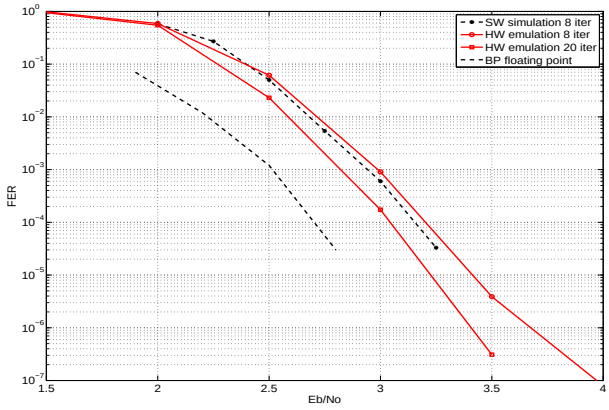


Fig. 12. Performance curves obtained with software simulation and hardware emulation for a GF(64)-LPDC code; N = 192 symbols, R = 2/3. The number of iterations for the BP is fixed to 100.

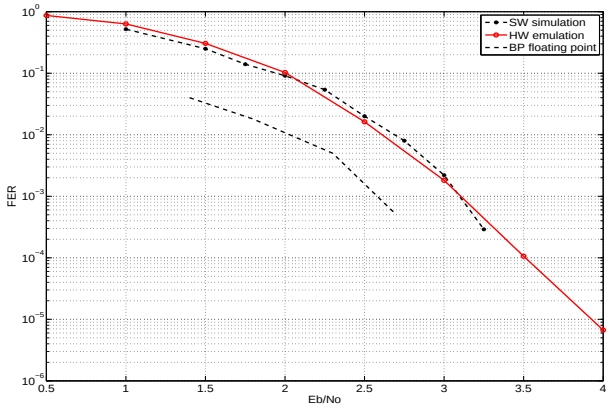


Fig. 13. Performance curves obtained with software simulation and hardware emulation for a GF(64)-LPDC code; N = 48 symbols, R = 1/2. The number of iterations for the BP is fixed to 100.

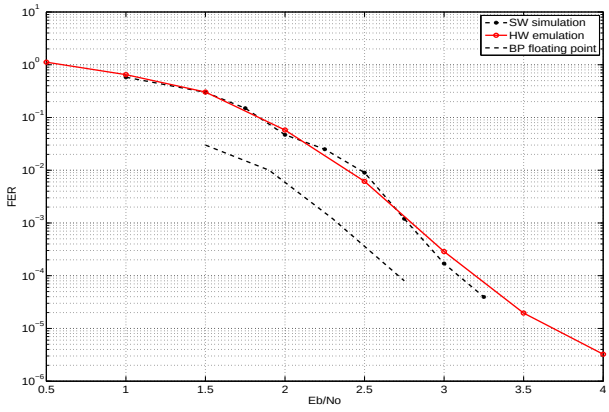


Fig. 14. Performance curves obtained with software simulation and hardware emulation for a GF(64)-LPDC code; N = 72 symbols, R = 1/2. The number of iterations for the BP is fixed to 100.

TABLE IV
SYNTHESIS COMPARISON OF STATE-OF-THE-ART NB-LDPC DECODERS. COMPARISON WITH [28] IS DISCUSSED IN THE TEXT.

Parameters	[23]	[26]	[27]	Our work
q	8	32	32	64
Target	FPGA Virtex2P	FPGA Virtex2P	–	FPGA Virtex4
Serial/parallel	Serial	31-parallel	31-parallel	Serial
Throughput (Mbps)	1	9.3	10	2.95
Algorithm	Mix Domain	Min-Max	Min-Max (optimized CNU)	EMS
Word length	8	5	5	6
Approx. Area (normalized)	1	10	4.6	4
Speed/area	1	1.08	2.17	0.74
Max. Frequency (MHz)	99.7	106.2	150	61.3
n_{it}	10-20	15	15	8

The curves show the good agreement between simulation and emulation results. Also, a gain of about 0.5 dB can be obtained when increasing the number of iterations from 8 to 20. The emulation results show that no error floor appears (up to a FER of 10^{-7}). Note that the performance of the implemented decoder is at less than 0.5 dB of the BP performance.

Figure 13 and Figure 14 consider $R = 1/2$ with $N = 48$ and $N = 72$ symbols, respectively. They both confirm the good agreement between emulation and simulation, and show that the performance of the implemented decoder is at less than 0.7 dB of the BP performance. The decoder generalization for different frame lengths and code rates is also validated.

C. Comparison with other NB-LDPC decoder implementations

Table IV summarizes the comparison of the synthesis results presented in [23] [26] [27] and our approach. Note that the GF order (q) and the decoding algorithm is not the same for each implementation, so the comparison is quite approximative but allows us to place our work in the state-of-the-art of NB-LDPC decoder implementations. In a general way, as we consider $q = 64$, complexity increase and significant performance gain are expected compared to [23], where $q = 8$, and [26] [27], where $q = 32$. The best speed-over-area ratio is presented by the 31-parallel ASIC implementation in [27], where the authors propose a trellis-Min-max algorithm for the CN processing. However, a performance loss of about 0.1 dB is to be expected, compared to $n_m = 16$-EMS decoding [10].

The serial implementation in [23] considers $q = 8$ and results in a 1-Mbps throughput and a synthesis on a Virtex2P device that consumes 4660 slices. This area is considered as a reference for the normalized area comparison in Table IV. Considering BP decoding, the GF(64) decoder would lead to an increase of complexity from $q^2_{\textbf{[23]}} = 8^2 = 64$ to $q^2_{\textbf{[our work]}} = 64^2 = 4096$ (i.e. a factor of 64). However, as we consider the EMS algorithm (with $n_m = 12$) the area is increased by only a factor 4 for the serial GF(64) decoder and the performance is at less than 0.5 dB of the BP performance for $N = 192$.

[10]Note that the authors in [27] consider $n_m = q/2$, and clasically $n_m << q$ in the EMS.

Note that the speed/area parameter is around 1 for [23][26] and 0.74 for our design. As [23] and [26] consider GF orders of 8 and 32, respectively, while our work considers $q = 64$, this comparison shows the interest of our work in terms of performance/area/throughput trade-off. Moreover, the reduced area required for serial architecture suggests that more complex semi-parallel architecture can be implemented, increasing the throughput of the decoding algorithm. Also, some effort should be dedicated to increase the maximum frequency of the design, knowing that the critical path is at the ECN.

While revising our paper, the work of [28] was published. There are many similarities between this work and ours: [28] uses the Bubble Check algorithm with the forward-backward implementation and both papers use a reduced-complexity VN processor. However, there are many significant differences: 1) in [28], the CN architecture is based on the Bubble-Check algorithm while our CN architecture is based on the more efficient and simplified algorithm called L-Bubble Check; 2) [28] proposes an interesting pre-fetching technique that permits to reduce the critical path of the Bubble Check; 3) the VN architecture in [28] is characterised by the use of the first L_{S-VN} values of the Intrinsic message ($L_{S-VN} \leq n_m$) for both computation of V2C messages and decision making. However, in our work, the VN architecture uses all the 64 intrinsic values for the computation of the V2C message and only the first 3 values for the decision making. In terms of complexity, similar results are obtained for a rate-1/2 NB-LDPC decoder [11]. The (960,480) NB-LDPC decoder implemented in [28] consumes 12444 slice registers, 15099 slice LUTs and operates at 100 MHz with a decoding throughput of 2.44 Mbit/s. A performance degradation of 0.5 dB is obtained compared to the BP algorithm at a FER of 10^{-4}, $n_m = 12$ and $n_{it} = 10$. In our implementation, the (72,36) NB-LDPC [12] consumes 6530 slice registers, 15906 slice LUTs and operates at 62 MHz with a decoding throughput of 1.73 Mbit/s. The same performance degradation of 0.5 dB is obtained with $n_m = 12$ and $n_{it} = 8$.

D. Toward decoding of NB-LDPC of high field order

Table V summarizes complexity of the main components as a fonction of m in the proposed architecture. Note that the **Flag** memory is the only component that has a size scaling with $q = 2^m$. As mentioned in section IV-B, this **Flag** memory allows to determine if a given intrisic message $\lambda(l)^{GF}$ belongs to the received $C2V^{GF}$ messages (refer to section IV-B). This task can also be done using an associated memory of n_m words of size m. If we do so, all the elements in the architecture scale with m, i.e., $log_2(q)$, except for the GF multiplier that scales in m^2 but represents a small part of the overall decoder. In other words, doubling the size of the field order would only have a small impact on the architectural cost. Thus, the use of CAM for the **Flag** memories opens the way to efficient decoding of high-order NB-LDPC codes, such as GF(256) or even higher.

[11]The implementation of a rate-2/3 decoder is not considered in [28]

[12]Note that the size of the codeword does not have any impact on the processing hardware but only on the memory size

TABLE V
COMPLEXITY AND PROCESSING TIME OF THE MAIN COMPONENTS AS A FONCTION OF m

Component	Complexity	Number of clock cycles
IGM (Variable Node)	m PE (see [32])	m
Δ		m
eLLR (in VN)	m	1
flag (in VN)	$q = 2^m$	1
U, V memories (in CN)	word of size $n_b + m$	1
GF multiplier	m^2	1
RAM y	$\frac{N}{d_c} \times m$ words	1
RAM V2C	word size of $n_b + m$	1

VII. CONCLUSION

This paper is dedicated to the architecture design of a GF(64) NB-LDPC decoder based on a simplified version of the EMS algorithm. Particular attention was given to NB LLR generation, VN update, codeword decision and reduced-complexity CN processing. For a frame length of 192 symbols, the FPGA-based decoder implementation consumes 19 Kslices on a Virtex 4 device and operates at 2.95 Mbps for 8 decoding iterations. The implementation is also generalized for other code rates and lengths and, in all cases, the hardware performance is at less than 0.7 dB of the BP decoding performance. The integration of the decoder in a hardware emulator platform provided emulation results showing that no error floor appears up to a FER of 10^{-7}. A general comparison of our synthesis results with the existing works shows the interesting performance/area/throughput trade-off of our design. Moreover, as highlighted in the previous section, replacing the **Flag** memory in the VN by a CAM of size n_m, makes that the architecture complexity scales in $n_m \times m$, (with $q = 2^m$). In other words, decoding very high-order field NB-LDPC codes, such as GF(256) or even GF(4096), is feasible with the proposed architecture.

From this work we can draw important conclusions about the implementation of EMS-like algorithms for NB-LDPC. First, the design of the VN is as complex as the design of the CN, even if most of the papers in the literature focus on the CN implementation which is considered as the bottleneck of the decoder. Note that the high complexity of the VN is due to the use of ordinate lists to represent the messages, which constitutes a high overhead cost. Second, many computations in the CN are useless: among the $d_c \times n_m$ inputs, less than $3 \times n_m$ are used in the output. Thanks to this point, it should be possible to decrease the number of computations in the CN to generate an output. To conclude, efficient decoding of NB-LDPC is still an open field. Other techniques should be invented to represent messages and/or to process parity-checks and variable updates.

ACKNOWLEDGMENT

This work is supported by INFSCO-ICT-216203 DAVINCI "Design And Versatile Implementation of Non-binary wireless Communications based on Innovative LDPC Code" (www.ict-davinci-codes.eu) funded by the European Commission under the Seventh Framework Program (FP7). The work has been done using also resources of the CPER PALMYRE II, with FEDER and the Brittany region fundings. The authors would also like to thank Dr. Yvan Eustache for synthesis and emulation results.

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. Sridhara and T. Fuja, "Low density parity check codes defined over groupes and rings," in *Proc. Inf. Theory Workshop*, Oct. 2002.

[4] 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.

[5] X. Jiand, Y. Yan, X. Xia, and M. Lee, "Application of non-binary LDPC codes based on euclidean geometries to MIMO systems," in *Int. Conference on wireless communications and signal processing, WCSP'09*. Nanjing, China, Nov. 2009, pp. 1–5.

[6] F. Guo and L. Hanzo, "Low-complexity non-binary LDPC and modulation schemes communicatins over MIMO channels," in *IEEE Vehicular Technology Conference (VTC'2004)*. Los Angeles, USA, Sept. 2004.

[7] J. Chen, L. Wang, and Y. Li, "Performance comparison between non-binary LDPC codes and Reed-Solomon codes over noise burst channels," in *IEEE Int. Conf. on Comm., Circuits and Systems (ICCCAS'2005)*. Hong Kong, China, May 2005.

[8] P. S. A. Marinoni and S. Valle, "Efficient design of non-binary LDPC codes for magnetic recording channels, robust to error bursts," in *IEEE Int. Symp. on Turbo Codes and Related Topics*. Laussane, Switzerland, Sept. 2008.

[9] W. Chen, C. Poulliat, D. Declercq, L. Conde-Canencia, A. Al-Ghouwayel, and E. Boutillon, "Non-binary LDPC codes defined over general linear group: Finite length design and practical implementation issues," in *IEEE 69th Vehicular Technology Conference: VTC2009-Spring*. Barcelona, Spain, April 2009.

[10] L. Sassatelli and D. Declercq, "Non-binary hybrid LDPC codes - structure, decoding and optimisation," in *IEEE Inf. Theory Workshop (ITW'2006)*. Chengdu, China, Oct. 2006.

[11] B. Shams, D. Declercq, and V. Y. Heinrich, "Non-binary split LDPC codes defined over finite groups," in *Proc. of IEEE ISWCS'2009*. Siena-Tuscany, Italy, Sept. 2009.

[12] 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.

[13] L. Barnault and D. Declercq, "Fast decoding algorithm for LDPC over GF(2^q)," in *Proc. Inf. Theory Workshop*. Paris, France, Mars 2003, pp. 70–73.

[14] H. Song and J. R. Cruz, "Reduced-complexity decoding of q-ary LDPC codes for magnetic recording," *IEEE Trans. Magn.*, vol. 39, pp. 1081–1087, Mars 2003.

[15] 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.

[16] 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.

[17] J. Zhao, F. Zarkeshvari, and A. H. Banihashemi, "On implementation of min-sum algorithm and its modifications for decoding LDPC codes," *IEEE Trans. Commun.*, vol. 53, no. 4, pp. 549–554, April 2005.

[18] M. Fossorier, M. Mihaljevi, and H. Imai, "Reduced complexity iterative decoding of LDPC codes based on belief propagation," *IEEE Trans. Commun.*, vol. 47, p. 673, May 1999.

[19] F. Kschischang, B. Frey, and H.-A. Loeliger, "Factor graphs and the sum product algorithm," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 498–519, Feb. 2001.

[20] L. Conde-Canencia, A. Al-Ghouwayel, and E. Boutillon, "Complexity comparison of non-binary LDPC decoders," in *Proceedings of ICT Mobile Summit*. Santander, Spain, June 2009.

[21] V. Savin, "Min-max decoding for non binary LDPC codes," in *Proc. IEEE Int. Symp. Information Theory, ISIT'2008*. Toronto, Canada, July 2008.

[22] C. Spagnol, W. Marnane, and E. Popovici, "FPGA implementations of LDPC over GF(2^m) decoders," in *IEEE Workshop on Signal Processing Systems*. Shanghai, China, Oct. 2007, pp. 273–278.

[23] C. Spagnol, E. Popovici, and W. Marnane, "Hardware implementation of GF(2^m) LDPC decoders," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 56, no. 12, pp. 2609–2620, Dec. 2009.

[24] T. Lehnigk-Emden and N. Wehn, "Complexity evaluation of non-binary Galois field LDPC code decoders," in *Int. Symp. on Turbo Codes*, vol. 56. Brest, France, Sept. 2010, pp. 63–67.

[25] J. Lin, J. Sha, and Z. Wang, "Efficient decoder desing for nonbinary quasicyclic LDPC codes," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, pp. 273–278, Jan. 2010.

[26] Z. Xinmiao and C. Fang, "Efficient partial-parallel decoder architecture for quasi-cyclic nonbinary LDPC codes," *IEEE Trans. CAS-I*, vol. 58, no. 2, pp. 402–414, February 2011.

[27] ——, "Reduced-complexity decoder architecture for non-binary LDPC codes," *IEEE Trans. VLSI*, vol. 19, no. 7, pp. 1229–1238, July 2011.

[28] Y. Tao, Y. Park, and Z. Zhang, "High-throughput architecture and implementation of regular (2, d_c) nonbinary LDPC decoders," in *IEEE Int. Symp. Circuits and Systems (ISCAS)*. Seoul, Korea, May 2012, pp. 2625–2628.

[29] E. Boutillon and L. Conde-Canencia, "Bubble-check: a simplified algorithm for elementary check node processing in extended min-sum non-binary LDPC decoders," *Electronics Letters*, vol. 46, pp. 633–634, April 2010.

[30] ——, "Simplified check node processing in nonbinary LDPC decoders," in *Int. Symposium on Turbo Codes and Iterative Information Processing*. Brest, France, Sept. 2010, pp. 201–205.

[31] A. Valembois and M. Fossorier, "An improved method to compute lists of binary vectors that optimize a given weight function with application to soft-decision decoding," *IEEE Trans. Commun.*, vol. 5, no. 11, pp. 456–458, Nov. 2001.

[32] A. Al Ghouwayel and E. Boutillon, "A systolic LLR generation architecture for non-binary LDPC decoders," *Communications Letters, IEEE*, vol. 15, no. 8, pp. 851 –853, Aug. 2011.

[33] C. Poulliat, M. Fossorier, and D. Declercq, "Design of regular (2,dc)-LDPC codes over GF(q) using their binary images," *IEEE Trans. Commun.*, vol. 56, no. 10, pp. 1626–1635, Oct. 2008.

[34] X.-Y. Hu and E. Eleftheriou, "Binary representation of cycle Tanner graph GF(2^b) codes," in *IEEE Int. Conf. Commun. ICC'2004*. Paris, France, June 2004.

[35] L. Zeng, L. Lan, Y. Tai, S. Song, S. Lin, and K. Abdel-Ghaffar, "Transactions papers - constructions of nonbinary quasi-cyclic ldpc codes: A finite field approach," *Communications, IEEE Transactions on*, vol. 56, no. 4, pp. 545 –554, april 2008.

[36] R. Peng and R. Chen, "Design of nonbinary quasi-cyclic LDPC cycle codes," in *Information Theory Workshop*. Tahoe City, USA, Sept. 2007, pp. 13–18.

[37] D. Declercq, C. Poulliat, and E. Boutillon, "Report on robust and hardware compliant design of non-binary protographs," *DAVINCI Deliverable 4.5, avalaible at http://www.ict-davinci-codes.eu*, 2009.

[38] A. Venkiah, D. Declercq, and C. Poulliat, "Design of cages with a randomized progressive edge growth algorithm," *IEEE Commun. Letters*, vol. 12(4), pp. 301–303, April 2008.

[39] J. Zhao, F. Zarkeshvari, and A. H. Banihashemi, "On implementation of Min-Sum algorithm and its modifications for decoding LDPC codes," *IEEE Trans. Commun.*, vol. 53, no. 4, pp. 549–554, April 2005.

[40] M. Fossorier, M. Mihaljevic, and H. Imai, "Reduced complexity iterative decoding of LDPC codes based on belief propagation," *IEEE Trans. Commun.*, vol. 47, no. 5, pp. 673–680, May 1999.

[41] J. Zhang and M. Fossorier, "Shuffled iterative decoding," *IEEE Transactions on Communications*, vol. 23, no. 2, pp. 209–213, June 2005.

[42] M. Mansour and N. Shanbhag, "High-throughput LDPC decoders," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 6, pp. 976 –996, Dec. 2003.

[43] H. Wymeersch, H. Steendam, and M. Moeneclaey, "Log-domain decoding of LDPC codes over GF(q)," in *IEEE Intern. Conf. on Commun., ICC'2004*. Paris, France, June 2004, pp. 772–776.

[44] M. Desmet and A. Dewilde, "Wireless demonstrator description and test," in *INFSCO-ICT-216203 DAVINCI D3.3.2*. avalaible at www.ict-davinci-codes.eu/project/deliverables/D332.pdf, June 2010, pp. 1–24.

[45] C. Chavet and P. Coussy, "A memory mapping approach for parallel interleaver design with multiple read and write accesses," in *Proc. of IEEE ISCAS'2010*. Paris, France, June 2010, pp. 3168–3171.

[46] E. Boutillon, Y. Tang, C. Marchand, and P. Bomel, "Hardware discrete channel emulator," in *Int. Conf. on High Performance Computing and Simulation (HPCS 2010)*. Caen, France, June 2010, pp. 452–458.

[47] E. Boutillon, Y. Eustache, P. Bomel, A. Haroune, and L. Conde-Canencia, "Performance measurement of DAVINCI code by emulation," in *INFSCO-ICT-216203 DAVINCI D6.2.3*. avalaible at www.ict-davinci-codes.eu/project/deliverables/D623.pdf, July 2011, pp. 1–47.