Diversity of Non-binary Cluster-LDPC codes using the EMS algorithm

B. Shams*, D. Declercq†, V. Heinrich‡

* Kohat University of Science & Technology, 26000, Pakistan {bilal.shams@gmail.com}
† ETIS/ENSEA/UCP/CNRS-UMR8051, France {david.declercq@ensea.fr}
‡ STMicroelectronics, 38926, Crolles, France {vincent.heinrich@st.com}

Abstract

We propose a new decoding procedure for Non-binary Cluster-LDPC codes based on the observation that different non-binary Tanner graphs for the same binary code may result in different convergence behaviours. We apply this concept locally at the check nodes and use multiple instances of a check node in parallel. Their outputs are joined together to get a single extrinsic message for each edge of the node. Each instance uses the famous Extended-Min-Sum (EMS) algorithm with very low complexity. We are able to obtain better decoding performance in the error-floor region with equal or lower complexity. We obtain the results via Mont-Carlo simulations and we make a comparison of the complexities as well.

Keywords: Low complexity decoding algorithm, Extended Min-Sum, Non-binary LDPC codes, Diversity, Tanner Graph

1. Introduction

Low Density Parity Check (LDPC) codes are block error correcting codes that were first proposed by (1) and were considered in standards like DVB-S2, Wi-Max, DSL, W-LAN etc. However, they start to show their weakness
when the code size is small or moderate (2), (3), (4) and when higher order modulation is used for transmission (5), (6), (7), (8). Non-binary (NB) LDPC codes were thus proposed (9) that showed improved performance for codes of shorter length and high order modulation schemes. This is because NB-LDPC codes have a very sparse Tanner graph as compared to their binary counterparts (10),(11). However, the advantages of NB-LDPC come with the consequence of a heavily increased decoding complexity. The optimal “Belief Propagation (BP) algorithm” (12) has a decoding complexity of the order $O(q^2)$ for a code defined over a Galois field of order-$q$ (13). Similarly the memory requirements are of the order $O(q)$ (6). Consequently the implementation of a LDPC decoder of order $q > 64$ becomes practically impossible (14). Amongst some low complexity algorithms, the Fast Fourier Transform (FFT) based BP (3) had a complexity of $O(q \log(q))$. The NB Min-Sum (15), which is a generalization of the binary Min-Sum, carried a complexity of $O(q^2)$ in addition (16). The Extended Min-Sum (EMS) algorithm was proposed in (14) and later modified in (17), where messages of size $n_m$ were used instead of $q$, where $n_m << q$. This brought down the complexity to order $O(n_m q)$ at the check nodes. Some other algorithms e.g. symbol based flipping algorithm (18) and stochastic decoders (19)-(20) were also proposed, however they worked only for low order codes. The EMS algorithm is, so-far, considered by numerous researchers as the best algorithm that has been proposed for codes defined over very high order Galois fields. It gives a very good decoding performance with a moderate complexity of implementation. It has been studied by researchers and several variants of the EMS algorithm has been proposed (21; 22; 23).
For better decoding performance, not only the decoding algorithm is important, but also the structure of the code plays an important role. The cycles and stopping sets that exist in a code play a huge role in defining its decoding performance (24). With this goal in mind, numerous mechanisms have been proposed for the construction of better LDPC codes. The idea is to construct a code with the highest possible girth (25; 26; 27; 28). Many powerful codes have been developed that include Generalized LDPC (GLDPC) codes (29), Doubly-Generalized LDPC (DGLDPC) codes (30), Tail-biting LDPC (TLDPC) codes (31) etc. One family called split-LDPC codes was proposed in (32) which was based on splitting the non-binary $GF(q)$ symbols into several sub-symbols using the binary image of the non-binary code. This definition was further generalized and hybrid-LDPC codes was proposed in (33) where the elements were defined over a finite group rather than Galois fields. This was a very general definition of LDPC codes that encompasses various other families. They were further studied in (34) and the name was changed to Cluster-LDPC codes because the code is formed of binary clusters. This family of codes showed very good decoding performance and is being considered for the European DA-VINCI project (35).

Thus, in order to have a good decoder, one needs to combine a good decoding algorithm with a code having a good structure. Therefore, one can predict very good performance when the EMS algorithm is used with Cluster-LDPC codes (36; 37). In this paper, we make use of the diversity offered by Cluster-LDPC codes and propose a decoder that further reduces the decod-
ing complexity while improving the performance in the error floor region. The paper is organized as follows: In section 2, we explain the structure of non-binary Cluster-LDPC codes and its EMS-decoding algorithm. In section 3, we explain the diversity offered by Cluster-LDPC codes and based on it, propose a new decoding procedure. We present results on Monte-Carlo simulations and also make an analysis of the complexity. In section 4, we conclude the paper.

2. Non-binary cluster-LDPC codes

2.1. Definition and Structure

An LDPC code is a linear block code defined over a very sparse parity check matrix $H$ with dimensions $(M \times N)$ and the elements belonging to the Galois field $GF(q)$. Let $x = [x_0, ..., x_{N-1}]$ be a codeword defined also over a Galois field $GF(q)$. The $i^{th}$-parity equation can also be written as:

$$\sum_{j: h_{ij} \neq 0} h_{ij} x_j = 0 \quad \text{in} \quad GF(q)$$

where $h_{ij}$ are the non-zero elements from $GF(q)$. This definition can be generalized to the case of non-binary code ensembles defined over a finite group $\mathbb{G}(2^p)$ (38). The $i^{th}$-parity equation can thus be written as:

$$\sum_{j: h_{ij} \neq 0} h_{ij}(x_j) = 0 \quad \text{in} \quad \mathbb{G}(2^p)$$

where $h_{ij} = \mathbb{G}(2^p) \leftrightarrow \mathbb{G}(2^p)$ is a linear mapping function. This is a very general definition and encompasses the classical non-binary $GF(q)$-LDPC codes as a special case. The functions $h_{ij}$ are linear and can be represented in the form of a binary matrix of size $(p \times p)$ denoted by $H_{ij}$. The binary matrix
$H_{ij}$ representing the function $h_{ij}(\cdot)$ is called a "cluster" and thus this family is called Cluster-LDPC codes.

$$H_{ij}$$

This definition is further generalized and the clusters are represented as rectangular binary matrices of size $(p_2 \times p_1)$ (39). In principle, the values of the column dimension of the clusters can be different for a single parity check equation defined in eq. (2), as long as $p_{1j} < p_2$ (40). The code is no longer defined in a single order, which makes it difficult to describe them algebraically. In order to simplify the presentation of the code, in this paper, we fix the size of the clusters to $(p_2 \times p_1)$ and describe the structure of Cluster-LDPC codes with the binary image of the parity check matrix $H_{bin}$, which is built from the binary $(p_2 \times p_1)$ clusters $H_{ij}$ as shown in Fig. 1. The null elements of the parity matrix are represented with an all-zero matrix of size $(p_2 \times p_1)$.

The basic idea of cluster-LDPC codes is to decode a binary parity check matrix with a non-binary Tanner graph. The binary image $H_{bin}$ corresponds to a binary code which is locally dense but is cluster-wise sparse. This means
that the non-binary Tanner graph will be sparse if the edges connecting the variable and check nodes represent the connections between the clusters i.e. functions $h_{ij}$. The variable nodes are then defined in order-$2^{p_1}$, whereas the check nodes are defined in order-$2^{p_2}$. Fig. 2 shows the messages flowing across the Tanner graph of cluster-LDPC codes. The messages in the graph are explained in section 2.2.

Now, the binary image representation form a local component code with a parity matrix $H_{cc}$ composed of $p_2$ rows and $p_1 d_c$ columns, formed from $d_c$ clusters. The component code formed from the $ith$-parity equation is:

$$H_{cc} = [ H_{i1} H_{i2} ... H_{id_c} ]$$

The symbols $x_j$ of the group $G(2^{p_1})$ carry a binary map of $p_1$ bits. Similarly, each symbol of the group $G(2^{p_2})$ carry a $p_2$ bits binary map. Using them with above specified notations for the binary clusters $H_{ij}$, the parity equation is

Figure 2: The messages flowing across a Tanner graph for cluster-LDPC codes
written in matrix form as:

\[ \sum_{j : H_{ij} \neq 0} H_{ij} X_j^T = 0^T \]  \hspace{1cm} (4)

where 0 represents the all-zero vector.

When decoding the non-binary Tanner graph of the code, the \((p_2 \times p_1)\) non-zero clusters \(H_{ij}\) correspond to one-to-one functions which transforms a binary map of \(p_1\) bits to \(p_2\) bits.

\[ \{b_{\alpha_n}[k]\}_{k=1...p_2} = H_{ij} : \{b_{\alpha_m}[k]\}_{k=1...p_1} \]  \hspace{1cm} (5)

where \(\{b_{\alpha_n}[k]\}_{k=1...p}\) and \(\{b_{\alpha_m}[k]\}_{k=1...p}\) are the binary representations of the \(\alpha_n\) and \(\alpha_m\) respectively. Its equivalent, the non-binary case, corresponds to a linear mapping of symbol \(\alpha_n \in \mathbb{G}(2^{p_1})\) to \(\alpha_m \in \mathbb{G}(2^{p_2})\). An example of the process is shown in Fig. 3 for a cluster of size \((4 \times 2)\).

2.2. The EMS Decoding algorithm for cluster-LDPC Codes

The EMS algorithm is a generalization of the Min-Sum algorithm that uses a log-likelihood ratios (LLR) instead of a probabilities for the symbols \(\alpha_i \in \mathbb{G}(q)\):

\[ L = [L[0] \ L[\alpha_1] \ L[\alpha_2] \ldots \ L[\alpha_{q-1}]] \]  \hspace{1cm} (6)

The LLR-value \(L[\alpha_i]\) is represented as:

\[ L[\alpha_i] = \log \left( \frac{P(\alpha_i)}{P(\alpha_{\text{max}})} \right) \]  \hspace{1cm} (7)

where \(P(\alpha_i)\) is the probability of the symbol \(P(\alpha_i)\) and \(P(\alpha_{\text{max}})\) is the most probable symbol of the message vector. This representation of LLRs is better
Figure 3: Cluster of size \((4 \times 2)\) and the mapping of a order-\(p_1 = 2\) message to order-\(p_2 = 4\) suited for fixed point implementations as it results in a better quantization scheme. Saturation occurs at the less reliable symbols and not at the most reliable ones. Hence, it reduces the quantization noise which in turn improves the decoding performance (41).

The EMS algorithm reduces the complexity of the MinSum algorithm by using a truncated message vector. The idea is to sort the LLR-vector in descending order of LLR-values and consider only the highest \(n_m\) values where \(n_m << q\). The remaining values are considered null. A message vector in the EMS algorithm would thus be represented as:

\[
L = [L[0]L[1]...L[\alpha_{n_m-1}] 0 0 ... 0]
\] (8)
The decoder is initialized with information from the channel. The following four steps are then iteratively repeated:

2.2.1. Variable Nodes Update

All the messages involved at a variable nodes are of the same order and represent all the possible $2^p$ symbols without any message truncation. For a symbol $\alpha^k \in \mathbb{G}(2^p)$, the output message $U_{vf_i}$ on the edge connected to the function node $f_i$ is computed as:

$$U_{vf_i}[\alpha^k_{p_1}] = L_{ch}[\alpha^k_{p_1}] + \sum_{j=0, j \neq i}^{d_u-1} V_{f_{ji}v}[\alpha^k_{p_1}]$$

(9)

where $L_{ch}$ is the information from the channel, $V_{f_{ji}v}$ is the input on the edge connected to function node $f_j$. This process requires only a simple adder and is very simple as compared to the classical EMS for $GF(q)$ codes.

2.2.2. Function Nodes Update

The messages are updated using the functions $h_{ij}(.)$ which represents the mapping of messages from $\mathbb{G}(2^p) \leftrightarrow \mathbb{G}(2^q)$. Using the equation 5, the mapping is computed as $\alpha_m = h_{ij}(\alpha_n)$. The update procedure is thus represented as:

$$U_{pc}[\alpha^m_{p_2}] = U_{vp}[\alpha^n_{p_1}]$$

(10)

where $\alpha^m_{p_2} \in \mathbb{G}(2^q)$ and $\alpha^n_{p_1} \in \mathbb{G}(2^p)$ are symbols of two different orders.

2.2.3. Check Nodes Update

The EMS algorithm is executed using the forward-backward (F/B) strategy (42). The F/B strategy divides the whole process into several elementary procedures, with each elementary module involving only two inputs and a
single output. The outputs of the modules are joined together in a forward-backward manner in order to compute the extrinsic output at each edge of the check node. The F/B strategy for a degree $d_c = 5$ check node is shown in Fig. 4. The input messages are represented as $\{U_0, ..., U_4\}$ and the output messages $\{V_0, ..., V_4\}$.

![Figure 4: The Forward-Backward procedure applied to a degree $d_c = 5$ check node](image)

The input messages are sorted in ascending order of LLR-values and the log-domain check update process is then carried out. The input messages are defined in order-$2^{p_2}$, but have a size of $2^{p_1}$ elements. This is for the reason that the remaining $(2^{p_2} - 2^{p_1})$ symbols are restricted states. Since, the check update process is based on the EMS procedure, the intermediate messages are, however, truncated to size $n_m << 2^{p_2}$, as they may or may not include restricted states. Since the messages are truncated and sorted in ascending
order of LLR-values, the symbol information corresponding to each LLR-value is carried in a separate vector $\beta_U$ with every LLR-vector $U$. A message of $n_m$ values is thus composed of two vectors and denoted as:

$$U_{ij} = [L_{U_{ij}}[\alpha_1] L_{U_{ij}}[\alpha_2] L_{U_{ij}}[\alpha_3]...L_{U_{ij}}[\alpha_{n_m-1}]] \tag{11}$$

$$\beta_{U_i} = [\alpha_1 \alpha_2 \alpha_3...\alpha_{n_m-1}] \tag{12}$$

The elementary process computes the lowest $n_m$ combinations of the input LLR-values. For an elementary process, consider $U_{f_{1c}}$ and $U_{f_{2c}}$ as inputs and $I_1$ as the output intermediate message. The vectors $\beta_{U_{f_{1c}}}$, $\beta_{U_{f_{2c}}}$ and $\beta_{I_1}$ carry their corresponding symbol information. Let $S(\beta_{I_1})[k]$ be the set of all possible symbol combinations satisfying the parity equation:

$$\beta_{U_{f_{1c}}}[i] \oplus \beta_{U_{f_{2c}}}[j] \oplus \beta_{I_1}[k] = 0 \tag{13}$$

The output messages is thus computed as:

$$I_1[k] = \min_{S(\beta_{I_1})[k]} (\beta_{U_{f_{1c}}}[i] + \beta_{U_{f_{2c}}}[j]) \tag{14}$$

For further details on the process, the reader is referred to (17).

2.2.4. Inverse Function Nodes Update

At the check nodes output, a message $V_{cp}$ is an order-$2^{p_2}$ message of length $n_m$, with the vector $\beta_{V_{cp}}$ carrying the corresponding symbol information. The function nodes then maps the message $V_{cp}$ into a message $V_{pv}$ defined in order $2^{p_1}$ by using the function $f_{ij}(.)$ in the reverse direction.

$$V_{pe}[\alpha_{p_1}^j] = \begin{cases} V_{cp}[\alpha_{p_1}^i] & \text{if } \alpha_{p_2}^i \in S(\beta_f) \\ 0 & \text{otherwise} \end{cases} \tag{15}$$
These steps are repeated iteratively. At the end of each iteration, the APP is calculated and tests are performed to verify whether a valid codeword had been received or not. In the case of an invalid codeword, the iterations are repeated until either a valid codeword is received or a fixed number of iterations have expired. If the iterations expires without decoding a valid codeword, a decoding failure is declared for the received word.

3. Diversity of NB-cluster LDPC codes

LDPC codes defined over groups offer diversity in terms that different non-binary Tanner graphs of the same code formed form the binary PCM can derive different decoding performances (43). We make use of this idea and apply it locally at the check nodes level. We propose to use several discrete processes of the same check node in parallel with very low values of $n_m$. The output of all the various processes are then fusioned together to form the extrinsic outputs that exhibits good decoding performance.

The parity equation for the $i^{th}$ check node, as described in eq. 2 can be seen as a local codeword composed of $d_c$ words. Applying a random permutation on the sequence order of the words will keep the same code space but will result in a different local decoding graph. As a consequence, if the same inputs are provided to the two graphs, the output dynamics of the two codes will be different. This result in a different convergence behaviour for each decoder.
To elaborate this further, consider eq. 3, where $H_{cc_i}$ represent a component code at the check nodes level. It is composed of $d_c$ binary clusters $H_{ij}$, with $j = 1...d_c$, each of size $(p_2 \times p_1)$. The $(p_2 \times p_1d_c)$ binary image of the component code represents a binary LDPC code. If we give a random permutation to the order of the components $H_{ij}$, it will represent a different binary LDPC codes and hence a different binary Tanner graph.

$$H_{cc_i}^{(1)} = [H_i H_{i2}... H_{id_c}]$$ (16)

$$H_{cc_i}^{(2)} = [H_{i2} H_{id_c}... H_{i3}]$$ (17)

An example of the component codes along with their corresponding binary Tanner graphs is given below in Fig. 5. The two binary Tanner graphs are different from one another, however, at the non-binary level, the code space remains the same. For a truncated message of size $n_m$, the two codes $H_{cc_i}^{(1)}$ and $H_{cc_i}^{(2)}$ will result in different decoding behaviour and hence different performance.

$$H_{cc_i}^{(1)} = \begin{bmatrix}
10 & 01 & 00 & 01 & 01 \\
01 & 00 & 10 & 10 & 10 \\
10 & 10 & 11 & 01 & 00 \\
00 & 10 & 01 & 10 & 01
\end{bmatrix}$$

$$H_{cc_i}^{(2)} = \begin{bmatrix}
00 & 01 & 01 & 10 & 01 \\
10 & 10 & 10 & 01 & 00 \\
11 & 00 & 10 & 10 & 10 \\
01 & 01 & 10 & 00 & 10
\end{bmatrix}$$

Figure 5: Random permutations of component code $H_{cc_i}$, and their corresponding binary Tanner graphs

We, now, define a diversity as a random permutation applied to the order of
the elements of the local parity check matrix of a check node.

\[ D^{(k)}_{p_k} = \pi^{(k)}(H_{ij}) \]  

(18)

A diversity set is then defined as a set of such type of permutations.

\[ \{H_{ij}, D^{(1)}_{p_1}, D^{(2)}_{p_2}, \ldots, D^{(N_d-1)}_{p_{N_d-1}}\} \]  

(19)

where \( N_d \) is the number of diversities. The check node can now be processed with the different elements of the diversity sets. For the same input LLR-vectors, each instance of the check node using the diversity set will result in different extrinsic output LLR-values. The outputs of the different instances are then fused together to obtain the output LLRs which are then fed to the variable nodes. This process is shown in Fig. 6.

To fuse the outputs of the various instances, there are various merging techniques that one can think of e.g. \( \max(\cdot) \), \( \text{average}(\cdot) \), \( \text{median}(\cdot) \) etc. However, since the check update process is founded on the EMS algorithm, the goal of which is to compute the minimum LLR-value of the combination of inputs that verify the parity. Thus, \( \min(\cdot) \) is the obvious choice of operator to merge the LLR-vectors. With the other operators mentioned above, the performance was not satisfactory. For a certain symbol, we choose the lowest LLR-value of that symbol amongst all the LLR-vectors. For example, using \( N_d = 3 \), for the symbol \( \alpha^i \), the output is chosen as:

\[ V_0[\alpha^i] = \min(L_0^0[\alpha^i], L_1^0[\alpha^i], L_2^0[\alpha^i]) \]  

(20)

where \( L_k^{V_j} \) is the \( j^{th} \) LLR-vector corresponding to the output LLR-vector \( V_j \) and associated with the check node instance \( k \).
Figure 6: Multiple instances of a check update process scrambled in parallel using diversity of group-LDPC codes

The output LLRs, while using the diversity set, exhibit better convergence behaviour than while using a single instance of the check node. It makes use of the diversity and thus better LLRs are constructed. We prove this with Monte-Carlo simulations as explained in the next section. This method can, thus, be helpful in proposing a low complexity decoder for group-LDPC codes i.e. instead of using a single instance of a check node with a certain value of $n_m$, employ $N_d$ instances of the check node with a very low value of $n_m$. This
results in an increase in the number of processes but it reduces the overall complexity of the decoder as the complexity at each process separately is significantly reduced. The effect on the over-all complexity is dependent on the number of instances $N_d$ and the value of $n_m$. We discuss this in section 3.2.

Another important point to note is the permutations applied to the sequence order of the non-zero clusters $H_{ij}$. While constructing a PCM, the binary clusters $H_{ij}$ are positioned such that the code attains the best performance i.e. it has a higher minimum distance and a larger girth. However, a random permutation of the clusters may effect this performance and thus the diversity of group-LDPC may not be efficiently utilised. Thus, different permutations may result in different decoding performances and the best permutation set can be chosen either at the time of construction of the PCM or by brute force method.

3.1. Monte-Carlo simulation results

In this section, we present the decoding performance of the proposed decoder that uses the diversity of group-LDPC codes. The performance is calculated for various values of $n_m$ and $N_d$. The encoded data is BPSK modulated and transmitted over an AWGN channel. At the decoder, the number of iterations in the Tanner graph is fixed to 50. The Frame Error Rate (FER) is calculated for 100 errors in the total number of frames received, unless mentioned otherwise.

Fig. 7 presents the decoding performance for a half-rate code of length
$N = 3000$ bits and $(p_1, p_2) = (3, 6)$. Thus, the variable nodes are processed in $G(8)$ and the check nodes in $G(64)$. We compare the performance for $N_d = \{1, 2, 3\}$ and $n_m = \{6, 9, 12\}$. The decoder with $N_d = 1$ is the cluster-EMS decoder proposed in section 2. For $n_m = 12$, there is a gain of about 0.1 dB with $N_d = 2$ as compared to the decoder with $N_d = 1$. Similarly for $n_m = 9$, with $N_d = 2$ we gain about 0.15 dB. Moreover, the decoders with $N_d = 2$ tend to have a lower error floor. This can also be observed from the performance of the decoder with $N_d = 3$ and $n_m = [6, 6, 6]$. As compared
to the regular cluster-EMS decoder with $n_m = 12$ decoder, there is a loss of about 0.2dB in the waterfall region. However, this loss reduces for higher Eb/No, and with $N_d = 3$, we obtain better performance at low error rates of about $10^{-8}$. For $N_d = 2$ and $n_m = [9, 6]$, there is also a gain of about 0.05dB as compared to the regular cluster-EMS decoder with $n_m = 9$. Here it is noteworthy to mention that the FERs for $10^{-8}$ only were obtained with 10 frames in errors, due to the huge run time of the simulations.

Fig. 8 presents the decoding performance for the same code as in Fig. 7 but with a different order of variable nodes. The clusters are of size $(6 \times 4)$, and thus the variable nodes are processed in order $2^4$. With $N_d = 2$ and $n_m = 9$, we see the same performance as before i.e. better performance for higher $Eb/No$.

Fig. 9 presents the performance for a high rate code i.e. $R = 0.88$ and
Figure 9: Decoder diversity $N = 4800$, $R = 0.88$, $(p_1, p_2) = (3, 7)$

length $N = 4800$ bits. We observe that with $N_d = 2$ and $n_m = 24$, we attain the same performance as the regular cluster-EMS with $n_m = 32$. Similarly, for $N_d = 2$ and $n_m = 18$, we attain almost the same performance as the regular cluster-EMS decoder with $n_m = 24$.

Thus, we can deduct from the results that, for the same value of $n_m$, we gain in performance for a higher value of $N_d$. Therefore, we can obtain a decoder with the same performance but with a lower value of $n_m$. Moreover, while making use of the diversity of cluster-LDPC codes, we attain a lower error floor. This property is inherent to the code structure of cluster-LDPC codes. The various instances of the check nodes permute over the various local codes and we choose the output of the local code with the larger local
girth. This result in better extrinsic output values and thus better decoding performance. However, the disadvantage associated is that the number of processes are increased $N_d$ times. We discuss the effect of the value of $N_d$ on the complexity in the next section.

### 3.2. Complexity comparisons

The EMS based algorithm uses truncated message vectors of size $n_m$, the value of which should be selected such that it assures both, decoding performance and an acceptable decoding complexity. For cluster-LDPC codes, the added advantage of cluster codes is that the messages are reserved in their full format, without any truncation, at the variable nodes. Message truncation only occurs for the intermediate message computed during the F/B procedure at the check nodes. Table 1 gives a comparison of the memory requirement for various algorithms.

<table>
<thead>
<tr>
<th>Algo.</th>
<th>Variable Nodes</th>
<th>Check Nodes (Input/Output)</th>
<th>Check Nodes (Intermediate)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$GF(q)$-EMS</td>
<td>$n_m$</td>
<td>$n_m$</td>
<td>$n_m$</td>
</tr>
<tr>
<td>Cluster-EMS</td>
<td>$2^{p_1}$</td>
<td>$2^{p_1}$</td>
<td>$n_m$</td>
</tr>
<tr>
<td>Our Algo</td>
<td>$2^{p_1}$</td>
<td>$2^{p_1}$</td>
<td>$n_m$</td>
</tr>
</tbody>
</table>

Table 1: Size of LLR-vectors

We can see from Table 2, that the variable node process requires only a simple adder. A degree $d_v = 2$ node of order $2^{p_1}$, will execute the process in $2^{p_1}$ cycles. For a higher degree node, it is increased $(d_v - 1)$ times. With a parallel architecture, using $2^{p_1}$ processing units, the update process requires only
a single clock cycle. This makes our proposed algorithm a good candidate for high rate applications.

<table>
<thead>
<tr>
<th>Algo.</th>
<th>Min</th>
<th>Real Addition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$GF(q)$-EMS</td>
<td>$nm(n_m + 2)$</td>
<td>$n_m$</td>
</tr>
<tr>
<td>Cluster-EMS</td>
<td>-</td>
<td>$2^{p_1}$</td>
</tr>
<tr>
<td>Our Algo.</td>
<td>-</td>
<td>$2^{p_1}$</td>
</tr>
</tbody>
</table>

Table 2: Number of operations at a variable node with $d_v = 2$

Table 3 shows that the number of operations at the check nodes increased by a factor $N_d$, which however is reduced by using a smaller value of $n_m$. Here $n_{op}$ is the number of operations required to compute the lowest $n_m$ LLR combination values. It is generally considered $n_{op} = 2n_m$ as proposed in (41).

<table>
<thead>
<tr>
<th>Algo.</th>
<th>Min</th>
<th>$GF(q)$-Addition</th>
<th>Real Addition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$GF(q)$-EMS</td>
<td>$3(d_c - 2)n_{op}\sqrt{2n_{op}}$</td>
<td>$3(d_c - 2)n_{op}$</td>
<td>$3(d_c - 2)(n_{op} + n_m)$</td>
</tr>
<tr>
<td>Cluster-EMS</td>
<td>$3(d_c - 2)n_{op}\sqrt{2n_{op}}$</td>
<td>$3(d_c - 2)n_{op}$</td>
<td>$3(d_c - 2)(n_{op} + 2^{p_1})$</td>
</tr>
<tr>
<td>Our Algo.</td>
<td>$3(d_c - 2)n_{op}N_d\sqrt{2n_{op} + N_d}$</td>
<td>$3(d_c - 2)n_{op}N_d$</td>
<td>$(3(d_c - 2)(n_{op} + 2^{p_1}))N_d$</td>
</tr>
</tbody>
</table>

Table 3: Number of operations at a check node

The total silicon area of the decoder is increased when several instances of the check update process scrambled in parallel are added to the decoding procedure. However, by using a smaller value of $n_m$, we require smaller size components and thus the overall surface area is reduced. To get a complete and accurate idea on the comparison of the surface area for various values of $N_d$ and $n_m$, a physical implementation based on the RTL description is
needed. However, we have not realized the hardware implementation and here we present only an analytical comparison of the decoders based on perception and experience.

If we consider the serial architecture proposed in (37), the check node process compounds almost half of the decoder area. The remaining surface area is occupied by the memory elements, variable and function nodes update process, interleaver and other processes. We need to examine the effect of two factors; $N_d$ and $n_m$. Since the check update process constitutes almost half of the decoder area, for $N_d = 2$ and keeping the same value of $n_m$, the overall area is increased by a factor of 1.5 as shown in Fig. 10(a). On the other hand, the value of $n_m$ has a heavier effect on the surface area as it effects the input/output registers and sorter used in the check update elementary process. Moreover, more memory elements are required to store the intermediate messages in the F/B procedure. Hence, it can be said that doubling the value of $n_m$ results in an almost double increase in the surface area as shown in Fig. 10(b).

Using the above mentioned analysis, it can be stated that two check-update processes in parallel and the value of $n_m$ reduced by a factor of 1.5 will result in the same surface area as the regular cluster-EMS decoder. For example, cluster-EMS decoders with \{\(N_d = 1, n_m = 12\)\} and \{\(N_d = 2, n_m = 9\)\} will result in the same surface area for both decoders.
4. Conclusion

The Extended Min-Sum (EMS) algorithm applied to NB Cluster-LDPC codes offer very good performance, especially in the error floor region, with a low complexity. This can be further improved by making use of the diversity offered by NB Cluster-LDPC codes. Applying a random permutation to the sequence of the local codeword at a check node results in a different local Tanner graph, and hence a different convergence behaviour. We apply multiple instance of a single check node with different Tanner graphs and select the best output LLR-value for each symbol from amongst them. This results in better decoding performance but increases the complexity. The overall complexity is reduced by reducing the complexity at each instance. We see that with our proposed method, we have a lower error-floor with equal or lower complexity as compared to regular the cluster-EMS algorithm. With a suitable selection of the EMS truncation value $n_m$ and number of parallel...
instances $N_d$, we can have good error correction capability with a suitable surface area of the decoder.


[34] V. Savin and D. Declercq. Linear growing minimum distance of ultra-sparse non-binary cluster-ldpc codes. In in the proc. of ISIT’11, St. Petersburg, Russia, August 2011.


[38] A. Goupil, M. Colas, G. Gelle, and D. Declercq. Fft-based decoding of
general ldpc codes over abelian groups. IEEE Transactions on commu-

[39] L. Sassatelli and D. Declercq. Non-binary hybrid ldpc codes - structure,
decoding and optimization. In Proceedings of IEEE ITW’06, Chengdu,

[40] L. Sassatelli and D. Declercq. Analysis of non-binary hybrid ldpc codes.

MobileSummit 09, Santander, Spain, June 2009.

[42] Xiaofei Huang, Suquan Ding, Zhixing Yang, and Youshou Wu. Fast min-
sum algorithms for decoding of ldpc codes over $gf(q)$. In Proceedings of

[43] D. Declercq. Non-binary group decoder diversity for decoding dense
block codes. In Proceeding of IEEE Turbo-coding symposium, Lausanne,