Non-binary Split LDPC Codes defined over Finite Groups

Bilal Shams *, David Declercq †, Vincent Heinrich ‡

* ETIS/ENSEA/UCP/CNRS-UMR8051
95014, Cergy-Pontoise, France
bilal.shams@ensea.fr
† david.declercq@ensea.fr
‡ ST-Microelectronics
38926, Crolles, France
vincent.heinrich@st.com

Abstract—In this paper, we propose a practically implementable decoding algorithm for split LDPC codes with parity constraints defined over finite groups. The proposed decoding algorithm generalizes the orders of the variable and check nodes such that it may have messages of different order at the various nodes. This gives us a further degree of freedom in terms of better code construction. Using the binary image of the parity check matrix, we define the function node which maps lower order messages to higher order and vice-versa. In order to have a reduced complexity decoder which is practically implementable, we use the truncated messages concept at the check nodes and evaluate its performance. We show improved performance in the error floor region as compared to other non-split low complexity decoding algorithms.

I. INTRODUCTION

Binary Low Density Parity Check (LDPC) codes, first discovered by Gallager [1], show capacity approaching performance when the code size is very large. For moderate length codes, the performance can be improved by using non-binary LDPC (NB-LDPC) Codes which, however, offer an increased decoding complexity [2]. Consequently, a lot of research has been carried out on decreasing the decoding complexity for NB-LDPC codes as well as the construction of better codes for improved performance.

In this context, a new family of LDPC codes named as split codes was introduced in [3] which provides a solution for decreasing the memory required for decoding LDPC codes. The basic concept of split codes is that the variable and check nodes are not processed in the same order of messages, rather the variable nodes are processed in a lower order, reducing memory requirements at the variable node. A symbol in \( GF(2^p) \) can be written in a \((p \times p)\) binary matrix form [4]. This matrix is split into sub-matrices, splitting symbols into sub-symbols and lowering the field order at the variable node. However, the check nodes are processed in \( GF(2^p) \), in order to gain the advantages of a \( GF(2^p) \) code. Therefore, at the check nodes, messages of the lower field order have to be combined to form messages of order \( GF(2^p) \). On the contrary, at the variable nodes, the messages in \( GF(2^p) \) have to be marginalized to the lower order. This leads to a decrease in the memory required at the variable nodes since lower sized vectors are used to store the probabilities.

In this paper, we generalize the parity constraints of split codes to finite groups i.e. the parity check equations are defined over finite groups instead of finite fields [5]. This gives us the opportunity to replace the finite field multiplication with a general linear operator [6]. This generalization of the parity constraints gives us a further degree of freedom in terms of code construction as the various nodes of the code may belong to finite sets with different orders [7].

In section 2, we discuss the generalization of the parity constraints to groups and their associated function nodes. In section 3, we discuss the function nodes for split codes defined over groups. Here we also discuss the generalization of the splitting order along with the decoding algorithm for split codes. In order to simplify the decoding procedure, we adapt the message truncation concept, also termed as EMS [8], for split codes. In section 4, we report the simulation results comparing the performance of our proposed algorithm with the regular-EMS algorithm for various codes.

II. PARITY CHECK CONSTRAINTS OVER FINITE GROUPS

With codes defined in a finite field, the parity check equations comprise of multiplications with the field elements. However, with codes defined on finite groups, the parity constraints can be generalized by a function [9]. This allows us to consider a wider class of parity check codes compared to codes defined in a finite field [6]. The parity equation in the general case is:

\[
\sum_i f_{ij}(v_i) = O \quad \text{in } G
\]

where \( f_{ij}(.) \) is the general function over the group \( G \) which can be either linear or non-linear. It has the following properties [5]:

- For the case when the parity check codes are defined in a finite field \( GF(q) \), the function \( f_{ij}(.) \) is linear and
represents a circular permutation which corresponds to the multiplication by the non-zero field element $b_{ij} \in GF(q)$.

- When the code is defined in a group $G(q)$, any closed function from $G(q)$ to $G(q)$ can be used.
- A special case of interest for codes defined in a group $G(q)$, is when the function $f_{ij}(.)$ is linear and it has a binary matrix representation of size $(p \times p)$. This is a generalized case of the codes defined in a finite field because it can either be a cyclic permutation or a random projection depending on whether the matrix is full-rank or rank-deficient.

Another advantage of the generalization of the parity equations to groups is that it does not affect the decoding complexity [6]. However, as we make use of the binary images of group elements, we have to consider groups whose cardinality $q$ is a power of 2. In this paper, we use codes defined over finite groups instead of finite fields [9], [10].

### A. Binary Images and Bit clustering

Let $\{b_{kj}[i]\}_{i=1..p}$ be the binary map of the symbol $c_{kj}$ forming the binary vector $b_{kj}$ and $H_{ij}$ as the $(p \times p)$ binary image of a linear function $f_{ij}(.)$ from $G(q)$ to $G(q)$. The binary image of a single parity check equation in the matrix form is written as [4]:

$$
\sum_{j=0}^{d_p-1} H_{ij}.b_{kj} = O_p
$$

(2)

with $O_p$ being the all zero vector of length $p$. The binary image of such a parity check equation can also be seen as a matrix of size $(p \times d_p.p)$, which forms a local component code. This code can then be decoded in two ways: (i) treat locally and decode using a local binary decoding algorithm (ii) treat as a single parity-check in a finite non-binary group and decode using the BP equations over its non-binary image.

We use the second method for decoding. The non-binary image is computed using a bit clustering process [5]. A cluster is defined as the $(p \times p)$ sub-matrix $H_{ij}$ in the binary image $H_b$ of the whole parity check matrix. Each time a cluster contains a non-zero value, an edge is created in the Tanner Graph which connects the variable node to its corresponding check node. A linear function $f_{ij}(.)$ is associated to each edge forming a closed function from $G(q)$ to $G(q)$ and it has $H_{ij}$ as its matrix representation. The function corresponds to a linear mapping within the considered group, such that:

$$
\alpha_n = f(\alpha_m)
$$

(3)

with $\alpha_n, \alpha_m \in G(q)$. It is obtained using:

$$
\{b_{\alpha_n}[k]\}_{k=1..p} = H_{ij}.\{b_{\alpha_m}[k]\}_{k=1..p}
$$

(4)

where $\{b_{\alpha_n}[k]\}_{k=1..p}$ and $\{b_{\alpha_m}[k]\}_{k=1..p}$ are the binary representations $\alpha_n$ and $\alpha_m$, respectively. The matrix $H_{ij}$ can either be a full rank matrix or it can be rank-deficient. If it is full rank, the function corresponds to a random permutation of message values and all the elements of the message are mapped by the function, whereas when it is rank deficient, the function is not a full mapping i.e. not all the elements of a message are projected by $f_{ij}(.)$.

Fig.1 shows two different non-zero binary clusters with their respective mappings and explains the two different cases of projection. The binary cluster in Fig.1.a shows a full rank matrix with a full-rank projection where all the elements of the message are mapped. The binary cluster in Fig.1.b represents a rank-deficient projection where some elements are projected more than once, leaving others un-projected, which are then considered to be null-valued.

### III. SPLIT CODES

#### A. Function nodes for Split Codes

We extend the concept defined in the previous section to split codes by considering rectangular binary clusters. Instead of considering a square cluster of size $(p \times p)$, we consider a cluster of size $(p_2 \times p_1)$, with $p_1 < p_2$. The function then makes a projection from a message of order $G(2^{p_1})$ to a message of order $G(2^{p_2})$.

An example of a projection of an order-4 message to order-16 can be seen in Fig. 2 which has a rectangular binary cluster. As can be observed, $(2^4 - 2^2)$ elements are left un-projected in the order-$2^4$ message and they are then considered to be null-valued.

This procedure thus allows us to map an order-$G(q_1)$
message $U_{vp}$ of a variable node to an order-$G(q_2)$ message $U_{pc}$ at the check nodes input:

$$U_{pc}[\beta_j] = \begin{cases} U_{vp}[\alpha_i], & \text{if } \beta_j = f_{ij}(\alpha_i) \\ 0, & \text{elsewhere} \end{cases}$$  \hspace{1cm} (5)$$

where $\beta_j \in G(q_2)$ and $\alpha_i \in G(q_1)$.

Likewise, in the reverse direction, the messages of order-$2^{p_2}$ from the check nodes can be mapped back to messages in order-$2^{p_1}$. This results in a memory reduction at the variable node by a factor of $(2^{p_2} - 2^{p_1})$ for each message.

### B. Splitting Order of Variable and Check Nodes

This concept generalizes the splitting order of symbols i.e. the various symbols nodes can be split to various orders $2^{p_1}, 2^{p_2}, ..., 2^{p_n}$, where $p_1 \neq p_2 \neq p_3 \neq ... \neq p_n$. Likewise, the order in which the check nodes are processed can also be generalized. For this purpose, the cluster size has to made variable for the various check nodes. However, here we have to note that for a single check node all the input messages must be of the same order. Fig. 3 shows a parity check matrix with different order of processing for the variable nodes ($2^{p_1}, 2^{p_2}, 2^{p_3}$) and the check nodes ($2^{p_2}, 2^{p_3}, 2^{p_n}$). We can observe that for the first check nodes all the variable nodes messages of orders ($2^{p_1}, 2^{p_2}, 2^{p_3}$) are mapped to order $2^{p_2}$ by the function nodes. Similarly, the various other function nodes map messages of different orders between the check nodes and the variable nodes.

The generalization of the order of variable and check nodes gives us freedom in terms of the code-construction and helps in the development of better codes. There is, however, a disadvantage associated; for a fixed order $p_2$ of check nodes, while splitting the symbol nodes to order $p_1$, the degree of check nodes is increased by a factor of $p_2/p_1$, hence increasing the check nodes processing complexity. However, this increase in complexity can be balanced by making use of the $(2^{p_2} - 2^{p_1})$ zeros in a message vector at the check nodes input. The memory requirements at the check nodes are also reduced by storing the messages in $2^{p_1}$-sized vectors, neglecting the null values in the input messages.

A change in the order of the check nodes effects the number of check nodes. They are inversely proportional to each other; an increased check nodes order decreases the number of check nodes, whereas a decreased order increases the number of check nodes.

### C. Decoding Algorithm

The Tanner graph for Split Codes generalized to groups is shown in Fig. 4. For the purpose of simplicity, all the variable nodes are processed in order $2^{p_1}$ and all the check nodes in order $2^{p_2}$. The function nodes $f_{ij}(\cdot)$ maps $2^{p_1}$-order messages to an order of $2^{p_2}$ while they are being passed from the variable nodes towards the check nodes and vice-versa in the reverse direction.

We consider the message values as being Log Density Ratios (LDRs) of the probabilities of the symbols. The decoder is initialized with information from the channel. The decoding algorithm consists of four main iterative steps.

- **Variable Nodes Update**: An output message on an edge of a variable node is calculated as the sum of the channel likelihoods and all the other input messages excluding the message from the edge for which the output is being calculated.

- **Function Nodes Update**: The function nodes are updated according to Eq.(5) where messages in $G(q_1)$ are mapped to order $G(q_2)$. For each function node, there may exist a separate function depending on its binary cluster.
messages are truncated by considering the make use of concept used in the EMS algorithm where decoding algorithms for NB-LDPC Codes. Therefore, they are, however, compatible with other low complexity computational complexity rendering the decoder practically implementable. We refer to our algorithm as split-EMS.

In Fig. 5, we show the decoding performance of our split-EMS algorithm versus the non-split regular EMS algorithm. We consider a length 576-bits LDPC code of rate $R = 1/2$ and defined in $G(64)$. The cluster size for the split codes is $p_1 = 3$ bits which allowed to process the variable nodes in messages of order-8. The messages are passed between the nodes using the Flooding schedule with a maximum of 100 iterations per codeword. We verified the performance for $n_m = 12$ and $n_m = 18$ for the two algorithms. We observe a loss of around 0.3dB and 0.4dB for $n_m = 12$ and $n_m = 18$ respectively in the water-fall region for split-EMS. However, the performance gap between the two algorithms becomes smaller at higher $E_b/N_0$ and it seems that split codes may outperform the regular-EMS for very high $E_b/N_0$.

In Fig. 6 we compare the performance of the two algorithm for larger length code i.e. 2304-bits. The check nodes were processed in order-64, however, the variable nodes for split codes are processed in order-16 i.e. with $p_1 = 4$. We verified the performance again for $n_m = 12$ and $n_m = 18$. There is a loss of 0.2dB for both $n_m = 12$ and $n_m = 18$ in the water-fall zone for split-EMS as compared to the EMS algorithm. However, for higher $E_b/N_0$, we clearly see an improved performance by the split-EMS as compared to the EMS algorithm. This improved performance can be explained by the fact that split NB-LDPC codes have a binary parity check matrix which is locally less dense, resulting in a larger girth (length of the shortest cycle in the code).

In Fig. 7, we compare the performance of two split codes with different splitting orders and the EMS algorithm. The two split codes are characterized by cluster sizes $p_1 = 3$ and $p_1 = 4$ respectively. The NB-LDPC code is defined in $G(64)$ and has a length of $N_0 = 3000$ bits with $R = 1/2$. We see the same kind of performance difference as before i.e. loss of performance in the water-fall region and improved performance in the error-floor region. However, an important
point to note here is that the same split code with a splitting order $p_1 = 4$ performs better than $p_1 = 3$. This proves that choosing a proper splitting order plays a vital role in the decoding performance of split codes.

In Fig. 8, we compare the performance of a code of length 4800-bits with $R = 0.88$. We also changed the processing order of the check nodes for split-EMS. The check nodes of the split codes are processed in messages of order-128 as compared to the regular-EMS whose check nodes are processed in messages of order-256. The cluster size of split codes are of size $(7 \times 4)$ as compared to the binary cluster size of the regular-EMS of $(8 \times 8)$. We compare the performances for $n_{m} = 20$ and $n_{m} = 32$. There is a loss in performance of 0.05dB for $n_{m} = 20$. However, for $n_{m} = 32$, Split-EMS algorithm outperforms the EMS algorithm. This is because the check nodes of split-EMS are processed in a smaller field order and while considering $n_{m} = 32$ for both algorithms, we lose less information as compared to the EMS algorithm. Secondly, the number of check nodes have increased because of the decreased order of the check nodes, which increases the error correcting capability of the code. Thus, not only choosing a proper order for the variable nodes improves the performance, a proper order for the check nodes also plays a pivotal role in the decoding performance of split codes.

V. CONCLUSIONS

Split codes were introduced with the aim to reduce the memory required for decoding by treating the variable and check nodes in different orders. We proposed a decoding algorithm for split codes which is practically implementable. We generalized the parity constraints of the check nodes to groups enabling us to have general linear function nodes in the Tanner graph. These linear functions map messages from one order to another. In order to have reduced-size vectors at the check node as well, the message truncation concept was used. We showed that choosing a proper splitting order for the variable and check nodes effects the decoding performance. Low memory requirements, less check nodes processing complexity and better performance in the error-floor region make the proposed decoder a good candidate for hardware implementation.

REFERENCES