DEMOCRATIC AND POPULAR REPUBLIC OF ALGERIA

MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

Ecole Nationale Polytechnique

Electronic Department
Laboratoire des Dispositifs de Communication et de Conversion Photovoltaique

Submitted in partial fulfillment of the requirements for the Master Degree

---

Design of Low Complex Non Binary LDPC Decoder using Min-Max algorithm

---

HARABI Kamel-eddine

Supervised by : Mr. M.TAGHI

Presented on : 02/07/2017

Jury members :

President	Mr. S. AIT CHEIKH	Professor ENP
Examiner	Mr. D. BERKANI	Professor ENP
Supervisor	Mr. M.TAGHI	Assistant Professor ENP

ENP 2017

DEMOCRATIC AND POPULAR REPUBLIC OF ALGERIA

MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

Ecole Nationale Polytechnique

Electronic Department
Laboratoire des Dispositifs de Communication et de Conversion Photovoltaique

Submitted in partial fulfillment of the requirements for the Master Degree

---

Design of Low Complex Non Binary LDPC Decoder using Min-Max algorithm

---

HARABI Kamel-eddine

Supervised by : Mr. M.TAGHI

Presented on : 02/07/2017

Jury members :

President	Mr. S. AIT CHEIKH	Professor ENP
Examiner	Mr. D. BERKANI	Professor ENP
Supervisor	Mr. M.TAGHI	Assistant Professor ENP

ENP 2017

Abstract

ملخص

شفرات اختبار التكافؤ منخفضة الكثافة (LDPC) المعرفة على حقل غالوا GF(q)، والتي تسمى أيضا شفرات اختبار التكافؤ الغير ثنائية(NB-LDPC)، هي عبارة عن امتداد لشفرات LDPC الثنائية مع أداء أفضل بكثير. نادرا ما يناقش تصميم وإنشاء مفكك التشفيرLDPC الغير ثنائي بسبب التعقيد في الإنشاء على مستوى الأجهزة. في هذا المشروع، ركزنا على تصميم مفكك شفرات NB-LDPC منخفض التعقيد باستخدام خوارزمية الحد الأدنى-الحد الأعلى (Min-Max) .
المساهمات الرئيسية لهذا العمل تتوافق مع تصميم الأجزاء الأساسية للمفكك مثل عقدة الاختبار(CN)، عقدة المتغير (VN) ومقرر الشفرة وذلك باستخدام تقنيات ذات كفاءة. تم تفصيل تصميم مفكك التشفير ومكوناته، وتم عرض وتوثيق تفاصيل مختلفة مثل المخططات ووظائف المكونات وأمثلة للشرح.
الكلمات المفتاحية : شفرات اختبار التكافؤ منخفضة الكثافة (LDPC) ,حقل غالوا GF(q), شفرات اختبار التكافؤ الغير ثنائية(NB-LDPC)، مفكك التشفير, تصميم, بنية .

Résumé

Les codes contrôle de parité à matrices creuses (Low-Density Parity-Check, LDPC) définis sur les corps de Galois GF (q), également appelés codes LDPC non binaires (NB-LDPC), sont une extension de codes binaires LDPC avec des performances nettement meilleures. La conception et l'implémentation des décodeurs NB-LDPC ont rarement été discutées en raison de leur complexité de l'implémentation matérielle. Dans ce projet, on est concentré sur la conception d'une architecture à faible complexité pour un décodeur NB-LDPC basé sur l'algorithme de décodage Min-Max. Les principales contributions de ce travail correspondent à la conception des blocs de base du décodeur, comme le bloc de nœud de contrôle (CN), le bloc de nœud variable (VN) et le bloc décision de code mot avec des techniques d'efficacité. La conception du décodeur et ses composants de base sont détaillés. De nombreux détails, comme les schémas de blocs et les fonctionnalités des composants, ainsi que des exemples d'explication, ont été présentés et documentés.
Mots cl´es: LDPC, Corps de Galois, NB-LDPC, Min-Max, Décodeur, Architecture, Conception, Implémentation.

Abstract

Low-density parity-check (LDPC) codes constructed over the Galois field GF(q), which are also called Non-Binary LDPC (NB-LDPC) codes, are an extension of binary LDPC codes with significantly better performance. The design and implementation of NB-LDPC decoders has rarely been discussed due to their hardware implementation complexity. In this project, we focused on the design of low complex architecture for a NB-LDPC decoder using the Min-Max decoding algorithm.. The main contributions of this work correspond to the design of the decoder basic blocks, as the check node (CN) block, the variable node (VN) block and the codeword decision block with efficiency techniques. The design of the decoder and its components are detailed, Various details like block schematics and the components functionality and explanation examples have been presented and documented.
Keywords : LDPC, Galois Field, NB-LDPC, Min-Max, Decoder, Architecture, Design, Implementation.

Dedication

I dedicate this work to my family, my dear parents for all their sacrifices for my education, to all my friends, and all people who taught me along my career.

Acknowledgments

I would like to express our gratitude towards our supervisor, Mr. M.TAGHI. for his valuable advices, encouragement and guidance throughout the course of this project.

I would also like to thank Mr. S.AIT CHEIKH and Mr. D.BERKANI for their support as members of jury.

Contents

List of Figures

List of Tables

Abbreviations

APP	A Posteriori Probability
ASK	Amplitude-Shift Keying
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
BP	Belief Propagation
BPSK	Binary Phase-Shift Keying
CN	Check Node
DVB	Digital Video Broadcast
DU	Decision Unit
ECN	Elementary Check Node
EMS	Extended Min-Sum
EVN	Elementary Variable Node
FER	Frame Error Rate
FPGA	Field Programmable Gate Array
GF	Galois Field
HD	Hard Decision
LDPC	Low-Density Parity-Check
LLR	Log-Likelihood Ratio
LTE	Long Term Evolution
PSK	Phase-Shift Keying
RAM	Random Access Memory
VN	Variable Node
WiFi	Wireless Local Area Network
WiMAX	Worldwide Interoperability for Microwave Access

General Introduction

Low Density Parity-Check (LDPC) codes are a class of linear block codes. They have been successfully included in numerous standards such as DVB-S2 [5], IEEE 802.16e and IEEE 802.11n , among others. These codes were first proposed in the 1962 PhD thesis of Gallager at MIT. But they remained largely neglected for over 35 years, because of the computational power to exploit iterative decoding schemes was not available until recently. The main reasons for their success are that their performance are close to the channel capability for long codewords [6].

Non-Binary LDPC (NB-LDPC) codes are an extensions of binary LDPC codes. These codes perform better than the binary LDPC codes in case of codes with low and medium codeword length. Despite the error-correcting performance advantages, NB-LDPC codes suffer from high decoding complexity. During the last decade, significant progress has been made in the development of low-complexity NB- LDPC decoding algorithms and the implementation of these algorithms in flexible dedicated very-large-scale integration (VLSI) circuits. The graphical representation of the NB-LDPC codes can be used in the implementation of these algorithms whose effectiveness has been shown on graph models such as the Belief Propagation algorithm generally noted BP. This algorithm guarantee optimal decoding performances but it has not great interest for a hardware implementation. Consequently, other algorithms based on approximations of the BP algorithm have been developed with the aim of ensuring a reasonable performance/complexity compromise. The well known ones are the Min-Sum, its variant Extended Min-Sum (EMS) and the Min-Max algorithm. The last one can be implemented by a more efficient architecture then the others with small performance degradation.

The objective of our project is to design a low complex NB-LDPC decoder based on Min-Max decoding algorithm. In particular, we provide concepts and solutions that enable flexible implementation and a compromise between decoding speed and implementation complexity, which are the basic requirements of modern communication standards.

Chapter 1 : NB-LDPC Decoder Theory

This first chapter aims to present fundamental principles and concepts that will be useful for the understanding and implementation of NB-LDPC codes.

This chapter begins by presenting the LDPC codes with brief explanation than present the NB-LDPC codes, and finally discuss the decoding algorithm with giving some details and examples about the Min-Max algorithm.

1.1. System model

Figure 1.1 depicts a high level block diagram of a simple channel coding system for LDPC codes, the massages are bits or symbols.

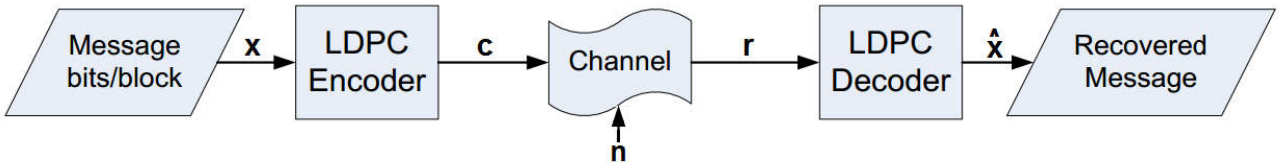

Figure 1.1. high level block diagram of channel coding system for LDPC

In what follow, for an (*N, K*) LDPC we will use :

• Information word *x* of length *K*.
• Code word *c* of length *N*.
• Received word *r* of length *N*.
• Decoded code word ĉ of length *N*.
• Decoded information word *x* of length *K* .
• code rate *R = K/N*

1.2. LDPC codes

Low Density Parity Check (LDPC) codes make up a class of linear block codes that are characterized by a sparse parity-check matrices *H* which mean that it contain only a very small number of non-zero entries[6]. They were first proposed in the 1962 PhD thesis of Gallager at MIT. in this section we are presenting fundamentals and characteristics of the LDPC codes .

1.2.1. Representation of LDPC Codes

There are two ways to represent LDPC codes. Like all linear block codes [8], they can be described via matrices. The second way is via a graphical description.

Matrix Representation

The LDPC code can be described by two basic matrix the generator matrix *G* of dimension *K* × *N* and the parity check matrix H of dimension *M* × *N* , with *K* is the length of the data before the coding process and *N* is the length after the coding , *M = N - K* is the number of

parity check equations or the bits of the parity. The relation between these two matrix is $H.\,G^T = 0$, which is the basic relation in the decoding process.

$$H = \begin{bmatrix} 0\,1\,0\,1\,1\,0\,0\,1 \\ 1\,1\,1\,0\,0\,1\,0\,0 \\ 0\,0\,1\,0\,0\,1\,1\,1 \\ 1\,0\,0\,1\,1\,0\,1\,0 \end{bmatrix} \qquad (1.1)$$

The illustrated matrix H in Eq.1.1 is a parity check matrix with dimension *N* x *M* for a *(8, 4)* code. The number of non-zeros element ('1's in binary codes) in a row of *H* is called the row weight 'w_r' , and the number of non-zeros element ('1's in binary codes) in a column is referred to as the column weight 'w_c' .

Graphical Representation

Tanner introduced an effective graphical representation for LDPC codes. Not only do these graphs provide a complete representation of the code, but they also help to describe the decoding algorithm as explained later.

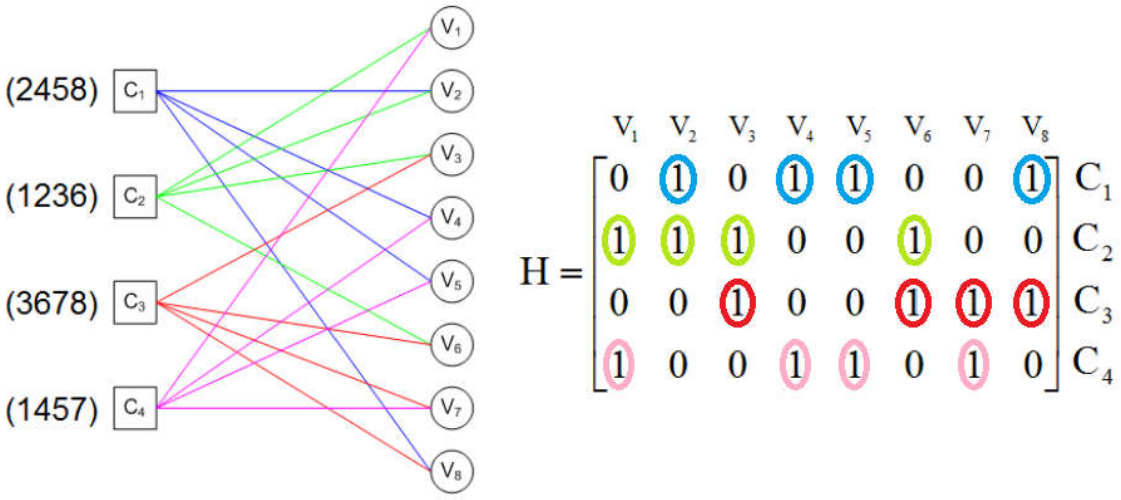

Figure 1.2. Tanner graph with corresponding H matrix for (8,4)LDPC code

Tanner graphs are bipartite graphs [9], that means the nodes of the graph are separated into two distinctive sets and edges are only connecting nodes of two different types. The two types of nodes in a Tanner graph are called variable nodes (VNs) and check nodes (CNs). Fig. 1.2. illustrates an example for such a Tanner graph and represents the same code as the matrix in H. The creation of such a graph is straight forward. It consists of M check nodes (the number of parity bits) and N variable nodes (the number of bits in a code word). Check node v_j is connected to variable node c_i if the element h_{ij} of H is a non-zero element (1 in case of binary codes).

1.2.2. Regular and Irregular LDPC Codes

LDPC code is called regular if w_c is constant for every column and $w_r = w_c \cdot (N/M)$ is also constant for every row [6]. The example in Eq.1.1 is regular with $w_c = 2$ and $w_r = 4$. We can also verify whether a code is regular or not from its graphical representation. If the number of

incoming edges is same for all check nodes and also for all variable nodes, then it is a regular code. If H is low density but the number of *1's* in each row or column isn't constant, the code is called an irregular LDPC code. The irregular LDPC codes give better decoding performance than regular ones, but the construction of the encoder as well as the decoder becomes much more complex.

1.2.3. Constructing LDPC Codes

There are several algorithms to construct suitable LDPC codes. Gallager [1] himself introduced one. Further, MacKay [4] proposed a way to semi-randomly generate sparse parity check matrices. Suitably chosen array codes also give good performance to the decoding algorithm. Constructing high performance LDPC codes is not a hard problem. In fact, completely randomly chosen codes are good with a high probability. The problem is that the encoding complexity of such codes is usually rather high.

1.2.4. LDPC Encoding Process

The objective of the encoding operation is to regroup a sequence of information symbols into words (or blocks) of equal length *K* [6], which are independently encoded than each information word uniquely mapped onto a code word of length *N*, by using the G matrix and the following operation :

$$c = G^T.x \qquad (1.2)$$

1.2.5. Decoding process

Error detection is done by computing the syndrome and check to see if it is all zeros. If it is not all zeros, then declare that an error has been detected. If it is all zeros, then assume that the codeword is correct. The syndrome s given by the Eq.1.3 :

$$H.r = H(c + n) = H.\ G^T.x + H.n = H.n = S \qquad (1.3)$$

where n is the error vector and *S* is the syndrome vector.

The main issue in the decoding operation is to find a decoding algorithm that can correct the error and find the desired estimated code word with efficient way to get good performances and simplify the decoder implementation.

1.3. NB-LDPC codes

Many error-correcting codes [7], such as low-density parity-check (LDPC) codes are defined over Galois Fields. The purpose of this section is to provide an elementary knowledge of the extension field GF(2^m) and a brief explanation for the NB-LDPC codes .

1.3.1. The Galois Field GF(2^m)

The Galois field is a finite field that consist of a specific addition and multiplication operations [10]. One way to create a Galois field is to use the set of integers *{0..., p - 1}* where *p* is prime and the (+) and (·) which are *modulo - p* addition and *modulo - p* multiplication, respectively.

The *GF*(2) is a finite field with the least number of elements *{0, 1}* that's why there are many binary codes that are defined over the binary field *GF*(2) , in this field the addition is the *XOR* operation and multiplication is the logic *AND* operation.
We can create extension fields (subfields) of the GF (2) one of these extension fields is based on power of 2 [7], the GF (2^m) where m is a positive integer and 2^m is the order (cardinality). The extension fields GF (2^m) are usually used in digital communication, since each element of Galois field GF (2^m) can be represented in a unique way in the form of a binary symbol of m bits.

1.3.2. GF(2^m) Power representation

The field elements can be represented as linear combinations of the elements in a basis (basis representation), as well as powers of a primitive element (power representation). the hardware complexities of finite field operations are heavily dependent on the element representations.
The power representation has low complexity and easy to implement. This representation is based on the primitive element of the GF (2^m), which is the root of the primitive polynomial for more details check the reference [10]. The *GF* (2^m) is the set of primitive element α powers and the element 0.

$$GF\ (2^m) = \{0, 1, \alpha, \alpha^2, ..., \alpha^{m-2}\} \qquad (1.4)$$

Table.1.1 illustrates the GF(8) power representation and the corresponding binary representation.

	α^2	α	1
0	0	0	0
1	0	0	1
α	0	1	0
α^2	1	0	0
α^3	0	1	1
α^4	1	1	0
α^5	1	1	1
α^6	1	0	1

Table 1.1 *GF* (2^3) power representation

1.3.3. *GF* (2^m) Arithmetic

Addition:

The addition of 2 elements of the *GF* (2_m) is the bit-wise *XOR*. for $x = (x_1x_2x_3)$ and $y = (y_1y_2y_3) \in GF(8)$,The addition operation of the *GF* (8) is:

$$x + y = (x1x2x3)\ XOR\ (y1y2y3) \qquad (1.5)$$

Multiplication

Using power representation, multiplication over *GF*(2_m) can be performed by adding up the exponents of the operands modulo 2^m - 1:

$$\alpha^i.\alpha^j = \alpha^{(i+j)mod(2^m-1)} \quad (1.6)$$

1.3.4. The NB-LDPC codes over $GF(2^m)$

The Non binary LDPC codes over $GF(q)$ can be seen as the generalization of binary LDPC codes over $GF(2)$, where q $=2^m$. In this case, each symbol can be represented by a *m-bit* binary tuple, In the parity check matrix H of a NB-LDPC code, the nonzero entries are elements of $GF(2^m)$. Also each information and codeword symbol is an element of $GF(2^m)$.

$$H = \begin{pmatrix} h_{0,0} & h_{0,1} & h_{0,2} & 0 & 0 & 0 \\ 0 & h_{1,1} & 0 & h_{1,3} & h_{1,4} & 0 \\ h_{2,0} & 0 & 0 & h_{2,3} & 0 & h_{2,5} \\ 0 & 0 & h_{3,2} & 0 & h_{3,4} & h_{3,5} \end{pmatrix}$$

$$h_{0,0} \cdot c_0 + h_{0,1} \cdot c_1 + h_{0,2} \cdot c_2 = 0 \quad \text{—— 1}$$
$$h_{1,1} \cdot c_1 + h_{1,3} \cdot c_3 + h_{1,4} \cdot c_4 = 0 \quad \text{—— 2}$$
$$h_{2,0} \cdot c_0 + h_{2,3} \cdot c_3 + h_{2,5} \cdot c_5 = 0 \quad \text{—— 3}$$
$$h_{3,2} \cdot c_2 + h_{3,4} \cdot c_4 + h_{3,5} \cdot c_5 = 0 \quad \text{—— 4}$$

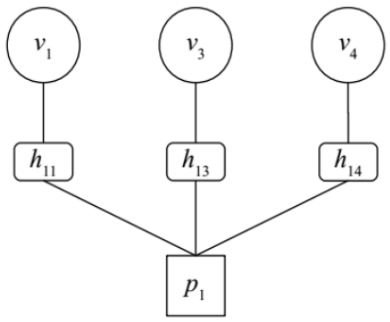

Figure 1.3. Non binary LDPC code over GF(2^m) example.

The matrix products of the parity equations are performed using the addition and multiplication operations of the Galois Field $GF(2^m)$. It is then preferable to add to the bipartite graph the new family of nodes called the permutation nodes which serve to model the multiplication of the symbols of the code word by the non-zero elements of the parity matrix *hij* . Figure 1.3 illustrates the bipartite graph for equation 2, by adding the permutation nodes that correspond to the elements $h11$,$h13$ and $h14$.

1.4. Decoding Algorithm

Actually there is more than one such decoding algorithm. There exists a class of algorithms that are all *iterative* procedures where, at each round of the algorithm, messages are passed from *variable nodes* to *check nodes*, and from check nodes back to variable nodes. Therefore, these algorithms are called ***message passing decoding algorithms***.

The basic steps of these algorithm are :

Initialization: in the first iteration the intrinsic messages received from the channel at the VNs are directly passed along the edges to CNs.

CN update : The CNs compute the extrinsic messages depending on the received messages from the VNs. then send them back to the VNs.

VN update : The VNs compute the extrinsic messages depending on the received messages from both the channel and the CNs.

Hard decision: after complete an iteration, a hard decision are made, then checking the codeword validity by using the syndrome.

These steps are repeated until it converge to a valid codeword . or maximum number of iteration is reached

1.4.1. Optimal Decoding Algorithm

One of the important message-passing algorithm is the **belief propagation algorithm**, In this algorithm the messages exchanged are a posteriori probabilities, The BP algorithm suffers from a prohibitive computational complexity. The reduced complexity decoding algorithms are built in the log-domain, and this is for the reason that it transforms the products into simple sums. In the Sum-Product Algorithm the four decoding steps are carried out in the log-domain using the Log Likelihood Ratio (LLR) [11]. Updating the CNs in the Sum-product requires a large amount of calculation and the complexity of the decoder still high.

The BP and sum-product algorithms guarantee optimal decoding performances but they have not great interest for a hardware implementation. Consequently, other algorithms based on approximations of the Sum-Product algorithm have been proposed with the aim of ensuring a reasonable performance / complexity compromise. The basic approximated algorithms are Min-Sum [12], EMS (Extended Min-Sum), and the Min-Max algorithm [13].

1.4.2. Min-Max Algorithm

The Min-max [13] algorithm can be implemented by a more efficient architecture comparison to the MS or EMS, Since the sum is replaced by the max operation in the CN computations.

We can resume the steps of the Min-Max algorithm by :

• Initialization: $$M_{v_j c_i}[\beta] = I_j[\beta] \qquad (1.7)$$

Iterations :

• Check node processing

$$M_{c_i v_j}[\beta] \approx \min_{\sum_{\substack{s\neq j \\ h_{is}\neq 0}} \theta_s = \beta} \{\max_{\substack{s\neq j \\ h_{is}\neq 0}} M_{v_s c_i}[\theta_s]\} \qquad (1.8)$$

• Variable node processing

$$M'_{v_j c_i}[\beta] = I_j[\beta] + \sum_{\substack{s\neq j \\ h_{is}\neq 0}} M_{c_s v_j}[\beta] \quad \beta \in GF(q) \qquad (1.9)$$

Normalization

$$M_{v_j c_i}[\beta] = M'_{v_j c_i}[\beta] - \min_{\beta \in GF(q)} (M'_{v_j c_i}[\beta]) \qquad (1.10)$$

• A posteriori information computation

$$APP_j[\beta] = I_j[\beta] + \sum_{h_{s,j}\neq 0} M_{c_s v_j}(\beta) \quad \beta \in GF(q) \qquad (1.11)$$

Hard decision for the jth symbol can be made as:

$$\hat{c}_j = \underset{\beta \in GF(q)}{argmax}\{APP_j[\beta]\} \quad j = 0, 1, ...N-1 \qquad (1.12)$$

The iterations can be carried out until the syndrome equals 0 or the maximum iteration number has been reached.

The LLR definition

In order to simplify the computations and deal only with positive values. the LLRs has new definition as illustrates the Eq.1.13 :

$$LLR(\beta) = -\ln \frac{p(c=\beta|r)}{\max\limits_{\theta \in GF(2^m)} \{p(c=\theta|r)\}} \quad \beta \in GF(2^m) \qquad (1.13)$$

Here the most reliable symbol has always the LLR value zero and the other symbols has a positive LLR value. as illustrated the table 1.2, an example of GF(8) elements and LLR values

GF	0	α^0	α^1	α^2	α^3	α^4	α^5	α^6
P_s	0.1	0.85	10^{-3}	10^{-7}	10^{-10}	0.05	10^{-10}	10^{-10}
-ln(P_s)	2.3	0.2	6.9	16.1	23.0	3.0	23.0	23.0
LLR_s	2.1	0	6.7	15.9	22.8	2.8	22.8	22.8

Table 1.2. Example of GF(8) LLRs values.

Before beginning the decoding process the computation of the LLR value for each symbol of the codeword is done. Considering a BPSK modulation and an Additive White Gaussian Noise (AWGN) channel, the received noisy codeword r consists of N × m binary symbols independently affected by noise. The BPSK modulation associates symbol '-1' to bit 0 and symbol '+1' to bit 1, and the BPSK demodulator output is the bit LLR value LLR(bj) of the bit bj. With the hypothesis that the GF(q) symbols are equiprobable, the LLR value of 2^m symbol LLR(2^m) is given by the equation:

$$LLR(\alpha^i) = \sum_{j=0}^{m-1} ((\alpha^i(j)) \oplus HD(b_j)) \times |LLR(b_j)| \qquad (1.14)$$

where j={0....m-1} is the bit position, HD(bj) is the hard decision of bit in position j, and $\alpha^i(j)$ bit values of the GF element α^i.

Example of GF(4) LLR values calculation :

In the GF(4), m=2 it means each symbol can be presented in 2 bits, the received LLR are :
LLR(b1)=2 ; LLR(b0) = -4, after the Hard Decision of each bit : (1,0)
the binary presentation of 0 = (0,0) then the LLR values is LLR(0) = 2
the binary presentation of α^0 = (0,1) then the LLR values is LLR(α^0) = 2 + 4 = 6
the binary presentation α^1 = (1,0) then the LLR values is LLR(α^1) = 0
the binary presentation α^2 = (1,1) then the LLR values is LLR(α^2) = 4

Chapter 2 : NB-LDPC Decoder Design

In this chapter, we will discuss the design of low complex decoder based on the Min-max algorithm for NB-LDPC codes. We will start from the decoder top level view and go one level deeper into the basic blocks. The decoder mainly consists of a check node unit, a variable node unit, and other additive blocks. An optimized scheme and corresponding architecture are developed for the check node processing. Employing this scheme, the speed of CN processing can be almost doubled when the check node degree is not small. Moreover, the computation units and the scheduling of the computations are optimized to reduce the area and decoding latency.

There are two basic design strategies, the layered and the non-layered one. The layered architecture has been chosen in our design because it has lower throughput, with lower area requirement. Our architecture is applied to a NB-LDPC code constructed over GF(8) as an example.

2.1. Global Architecture Description

From a high-level perspective, virtually all implementations of message passing LDPC decoders found in the open literature are derived from an isomorphic architecture [18] which is a direct mapping of the tanner graph.

The global architecture of decoder as illustrated in Figure 2.1 consists of different types of hardware components:

• VN unit (VNU) block and CN unit (CNU) block to compute the update equations.
• Interconnect network representing the edges of the graph and the hij/h ij multiplication block.
• Storage devices in order to save the extrinsic and the intrinsic messages.
• LLRs calculator block to calculate the GF(q)'s LLR values and the Syndrome block to check the codeword validity.

• Control unit which generate control signals in order to synchronize and control the data flow between the blocks.

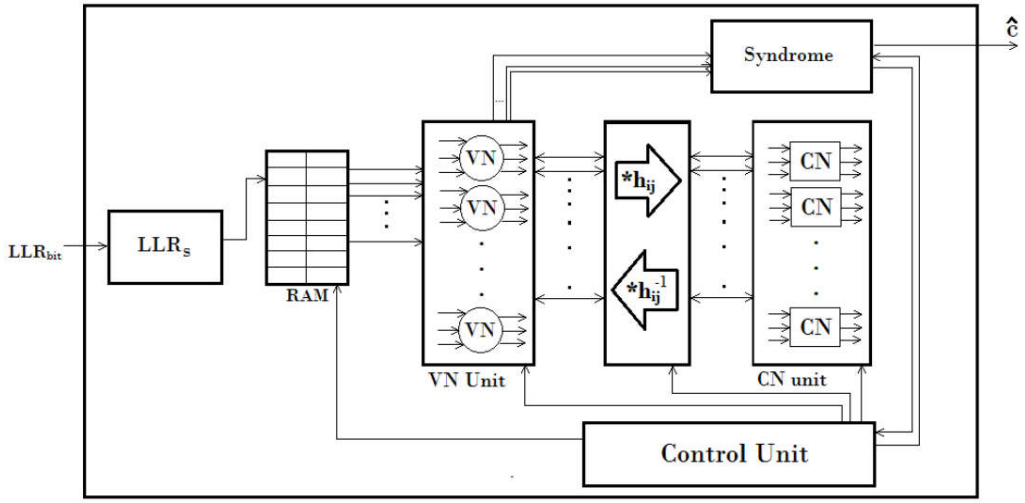

Figure 2.1. General NB-LDPC Decoder top level architecture.

Based on this prototype architecture, different implementation trade-offs are obtained through architectural transformations such as resource sharing across VNUs and CNUs and iterative decomposition of the update equations.

2.1.1. Layered architecture

The design can be partitioned into two basic architecture classes : the non-layered (Fullparallel) and the layered architecture [14,3]. The last one also can be divided into two strategies row parallel, and block-parallel. The Layered decoding has been widely adopted to reduce the memory requirement and increase the convergence speed of LDPC decoding.

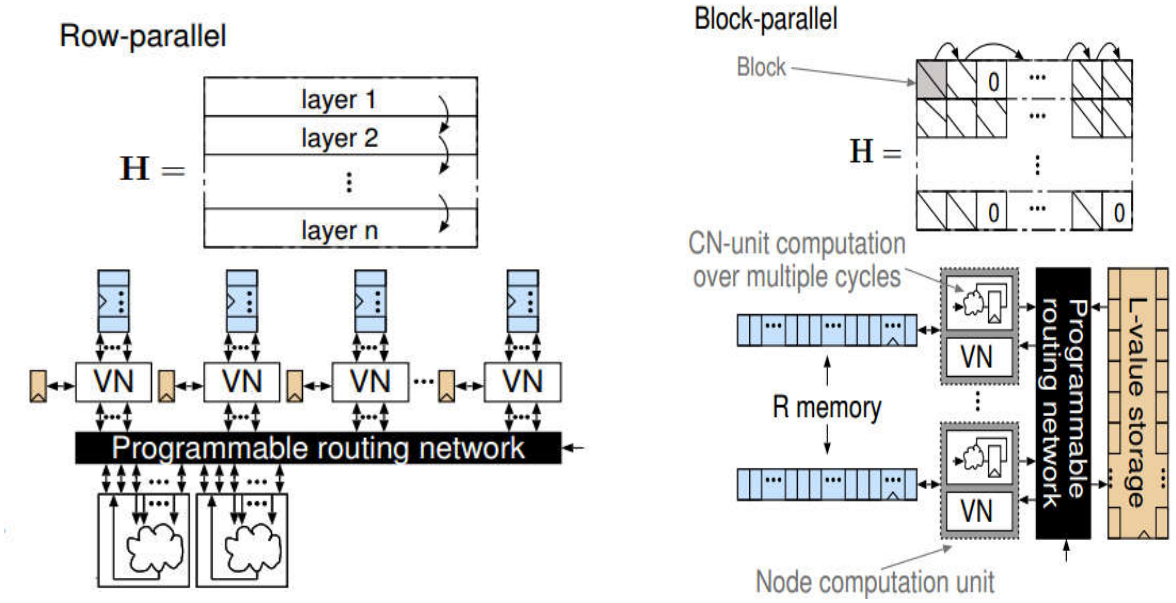

Figure 2.2. (a) Row parallel architecture, (b) Block parallel architecture.

The row-parallel design is a step towards less parallelism. The main objective is to reduce the area while maintaining very high throughput. The principle underlying row-parallel architectures is illustrated in Figure 5.2(b). Essentially, the parity-check matrix is partitioned vertically into layers. An iteration now consists of multiple cycles in which the VNUs access the messages corresponding to the current layer sequentially from a small storage array to compute the output messages and send them to the CNUs through a programmable routing network.

The row-parallel architectures provide an area advantage over full-parallel designs. The programmable routing network illustrated in Figure 5.2(b) required by the row-parallel architecture provides the flexibility to support multiple parity check matrices with a single decoder. For this reason, this architecture class has been recently considered in several flexible QC-LDPC decoders tailored to the emerging high-throughput wireless standards IEEE 802.11ad and IEEE 802.15.3c.

Block-parallel design rely on further resource sharing and further iterative decomposition. Figure 5.2(c) outlines the architectural principle, this architecture class is obtained by starting from the row-parallel approach and by partitioning the computation of a layer further into multiple cycles, corresponding to multiple blocks in the parity-check matrix. This iterative

decomposition simplifies the CN processing and allows for resource sharing also across the VNUs.

2.2. The Decoder Top Level Architecture

It has been shown that the row-parallel architecture class is highly scalable and supports the full range of throughput requirements found in modern wireless standards. Due to its favorable properties and spatially its flexibility i.e. be able to implement several given codes, without changing the design of the decoder. This architecture has been chosen for designing our decoder [16, 17].

$$H = \begin{bmatrix} 5 & 3 & 4 & 7 & 0 & 0 & 0 & 0 \\ 7 & 0 & 1 & 0 & 2 & 4 & 0 & 0 \\ 0 & 5 & 0 & 0 & 4 & 0 & 1 & 3 \\ 0 & 0 & 0 & 6 & 0 & 7 & 2 & 6 \end{bmatrix} \quad (2.1)$$

In what follow we will work on the regular (8.4)NB-LDPC code, with degree wr=4, and wc=2. The parity matrix of this code illustrated in Eq 2.1.

The row-parallel scheduling principle for the presented H matrix is illustrated in Figure 2.3, and consists in processing the CNs one by one in each iteration (CN in Tuner graph is similar to row in parity matrix). When a CN is processed, it immediately transmits the new extrinsic messages to its connected VNs, then the last ones update their extrinsic messages before moving on to the next CN update.

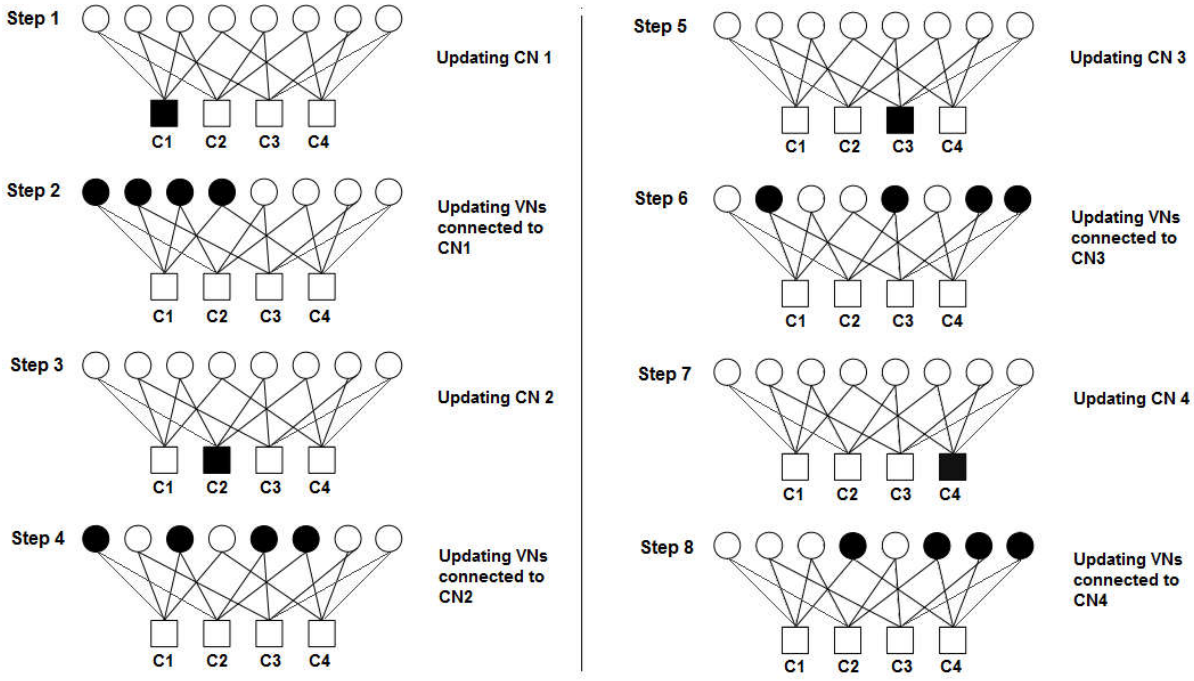

Figure 2.3. Row-Parallel scheduling principle for (8,4)NB-LDPC code.

Consequently, the row scheduling makes it possible to accelerate the convergence of the decoder, since in each iteration a CN cj benefits from the information of the previous iterations (updated by the previous CNs) and the information of the current iteration .

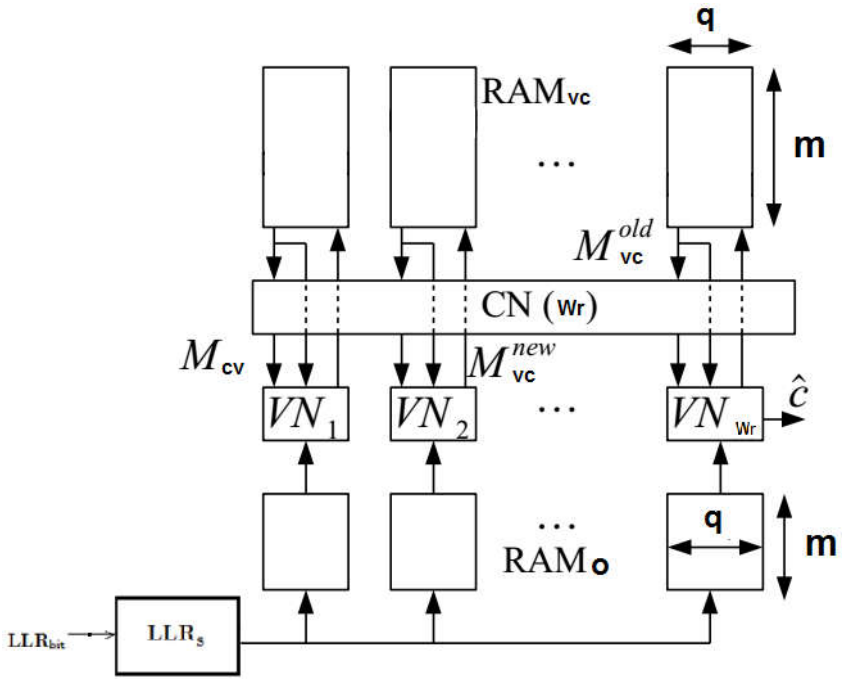

Figure 2.4. overall view of the decoder architecture.

The overall view of the decoder architecture is presented in Figure 2.4, and it is composed of a single check node block connected to wr=4 Variable Node block. The architecture also contains a memory system containing RAMvc in order to save the extrinsic messages of the VNs and of RAMo in order to save the intrinsic information I of the channel .

In the initialization step, the VNs receives the intrinsic messages from the RAMo and directly passed them to CNs. At each iteration the CN receives in parallel Mvc (v-to-c messages) old messages and provides after the CN processing the Mcv messages which will be immediately sanded and processed by the VNs to generate the new Mvc new messages that will replace the old Mvc messages in the RAMvc. This decoding process is repeated nit times. During each iteration, m updates of CN and N = wr × m updates of VN are performed. At the last iteration a decision is taken on each symbol. The codeword decision is performed in the VN processors. the VNs determine sequentially the value of the code word. We must note that since wc = 2, this decision process is done when the second edge of each VN is updated. This concludes the decoding process and the decoder then sequentially outputs ĉ to the syndrome block. 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 Mvjci(β) by Mvjci(β × hij) where the GF multiplication requires only a few XOR operations.

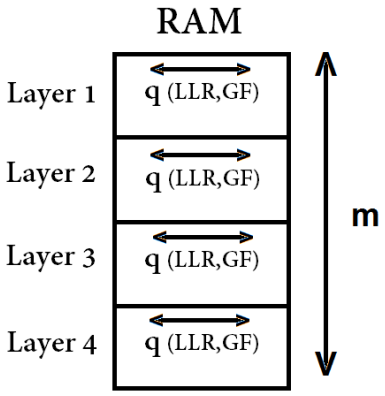

Figure 2.5. Structure of RAMs used in the decoder.

Structure of the RAMs: The intrinsic information and the Mvc messages are stored in wr memories RAMo and RAMvc respectively. Each memory bank contains information related to m messages (m is the number of layers or rows in our case), and each location containing q couples of GF(q) element and the corresponding LLR value.

The partition of the N variables in the wr memories is a coloring problem: the wr variables associated to a given CN should be stored each in a different memory bank to avoid memory access conflicts. This problem is solved by using highly structured code (as the quasi cyclic codes).

2.3. Check Node architecture

Computing the c-to-v messages in a straight-forward manner requires a complexity of $O(q^{wr-1})$. This manner requires complicated computations on GF (q) elements, and it is not suitable for efficient hardware implementation. Alternatively, the forward-backward scheme can be applied to the check node processing to avoid computing output message directly. This scheme consists in constructing the outgoing messages by a set of elementary operations, making it possible to reduce the processing latency. These elementary operations are performed by Elementary Check Node (ECN). Figure 2.6 illustrates the Forward-Backward architecture of CN with degree wr = 4. For clarity, the incoming CN messages are denoted by Mvjc and the outgoing messages are denoted by Mcvj .

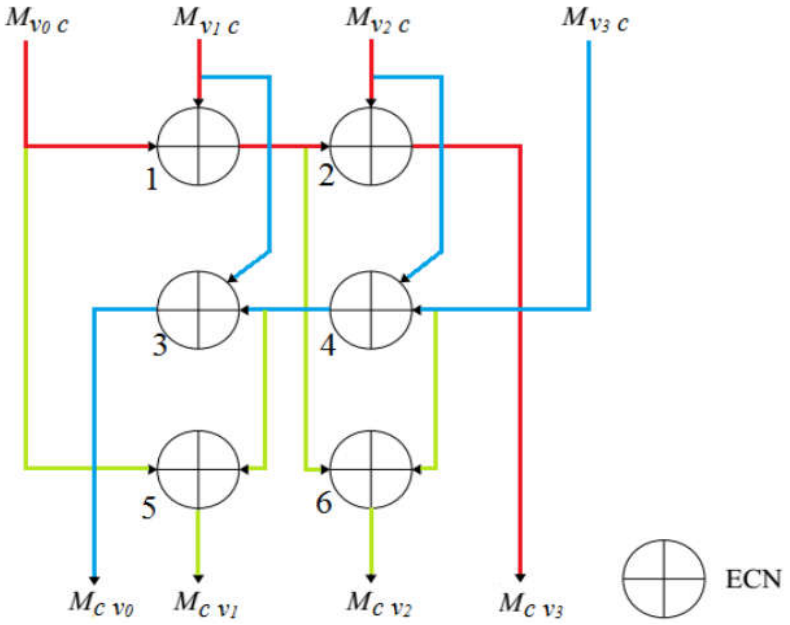

Figure 2.6. Forward-backward CN of degree wr=4 architecture.

The CN receives wr messages and generates also wr messages which will be transmitted to the VNs. This CN requires the implantation of 3(wr - 2) ECNs distributed over 3 layers, the Forward layer (the red direction), the backward layer (the blue direction) and the merging layers (the green directions). In our project we are constructing CN with degree wr = 4 that means we need to 6 ECNs. Figure 2.6 illustrate the scheme construction. The processing of the EVNs is iterative, At the beginning only the ECNs connected directly to the input data work first (ECN number 1 and 4 in the Figure 2.6), then after they complete their processing the other ECNs (2,3,5 and 6) can begin processing.

2.3.1. ECN architecture

The proposed CN architecture can be used in both cases, keeping all q messages in each vector or keeping only the nm < q most reliable messages in each vector. this can make the decoder more flexible, if the q is small we can keep all messages for better performance, in the other hand, if q is not small keeping only nm < q most reliable messages can reduce the computations latency.

The EVN compute an outgoing message vector from two incoming message vectors. The message vector consists of two parts: LLRs and corresponding GF (q) elements. Denote the LLR vectors by LA = [LA(0), LA(1), ..., LA(q - 1)] and LB = [LB(0), LB(1), ..., LB(q - 1)], and the corresponding finite field element vectors by GFA = [GFA(0), GFA(1), ..., GFA(q-1)] and GFB = [GFB(0), GFB(1), ..., GFB(q-1)], also denote the output LLR and corresponding finite field element vectors by LO and GFO. The entries in the output LLR vector for the Min-max decoding are the q minimum values of max(LA(i), LB(j)) with different GFA(i)+GFB(j) for any combination of i and j less than q. Computing the updating messages in ECN by the traditional solution can make a hardware demanding and a latency problem. To overcome this problem the ECN developed in our project which uses two incoming messages stored in the order of increasing LLR and generate sorted outgoing message is based on an efficient algorithm (our algorithm is based on an algorithm in the reference [3] with several important optimizations) that make the computations in minimum clock cycles and using

serial computation to reduce the hardware area. Two example input vectors for a NB-LDPC code over GF (8) are considered in table 2.1:

(GF, LLR)	(α^3,0)	(α^0,1)	(α^5,4)	(α^4,5)	(0,7)	(α^1,9)	(α^2,13)	(α^6,16)
(α^0,0)	(α^1,0)	(0,1)	(α^4,4)	(α^5,5)	(α^0,7)	(α^3,9)	(α^6,13)	(α^2,16)
(α^2,2)	(α^5,2)	(α^6,2)	(α^3,4)	(α^1,5)	(α^2,7)	(α^4,9)	(0,13)	(α^0,16)
(0,3)	(α^3,3)	(α^0,3)	(α^5,4)	(α^4,5)	(0,7)	(α^1,9)	(α^2,13)	(α^6,16)
(α^6,5)	(α^4,5)	(α^2,5)	(α^1,5)	(α^3,5)	(α^6,7)	(α^5,9)	(α^0,13)	(0,16)
(α^1,6)	(α^0,6)	(α^3,6)	(α^6,6)	(α^2,6)	(α^1,7)	(0,9)	(α^4,13)	(α^5,16)
(α^3,8)	(0,8)	(α^1,8)	(α^2,8)	(α^6,8)	(α^3,8)	(α^0,9)	(α^5,13)	(α^4,16)
(α^4,11)	(α^6,11)	(α^5,11)	(α^0,11)	(0,11)	(α^4,11)	(α^2,11)	(α^1,13)	(α^3,16)
(α^5,14)	(α^2,14)	(α^4,14)	(0,14)	(α^0,14)	(α^5,14)	(α^6,14)	(α^3,14)	(α^1,16)

Table 2.1. Example of ECN computation with two sorted inputs.

As illustrated in the table the most reliable q messages are distributed in the top left corner, the grey cells are the entries of the output message vector. We can see in the table a remarkable pattern for LLR distribution , based on this pattern we find a specific algorithm for the ECN processing.

Algorithm:

Initialization:
i = 0, j = 1, n = 0;
loop:
if$(L_A(i) < L_B(j))$
for k = 0 to j - 1
{ if (n = q) goto *stop*
if $(GF_A(i) + GF_B(k) \notin GF_O)$
$(L_O(n), GF_O(n)) \leftarrow (L_A(i), GF_A(i) + GF_B(k))$
n = n + 1 }
i = i + 1; goto *loop*
else
for k = 0 to i - 1
{ if (n = q) goto *stop*
if$(GF_A(k) + GF_B(j)) \notin GF_O$
$(L_O(n), GF_O(n)) \leftarrow (L_A(j), GF_A(k) + GF_B(j))$
n = n + 1 }
j = j + 1; goto *loop*

stop:

In the algorithm, n is the number of messages inserted into the output vector, i, j and k are used to indicate the positions in the input vectors, by using the table 2.1 we can explain the functionality of the Algorithm. Beginning from the top left corner of the table, a boundary is drawn to follow the comparison results of the LLRs. It goes down when the LLR in the LA < LB vector, and goes to the right otherwise. The testing result of whether LA(i) < LB(j) in the algorithm tells the direction of the segments in the boundary. Therefore, starting from the top left corner, by taking the entries to the left of the vertical segments and those above the horizontal segments of the boundary. If the GF element corresponding to an entry in the table

is the same as that of an entry previously inserted into the output vector, the new candidate entry will not be inserted, since the previous entry has smaller LLR.

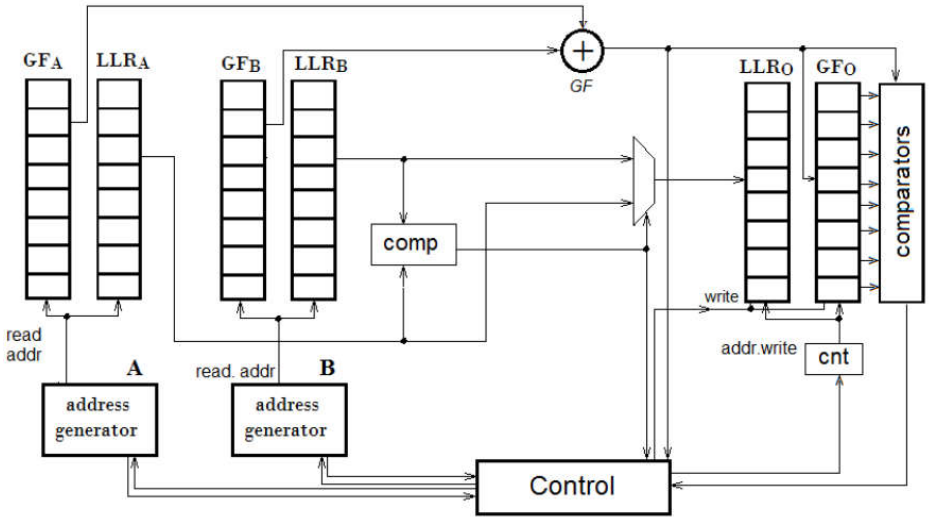

Figure 2.7. ECN architecture.

The previous algorithm can be implemented by the architecture illustrated in Figure 2.7. The comparisons of the LLRs start from the first entries in the two input vectors. In the case that LA(i) < LB(j) happens, LA(i) was the 'max' value when compared with any previous LB(k) with $0 \leq k < j$, since the LLRs in each vector are sorted. Hence, LA(i) should be inserted into the output vector together with GFA(i)+ GFB(k) for each $0 \leq k < j$. After this process is completed, i will increment so the next entry for the LA vector is read out in the next clock cycle to be compared with LB(j). Similarly, in the case that LA(i) > LB(j) , LB(j) will be inserted into the output vector with field element GFA(k) + GFB(j) for each $0 \leq k < i$.

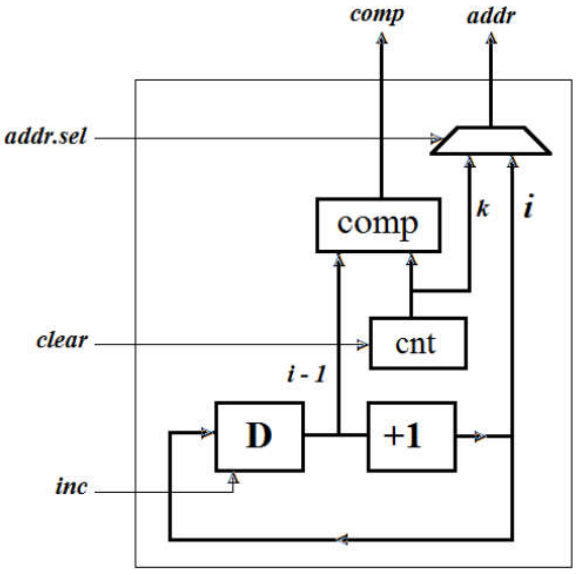

Figure 2.8. Address generator block architecture

It can be observed from the algorithm above that the addresses i and j either do not change or increase by one in each loop, in which the addressee generator blocks is the responsible of increasing them. If LA(i) < LB(j), the counter 'k' in the address generator B is cleared and used as the read address of the B vector. As the counter output increases, GFB(k) is read out

and added up with GFA(i) until the counter reaches j - 1. In the case that LA(i) > LB(j), the address generator A works in a similar way to generate the read address for the A vector. The write address n of the output vector increases by one each time a new LLR and corresponding finite field element needs to be inserted.

2.4. Variable Node architecture

In our project we have designed a VN with degree wc = 2, it means two extrinsic messages and one intrinsic message are the incoming messages of the VN. In our architecture all q messages are kept in each vector. The architecture of a VN is given in Figure 5.7.

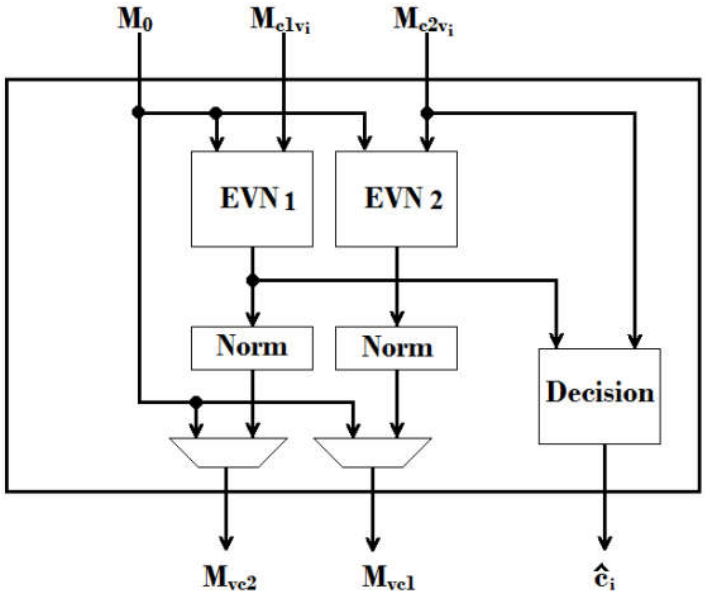

Figure 2.9. Variable Node architecture

It contains several Elementary Blocks that can make it capable to operates on three basic functions :

- Updating the VN: This operation is done by the EVN block, which receives two incoming message from the CNs and generate an new outgoing message to the normalization block .
- Normalization : The Norm block subtracts the smallest LLR in the input vector from each LLR in the vector so that the smallest LLR in each vector is brought back to zero.
- Decision: The Decision block, firstly calculates the APP by the sum of all VN inputs, then determine the estimated symbol c by taking the GF element corresponding to min LLR value.

2.4.1. EVN Architecture

The EVN computes stored outgoing message vector, by adding the LLRs of the two incoming message vectors corresponding to the same GF(q) element. Figure 5.8 illustrate the EVN architecture, two RAMs are used to storage the incoming message vectors, adder to add the LLR values, parallel GF elements comparators to help in finding the GF element address, sorter for the output vector and control unit to generate the control signals.

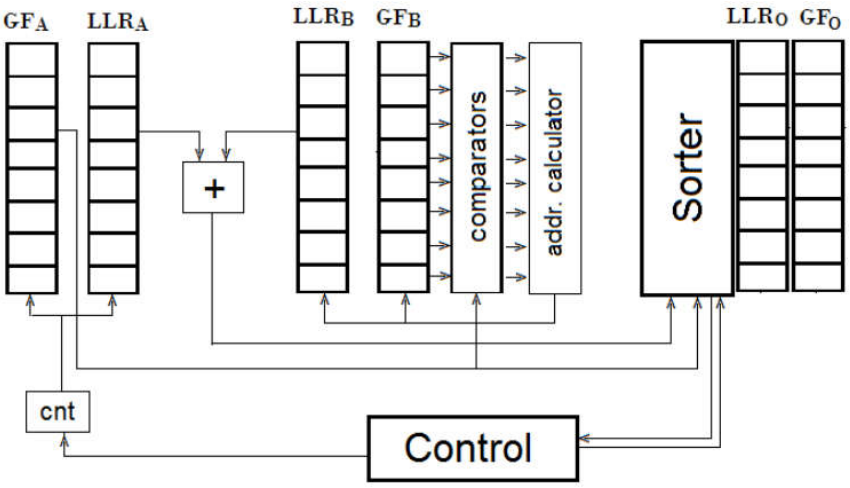

Figure 2.10. EVN architecture.

At first, one entry of the A vector is read out. Since the vectors are not sorted by GF element, the GF element of the GFA vector entry is compared with all those in the GFB vector. If there is a match, the addr calculator can give the corresponding GF element address, then using the addresses of A and B vectors to read theirs corresponding LLRs and perform the addition, the LLR addition result and its corresponding GF element are the entries of the output vector. The adder's output is connected to a sorter in order to sort and store the output elements. Using a counter to read out the next A vector entry and repeat these steps for all the q entries. All previous operations are controlled by the control unit.

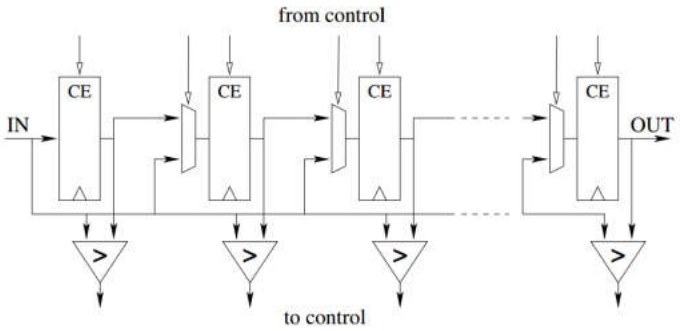

Figure 2.11. Sorter architecture .

An architecture for such a sorter was proposed in [15] is shown in Figure 2.11, This operator is used for inserting a new value into an already sorted list of values. The architecture of such operator with q registers and q parallel comparators in order to reduce the critical path of the processor to one single clock cycle.

2.4.2. Decision block

For the decision operation, we can use the output of EVN1 and Mc2v as inputs of the Decision block, which is a modified EVN block, that modification consist of taking as output the GF element corresponding to the min LLR value, which is the first element since the EVN integrate a sorter.

Conclusion

This project is dedicated to the architecture design of a low complex NB-LDPC decoder. we started with an understanding of the NB-LDPC codes concepts, and various variants of the error correcting decoding algorithms were investigated with an eye towards performance and feasible hardware design. Then we have chosen the Min-Max algorithm because of its low complexity in comparison with other algorithms. The complexity of the check node processing is further reduced in the Min-max algorithm with slightly lower coding gain.

It has been shown that there exist several NB-LDPC decoder architecture classes, and with the investigation on them we found that the row-parallel architecture class is highly scalable and supports the full range of throughput requirements found in modern wireless standards. Due to its favorable properties and spatially its flexibility i.e. be able to implement several given codes, without changing the design of the decoder.

In our project we have focused on the design of a row-parallel architecture for the NB-LDPC decoder and its basic blocks. Since modern communication systems decoders must support a wide range of different parity-check matrices, we provided a flexible decoder which can works with different block lengths and code rates.

In our project a particular attention was given to VN update, codeword decision and reduced complexity CN processing. We have designed the check node block using the forward backward technique to reduce the implementation complexity. In order to optimize the latency, we have used a practical algorithm to design the elementary check node block, that can work for both cases : all q messages are kept or only the nm << q most reliable elements. We have also designed the variable node block using elementary blocks for the same reason. From this work we can conclude about the implementation of Min-Max algorithms for NB-LDPC that the design of the VN is sampler than the design of the CN, because of that in our project we focus on the CN implementation which is considered as the bottleneck of the decoder.

Bibliography

[1] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inform. Theory*, vol. 8, no. 1, pp. 21-28, Jan. 1962.

[2] Christoph Roth. Vlsi design, optimization, and implementation of channel decoding in wireless systems. 2015.

[3] Fang Cai. *Low-complexity Decoding Algorithms and Architectures for Non-binary LDPC Codes*. PhD thesis, Case Western Reserve University, 2013.

[4] D. J. C. MacKay, "Good Error-Correcting Codes Based on Very Sparse Matrices," *IEEE Trans. Inform. Theory*, vol. 45, no. 2, Mar. 1999.

[5] Prof. Dr. Ing. Ulrich Reimers (auth.). *Digital Video Broadcasting (DVB): The International Standard for Digital Television*. Springer Berlin Heidelberg, 2001.

[6] Sarah J Johnson. *Iterative error correction: Turbo, low-density parity-check and repeat accumulate codes*. Cambridge University Press, 2009.

[7] Volker Kuhn Andre Neubauer, Jurgen Freudenberger. *Coding Theory - Algorithms, Architectures, and Applications*. Wiley Inter science, 2007.

[8] E. J. Weldon W. Wesley Peterson. *Error-Correcting Codes - Revised, 2nd Edition*. second edition, 1972.

[9] R. Tanner. A recursive approach to low complexity codes. *IEEE Transactions on Information Theory*, Sep 1981.

[10] Daniel J. Costello Shu Lin. *Error Control Coding: Fundamentals and Applications*. Prentice-Hall Computer Applications in Electrical Engineering Series. Prentice Hall, first edition, 1983.

[11] C. Qian, W. Lei, and Z. Wang. Low complexity LDPC decoder with modified sum-product algorithm. *Tsinghua Science and Technology*, Feb 2013.

[12] Li Zhang, Qin Huang, Shu Lin, Khaled Abdel-Ghaffar, and Ian F Blake. Quasi-cyclic LDPC codes: an algebraic construction, rank analysis, and codes on Latin squares. *IEEE transactions on communications*, 2010.

[13] Shu Lin, Shu mei Song, Bo Zhou, Jing yu Kang, Ying Y Tai, and Qin Huang. Algebraic constructions of non binary quasi-cyclic LDPC codes: Array masking and dispersion. In *9th International Symposium on Communication Theory and Applications (ISCTA)*, 2007.

[14] M. M. Mansour and N. R. Shanbhag. A 640-mb/s 2048-bit programmable LDPC decoder chip. *IEEE Journal of Solid-State Circuits*, March 2006.

[15] A. Voicila, D. Declercq, F. Verdier, M. Fossorier, and P. Urard. Architecture of a low complexity non-binary LDPC decoder. In *2008 Digest of Technical Papers - International Conference on Consumer Electronics*, Jan 2008.

[16] M.M. Mansour and N.R. Shanbhag. Low-power VLSI decoder architectures for LDPC codes. In *Low Power Electronics and Design, 2002. ISLPED '02. Proceedings of the 2002 International Symposium on*, pages 284–289, 2002.

[17] M.M. Mansour and N.R. Shanbhag. High-throughput LDPC decoders. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 2003.

[18] Hubert Kaeslin. *Digital Integrated Circuit Design: From VLSI Architectures to CMOS Fabrication*. Cambridge University Press, 1st edition, 2008.