DEMOCRATIC AND POPULAR REPUBLIC OF ALGERIA

MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

Ecole Nationale Polytechnique

Electronic Departement
Laboratoire des Dispositifs de Communication et de Conversion Photovoltaique

Submitted in partial fulfillment of the requirements for the Engineer Degree

Implementation of Non Binary LDPC Decoder on FPGA for Wireless Communication Systems

BALI Cherif HARABI Kamel-eddine

Supervised by : Mr. M.TAGHI

Publicly presented on : 18/06/2017

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 Departement
Laboratoire des Dispositifs de Communication et de Conversion Photovoltaique

Submitted in partial fulfillment of the requirements for the Engineer Degree

Implementation of Non Binary LDPC Decoder on FPGA for Wireless Communication Systems

BALI Cherif   HARABI Kamel-eddine

Supervised by :   Mr. M.TAGHI

Publicly presented on :   18/06/2017

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) قد تم إدراجها بنجاح في العديد من أنظمة الاتصالات اللاسلكية لأنها تحقق أداء تصحيح الخطأ قريبة جدا من حد شانون (Shannon). شفرات اختبار التكافؤ منخفضة الكثافة غير الثنائية(NB-LDPC) لديها أداء أفضل من الفرات الثنائية، في هذه الأطروحة ركزنا على تصميم وإنشاء بنية فعالة للأجزاء الأساسية من مفكك شفرات اختبار التكافؤ منخفضة الكثافة غير الثنائية باستخدام خوارزمية الحد الأدنى-الحد الأكبر (Min-Max). من أجل توفير مفكك شفرات مرن. قد تم تفصيل تصميم وتنفيذ مكونات مفكك التشفير. و توثيق تفاصيل مختلفة مثل مخططات أجزاء المفكك والمحاكاة.

الكلمات المفتاحية : شفرات اختبار التكافؤ منخفضة الكثافة (LDPC), شفرات اختبار التكافؤ منخفضة الكثافة غير الثنائية (NB-LDPC), حد شانون, تصحيح الخطأ, خوارزمية Min-Max, مفكك الشفرات, بنية, تصميم, إنشاء.

Résumé

Les codes de contrôle de parité de faible densité (LDPC) ont été inclus avec succès dans de nombreuses standard de communication sans fil, car ils atteignent des performances de correction d'erreur très proches de la limite de Shannon. Les codes LDPC non binaires ont de meilleures performances que les codes LDPC binaires. Dans cette thèse, nous sommes concentrés sur la conception et l'implémentation d'une architecture efficace des blocs de base du décodeur NB-LDPC à l'aide de l'algorithme Min-Max. Afin de fournir un décodeur flexible. La conception et l'implémentation des composants du décodeur sont détaillées. Divers détails comme les schémas de blocs et la simulation ont été documentés.

Mots clés: LDPC, NB-LDPC, Limite de Shannon, Correction d'erreur, Min-Max, Décodeur, Architecture, Conception, Implémentation.

Abstract

Low Density Parity-Check (LDPC) codes have been successfully included in numerous wireless communication standards, since they achieve error correction performance very close to the Shannon limit. Non-Binary LDPC codes has better performance than the binary LDPC codes, In this thesis, we focused on the design and implementation of efficient architecture of the NB-LDPC decoder basic blocks using the Min-Max algorithm. In order to provide flexible decoder. The design and implementation of the decoder components are detailed. Various details like block schematics and simulation have been documented.

Keywords : LDPC, NB-LDPC, Shannon limit, error correction, 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

We 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.

We 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

Abreviations

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

The reliable transmission of information over noisy channels is one of the basic requirements of wireless communication systems. Since these systems demand for high-speed information exchange between transmitter and receiver nodes, the channel impairments become more harmful, which reduce the reliability of the received information. To overcome this situation and provide more reliable communications, efficient channel coding techniques are required. Due to this requirement, these systems rely heavily on error correction codes to detect and correct transmission errors.

We distinguish two main families of error correction codes according to the way redundancy is added: block codes and convolutional codes [4]. 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 and implement a NB-LDPC decoder based on Min-Max decoding algorithm for wireless communication systems. 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 wireless communication standards.

Chapter 1

Chapter 1: General Digital Communication Systems

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

This chapter begins by illustrating the chain of communication and briefly explaining the role and interest of each block of the chain, before introducing the different types of error-correcting codes, and end up with a comparison of their performance.

1.1 Introduction

The reliable transmission of information over noisy channels is one of the basic requirements of digital information and communication systems. Here, transmission is understood both as transmission in space, e.g. over mobile radio channels, and as transmission in time by storing information in appropriate storage media. Because of this requirement, modern communication systems rely heavily on powerful channel coding methodologies. For practical applications these coding schemes do not only need to have good coding characteristics with respect to the capability of detecting or correcting errors introduced on the channel. They also have to be efficiently implementable, e.g. in digital hardware within integrated circuits.

Practical applications of channel codes include space and satellite communications, data transmission, digital audio and video broadcasting and mobile communications, as well as storage systems such as computer memories or the compact disc[7].

1.2 Wireless Digital Communication System

In Figure 1.1 the basic structure of a digital communication system is shown which represents the architecture of the communication systems in use today.

Within the transmitter of such a communication system the following tasks are carried out:

- Source Coding
- Channel Coding
- Modulation

- Communication Channel
- Channel Decoder

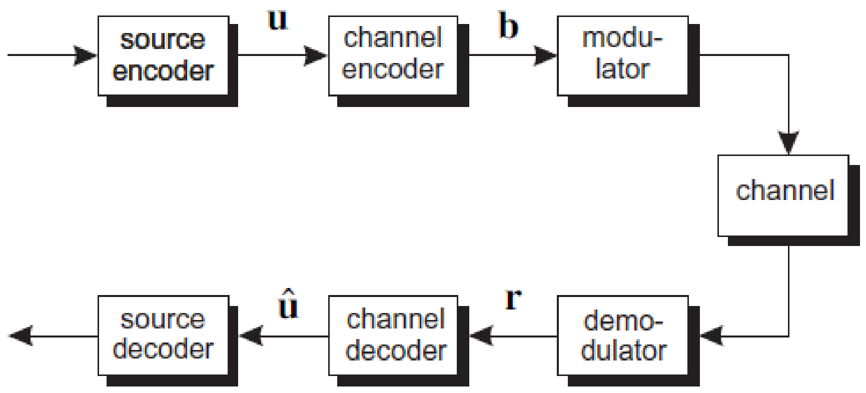


Figure 1.1: Communication System Chain

1.2.1 Source coding

Source codin g, data compression or bit-rate reduction involves encoding information using fewer bits than the original representation.Compression can be either lossy or lossless. Lossless compression reduces bits by identifying and eliminating statistical redundancy. No information is lost in lossless compression. Lossy compression reduces bits by removing unnecessary or less important information.The process of reducing the size of a data file is referred to as data compression. In the context of data transmission, it is called source coding (encoding done at the source of the data before it is stored or transmitted) in opposition to channel coding.

1.2.2 Channel coding

The purpose of channel coding is to protect the message from disturbances of the channel, by introducing a redundancy to the useful information in the message, redundancy and useful information are related by a given law.

At the Receiver, the channel decoder exploits the redundancy produced by the encoder for the purpose to detect and then to correct if possible the errors introduced during the transmission ,[8]. This point will be detailed more in the following sections.

1.2.3 Communication channel

The communication channel is the physical medium for routing a message between a source and one or more recipients. There are several types of channels, but in theory of information, the most used channels are called discrete channels A discrete channel is a stochastic system accepting at the input of the sequences of symbols defined on an alphabet X, and outputting sequences of symbols defined on an output alphabet Y, connected by a transition law $P_{y|x}$ i.e a stochastic matrix $M_{y|x}$.

$$M_{y|x} = \begin{bmatrix} P_{y_1|x_1} & \cdots & P_{y_j|x_1} \\ \vdots & \ddots & \vdots \\ P_{y_1|x_k} & \cdots & P_{y_j|x_k} \end{bmatrix}$$

1.2.4 Modulation

The modulation consists in performing a coding in the Euclidean space, a space generally adapted to the channels encountered in practice. For an M-ary modulation, each signal of L bits is associated with a signal $x_i(t), i = 1, ..., M$ of duration T chosen from the $M = 2^L$ signals[9].There are three major classes of digital modulation techniques used for transmission of digitally represented data:

- Amplitude-shift keying (ASK)
- Frequency-shift keying (FSK)
- Phase-shift keying (PSK)

Example of PSK is the Binary-Phase-shift keying is explained in the figure 1.2.

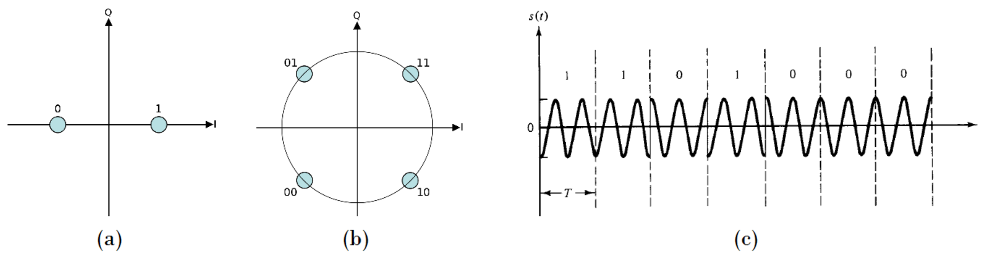


Figure 1.2: The BPSK Modulation (a) , its wave form (c) and the QPSK Modulation (b)

As for demodulation, its role is to extract the samples and to decide In favor of the symbols most probably emitted [9]. The data provided by the demodulation unit will be processed by the so-called decoder. There are two types of decoders, the first one being called a hard decision decoder, because it works on firm data ('0' or '1'). The second type is called a soft decision decoder, because the demodulator provides the decoder with a firm value accompanied by a reliability measure. The Figure 1.2 illustrate the BPSK example of digital modulation.

1.3 Information Theory

An important result of information theory is the finding that error-free transmission across a noisy channel is theoretically possible – as long as the information rate does not exceed the so-called channel capacity.

In order to quantify this result, we need to measure information.Within Shannon's information theory this is done by considering the statistics of symbols emitted by information sources[10].

1.3.1 Entropy

Let us consider the discrete memoryless information source shown (in Figure 1.3). At a given time instant, this discrete information source emits the random discrete symbol $X = x_i$ which assumes one out of M possible symbol values $x_1, x_2, ..., x_M$. The rate at which these symbol values appear are given by the probabilities $P_X(x_1), P_X(x_2), ..., P_X(x_M)$ with $P_X(x_i) = Pr\{X = x_i\}$.

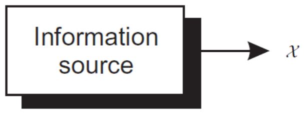


Figure 1.3: Discrete Information Source

The average information associated with the random discrete symbol X is given by the so-called entropy measured in the unit 'bit'

$$I(X) = -\sum_{i=1}^{M} P_X(x_i) \cdot log_2(P_X(x_i)) \tag{1.1}$$

1.3.2 Channel Capacity

With the help of the entropy concept we can model a channel according to Berger's channel diagram shown in Figure 1.4 (Neubauer, 2006a). Here, X refers to the input symbol and R denotes the output symbol or received symbol. We now assume that M input symbol values $x_1, x_2, ..., x_M$ and N output symbol values $r_1, r_2, ..., r_N$ are possible. With the help of the conditional probabilities

$$P_{X|R}(x_i|r_j) = Pr\{X = x_i|R = r_j\} \tag{1.2}$$

and

$$P_{R|X}(r_j|x_i) = Pr\{R = r_j|X = X_i\} \tag{1.3}$$

the conditional entropies are given by

$$I(X|R) = -\sum_{i=1}^{M}\sum_{j=1}^{N} P_{X,R}(x_i, r_j) \cdot log_2(P_{R|X}(x_j|r_j)) \tag{1.4}$$

$$I(R|X) = -\sum_{i=1}^{M}\sum_{j=1}^{N} P_{X,R}(x_i, r_j) \cdot log_2(P_{R|X}(r_j|x_i)) \tag{1.5}$$

With these conditional probabilities the mutual information

$$I(X; R) = I(X) - I(X|R) = I(R) - I(R|X) \tag{1.6}$$

can be derived which measures the amount of information that is transmitted across the channel from the input to the output for a given information source. The so-called channel capacity C is obtained by maximising the mutual information $I(X; R)$ with respect to the statistical properties of the input X, i.e. by appropriately choosing the probabilities $\{P_X(x_i)\}_{1\leq i\leq M}$. This leads to

$$C = \max_{\substack{\{P_X(x_i)\}\\ 1\leq i\leq M}} I(X;R) \tag{1.7}$$

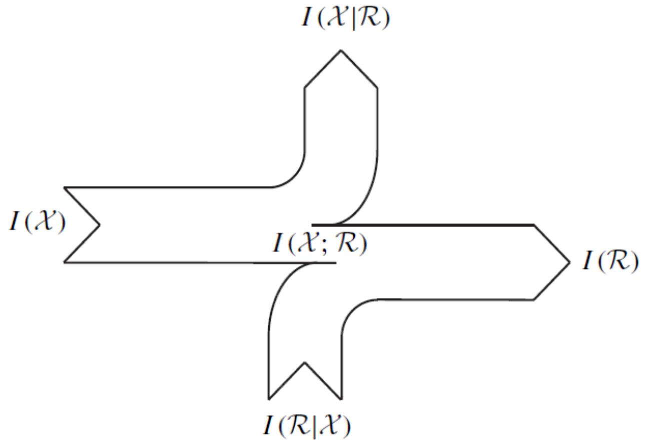

$I(\mathcal{X}|\mathcal{R})$
$I(\mathcal{X})$
$I(\mathcal{X};\mathcal{R})$
$I(\mathcal{R})$
$I(\mathcal{R}|\mathcal{X})$

Figure 1.4: Berger's channel diagram

1.3.3 Coding Theory

If the input entropy $I(X)$ is smaller than the channel capacity C,

$$I(X) \overset{!}{<} C$$

then information can be transmitted across the noisy channel with arbitrarily small error probability. Thus, the channel capacity C in fact quantifies the information transmission capacity of the channel.

1.3.4 Definitions

- The channel encoder is used to generate a code word c of N bits from a Word of information x of K bits. This code therefore generates M redundancy bits, with $M = N - K$, called parity bits, which we shall denote by the vector p.
- A code is said to be systematic if the symbols of x appear explicitly in c. The ratio of the number of bits Information and the number of bits of the transmitted codeword: $R = K/N$
- The symbols of the information message x and the code word c take their values In a finite field F_q with q elements, called Galois field ($GF(q)$) and those principal Properties are illustrated in reference [16]. For example for a binary code , symbols take their value in the body $F_2(GF(2))$ with two elements $\{1, 0\}$.
  The elementary operations of addition and multiplication in the field F_2 are given in the table 1.1.

a	b	$a \oplus b$	$a \odot b$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Table 1.1: Addition and multiplication on $GF(2)$

1.3.5 Error Correcting Performance Measuring

The digital communication systems use BER (Bit Error Rate) as a function of the SNR to evaluate codes Performances , but to obtain more accurate results, especially when comparing different types codes, another measure is used which gives the BER as a function of Eb / No. With:
Eb: The energy transmitted in one bit of information.
No: The spectral density of the noise.

$$BER = \frac{Number\ of\ erroneous\ bits}{Number\ of\ transmited\ bits} \tag{1.8}$$

1.3.6 Error Correcting Codes Classification

A code is called linear if the coding function is a linear mapping, otherwise it is called non-linear. When the treatments are required to obtain the detection properties or correction are made per block of N symbols, we say that we are dealing with a code in block. When the symbols generated by the source are not processed by the blocks, Continuously, we say that we are dealing with a convolutional code. For the rest of this chapter, We will only talk about block codes and convolutional codes.

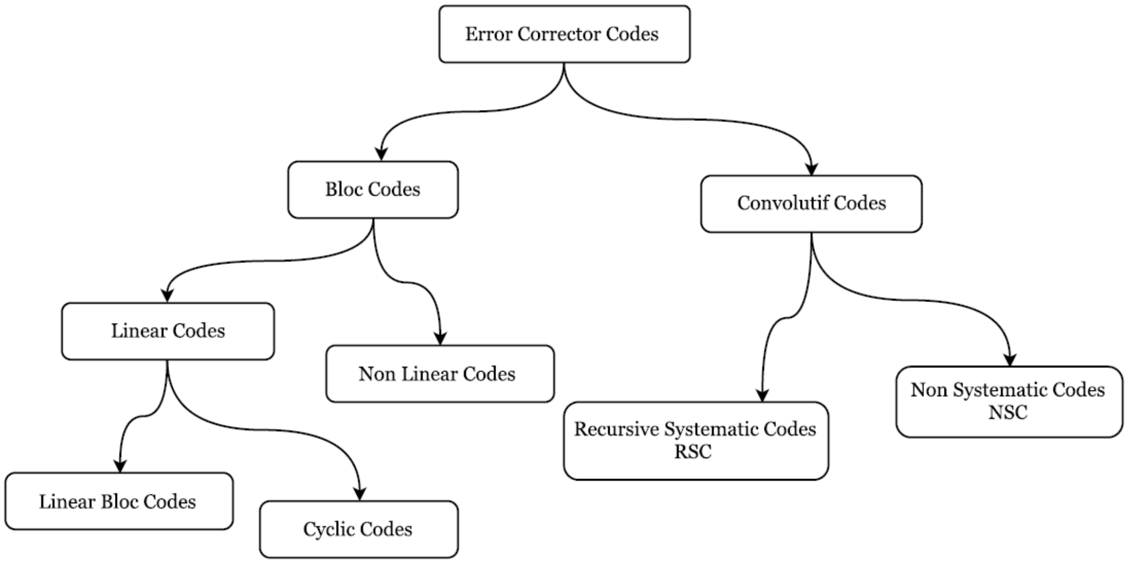


Figure 1.5: Error Correcting Codes Classification

1.4 Error Correcting Code Types

1.4.1 Bloc Codes

The purpose of the block encoding operation is to associate with each information word composed of K q-aire symbols a codeword composed of N q-aire symbols, this operation can be represented by an application g

$$\begin{gathered} g : F_K^q \to F_N^q \\ x \to c = g(x) \end{gathered} \tag{1.9}$$

According to the classification diagram given in Figure 1.5, Linear block codes divide in two main types:
Linear Bloc Codes: Are those where the code words are considered as Being elements in a vector space.
Cyclic Codes: Are those in which code words are considered to be Elements in an algebra, namely polynomials [11].

Linear Bloc Codes

The linear block codes are characterized by a matrix G of size (K, N) called the generating matrix. This matrix transforms an information message x of K bits into a code word c of N bits $(N > K)$ by the following matrix operation:

$$c = x.G \tag{1.10}$$

with:

$$G = \begin{bmatrix} g_{11} & g_{12} & \dots & g_{1N} \\ g_{21} & g_{22} & \dots & g_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k1} & g_{k2} & \dots & g_{kN} \end{bmatrix} \tag{1.11}$$

Each code word is a linear combination of the vectors G_i of G. Thus, one linear block code can be defined as a vector subspace with $K < N$ dimensions constructed in accordance with relation 1.9 . To facilitate the coding operation, it is always possible to put the matrix G in the systematic form, by combining the lines between them.

$$G_{syst} = [P^t | I_k] \tag{1.12}$$

$$I_k = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix} \tag{1.13}$$

and

$$P^t = \begin{bmatrix} r_{11} & r_{12} & \dots & r_{1N} \\ r_{21} & r_{22} & \dots & r_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ r_{k1} & r_{k2} & \dots & r_{kN} \end{bmatrix} \tag{1.14}$$

The linear block codes are also characterized by another matrix H of size (N, M) called the parity check matrix [9]. The main property of this matrix is:

$$H.c^t = 0 \tag{1.15}$$

$$H.G^t = 0 \tag{1.16}$$

and for a systematic code

$$H_{syst} = [I_M|P] \tag{1.17}$$

Once the encoding operation is complete, the message c will be transmitted through a channel which is generally noisy, a noise n is added to the latter. At the reception, the message will be given by the following relation:

$$r = c + n = x.G + n \tag{1.18}$$

by using 1.15 and 1.18 we get :

$$H.r^t = H.c^t + H.n^t = H.n^t \tag{1.19}$$

The product $H.r^t$ is called a syndrome, if the result of this product is a zero vector , then r is a code word, otherwise the vector r contains erroneous bits. The calculation of syndrome is the method used by most of the block codes, for detect the error presence and then, depending on the decoding algorithm, correct these errors if possible [11].

Bloc Codes examples

The first block codes are the Hamming codes introduced in 1950 by Richard Hamming [12]. These codes gave poor results compared to the criteria of Varshamov and Gilbert [4], which is why new error-correcting codes have been developed, for example: Reed-Solomon codes which are Classes of BCH cyclic codes. These codes, developed by IS Reed and G. Solomon [13], are widely used for the correction of group errors in most digital data carriers such as CD, DVD, blu-ray Discs, and in many standards such as DVB-T [5]. There are many other classes of codes in block, which are not going to be detailed in this manuscript as: Goppa codes which are widely used in McEliece and Niederreiter crypto-systems, Reed-Muller codes, the Golay codes ... etc. The most powerful block code to date is called LDPC (Low Density Parity Check). This family represents the object of our study, which will be presented in Chapter 2.

1.4.2 Convolutional Codes

The convolutive codes, invented in 1954 by Peter Elias [14], constitute a family of error-correcting codes, whose simplicity of coding and decoding are at the origin of their success. The principle is no longer to cut the message into finite blocks, but to consider it as a semi-infinite sequence $a_0 a_1 a_2 ... a_n$ of symbols which passes through a succession of shift registers whose number of stages m is called code memory and 2^m the number of possible states. The quantity $\mu = m+1$ is called the constraint length of the code and the ratio $R = K/N$ is called the coding efficiency. to illustrate the principle of convolutional codes, here is an example

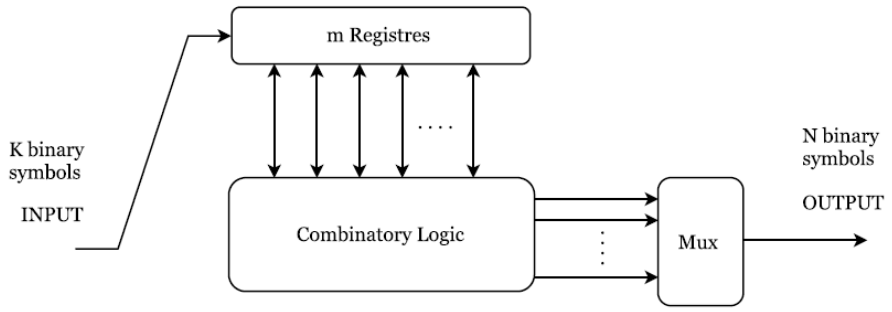


Figure 1.6: The Principle of convolutional code with coding efficiency R and m memory registers

presented in Figure 1.7, for $K = 1, m = 2$ and $N = 2$. At the instant t, the output bits X and Y are calculated by The following relationships:

$$\begin{cases} X = a_t + a_{t-1} + a_{t-2} \\ Y = a_t + a_{t-2} \end{cases}$$

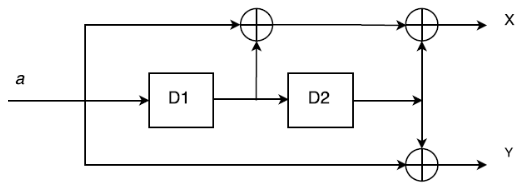


Figure 1.7: Convolutional Code example $R = 1/2$

NSC and RSC Codes

NSC refers to non-systematic codes and by RSC recursive and systematic codes

A convolutional code is a recursive code if the sequence passing through the shift registers is fed by the contents of its registers (see Figure 1.8(b)). If the K information symbols at the encoder input are explicitly found in the code, then the code is called systematic, otherwise it is called non-systematic (see Figure 1.8(a)). The non-systematic and non-recursive codes methods have high performances better than a systematic and non-recursive code and the reverse for weak SNRs. For this reason, the NSC codes were mainly studied and used up to early 1990s. It is recalled that the power of a code (correction capacity Of errors) and the complexity of the decoder increase with increasing memory M of this code. As for systematic recursive codes (RSC), they are used by Turbo-codes, as they are the only ones likely to reach the limit of Shannon[15].

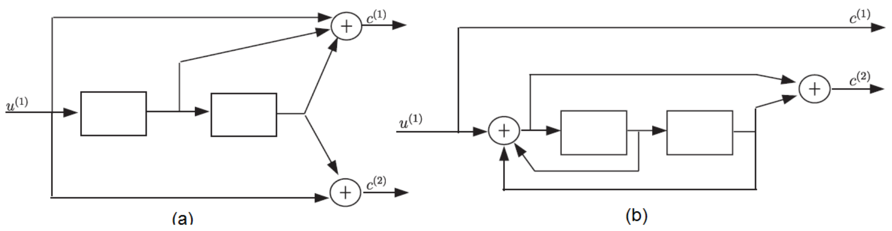


Figure 1.8: (a) Non systematic encoder,(b) Recursif systematic encoder

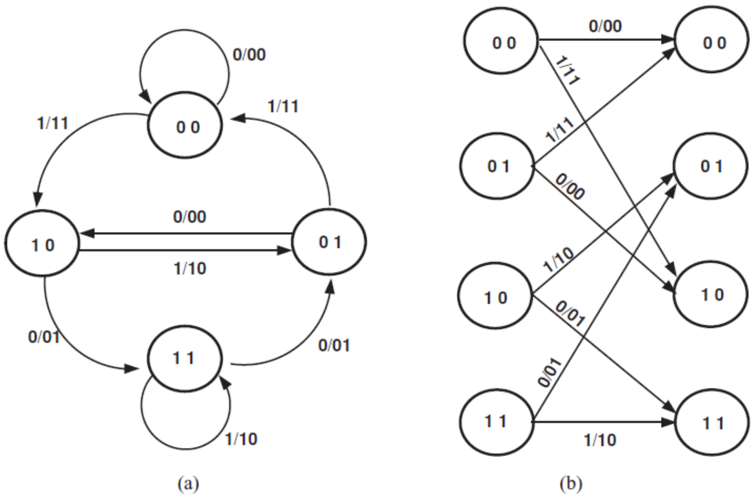


Figure 1.9: (a) The state transition diagram and (b) a segment of the trellis diagram for the Rate 1/2 systematic recursive binary convolutional encoder in Figure 1.8

Turbo Codes

The most famous of the convolutive codes is undoubtedly the turbo-code7 invented by C.Berro, A.Glavieux and P. Thitimajshima in 1993 [10]. These codes and LDPC codes form So-called advanced coding techniques. The turbo-code uses two (or multiple) convolutional encoders. Figure 1.10 shows the case of a concatenation Parallel, consisting of two identical convolution codes and one pseudo-random interleaver.

Each information word x is associated with a redundancy p, which can be divided into a redundancy p_0 resulting from the first encoder and a redundancy p_1 from the second encoder. For the first time, an iterative decoder is introduced. The idea, very simple in Itself, consists of a decoder comprising two subsets of decoding exchanging informations.

To explain the functioning of such a decoder, the notion of extrinsic information was introduced. This information that is exchanged between the decoders during the iterations. After a certain number of iterations, the firm decision is taken on a posteriori information. This information gathers both the information coming from the observation of the channel and the extrinsic information from the different decoders [16].

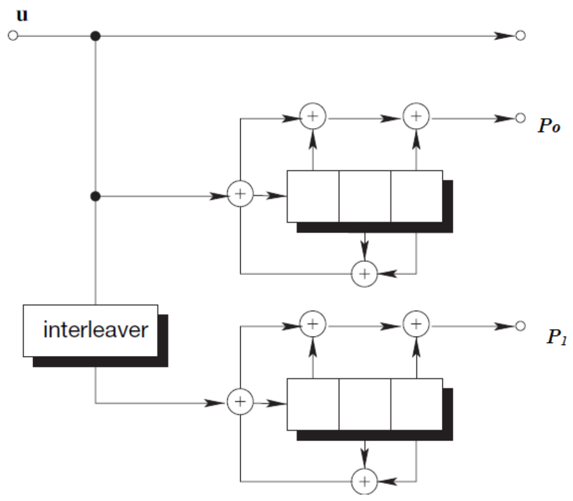


Figure 1.10: Turbo Encoder example

Code Performance Comparison

Low density parity check (LDPC) codes are a powerful FEC coding scheme that can achieve good error performance under very low signal-to-noise ratios. A communication system utilizing an LDPC code is able to operate very close to the channel capacity limit established by Claude Shannon in the 1940's. Figure 1.11 compares the performance of various coding schemes used in the communication industry. It shows that LDPC codes achieve the same code rate of 0.5 as many other codes. It can operate close to the Shannon Capacity Bound in a lower signal power environment.

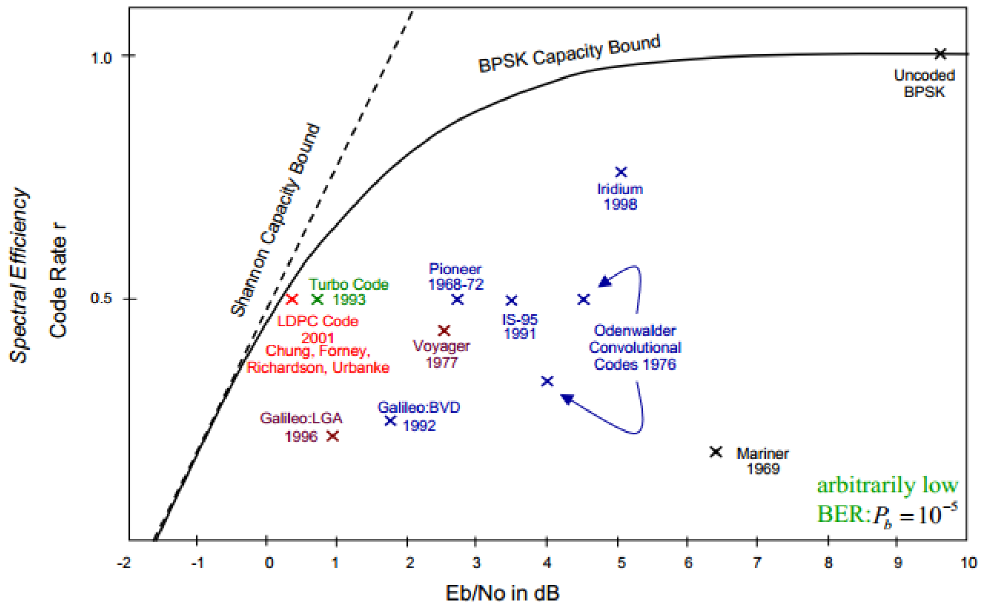


Figure 1.11: FEC Code Performance Comparison

1.5 Channel codes applications in different wireless standards

With the increasing capabilities of integrated circuits, both LDPC codes and turbo codes have gradually been considered in various practical applications. Figure 1.12 summarizes the proliferation of LDPC and turbo codes in practical communication systems with a focus on prominent recent and emerging wireless standards. Since the use of turbo codes was specified for UMTS in 1999, turbo codes have become the dominating choice for forward error-correction in enhanced-2G, 3G, and emerging 4G cellular standards. The practical consideration of LDPC codes lagged somewhat behind the application of turbo codes. After the rediscovery of LDPC codes, researchers focused on finding efficient structured codes that would simplify the encoding and decoding process while maintaining excellent performance. As a result of these research efforts, several interesting classes of structured codes emerged, and the particular class of quasi-cyclic (QC) LDPC codes has received special attention in the context of wireless systems. Structured LDPC codes appeared the first time in 2003 in the DVB-S2 satellite broadcasting standard for digital television and since then have been considered in many other advanced wireless systems, such as IEEE 802.16e WiMAX or IEEE 802.11n WLAN.

As implied in Figure 1.12, LDPC codes and turbo codes typically complement classical channel codes. Especially the well-established convolutional codes are still commonly found also in emerging wireless standards for low-rate services such as voice and control signaling, or as fallback solution for receiver terminals not supporting LDPC codes or turbo codes.

	Peak data rate [Mbps]	LDPC codes	Turbo codes	Classical codes
3GPP cellular standards				
GSM (2G)	0.010	×	×	√
GPRS (2.5G)	0.086	×	×	√
EDGE (2.5G)	0.237	×	×	√
E-EDGE (2.75G)	1.3	×	√	√
UMTS (3G)	0.384	×	√	√
HSPA (3.5G)	14.4	×	√	√
HSPA+ (3.75G)	42[a]	×	√	√
LTE (3.9G)	300[b]	×	√	√
LTE-A (4G)	3000[c]	×	√	√
WLAN, WMAN, and broadcasting standards				
802.11n WLAN	600	√	×	√
802.11ac Gb-WLAN	1730[d]	√	×	√
802.11ad WiGig	6760	√	×	√
802.16e WiMAX R1	75	√	√	√
802.16 WiMAX R1.5	144	√	√	√
DVB-S2	135	√	×	×

[a]2x2 MIMO, R8.
[b]20 MHz bandwidth, 4x4 MIMO, R8.
[c]100 MHz bandwidth, 8x8 MIMO, R10.
[d]80 MHz bandwidth, 4x4 MIMO.

Figure 1.12: Deployment of LDPC and Turbo codes in prominent wireless standards[1]

Chapter 2

Chapter 2: LDPC Code

Low-density parity-check (LDPC) codes are a class of linear block codes. The name comes from the characteristic of their parity-check matrix which contains only a few non-zero elements in comparison to the amount of zeros, They 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. As researchers struggled through the 1990s to understand just why turbo codes worked as well as they did, two researchers, McKay and Neal, introduced a new class of block codes designed to posses many of the new turbo codes features. It was soon recognized that these block codes were in fact a rediscovery of the LDPC codes developed years earlier by Gallager. Today, design techniques for LDPC codes exist which enable the construction of codes which approach the Shannon's capacity limit.

This chapter give a brief presentation of LDPC codes, including the Fundamentals and mathematical basics. we will introduce the deferent types of representation for these codes, their proprieties and classes. We will also present some LDPC construction methods and finally explain the coding and decoding operations and give some basic decoding algorithm .

2.1 Fundamentals of 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. This sparseness of H is essential for an iterative decoding complexity that increases only linearly with the code length. These codes are built from the simplest elementary code: the single parity check code.

2.1.1 Parity check code

Parity checking is the most basic form of error detection in communications. The simplest coding scheme is the single parity-check code. This code involves the addition of a single extra bit, called a parity-check bit, to the binary message, the value of this bit depends on the bits in the message. In an even-parity code the additional bit added to each message ensures an even number of 1s in every codeword.

Example (1) : Denote a code $\mathbf{C}$ consists of codeword of length $n = 6$, and the vectors $c =$

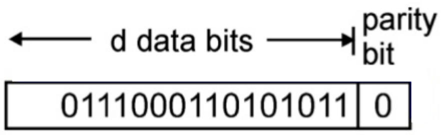


Figure 2.1: Parity check code

$[c_1\ c_2\ c_3\ c_4\ c_5\ c_6\ ]$, where each c_i is either 0 or 1 and every codeword satisfies the constraint:

$$c_1 \oplus c_2 \oplus c_3 \oplus c_4 \oplus c_5 \oplus c_6 = 0 \oplus 0 \oplus 1 \oplus 0 \oplus 0 \oplus 1 = 0 \tag{2.1}$$

Equation (2.1) is called a parity-check equation. which is an equation linking n binary data to each other by the exclusive or, denoted $\oplus$ operator. It is satisfied if the total number of 1s in the equation is even or null.

While the inversion of a single bit due to channel noise can easily be detected with a single-parity check code, this code is not sufficiently powerful to indicate which bit, or perhaps bits, were inverted. Moreover, since any even number of bit inversions produces a vector satisfying the constraint (2.1), patterns of even numbers of errors go undetected by this simple code.

Detecting more than a single bit error calls for increased redundancy in the form of additional parity bits. These more sophisticated codes contain multiple parity-check equations, every one of which must be satisfied by every codeword in the code.

Example (2): Denote the vector $c = [c_1\ c_2\ c_3\ c_4\ c_5\ c_6]$ that satisfy the three parity-check equations:

$$\begin{aligned} c_1 \oplus c_2 \oplus c_4 &= 0 \\ c_1 \oplus c_2 \oplus c_3 \oplus c_6 &= 0 \\ c_2 \oplus c_3 \oplus c_5 &= 0 \end{aligned} \tag{2.2}$$

Checking the vector $\hat{c} = [1\ 1\ 0\ 0\ 0\ 0]$ we see that :

$$\begin{aligned} 1 \oplus 1 \oplus 0 &= 0 \\ 1 \oplus 1 \oplus 0 \oplus 0 &= 0 \\ 1 \oplus 0 \oplus 0 &= 1 \end{aligned} \tag{2.3}$$

so $\hat{c}$ is not a valid codeword for this code because the parity-check equations did not satisfy . The Codeword constraints are making an equations system in order to simplify the working in these constraints are often written in matrix form, and so the constraints (2.2) become :

$$\underbrace{\begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}}_{H} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.4}$$

The matrix H is called a parity-check matrix. Each row of H corresponds to a parity-check equation and each column of H corresponds to a bit in the codeword. The $(j, i)th$

entry of H is 1 if the ith codeword bit is included in the jth parity-check equation. Thus for a binary code with m parity-check constraints and length-n codewords the parity-check matrix is an $m \times n$ binary matrix.

In matrix form a vector $\hat{c} = [c_1\ c_2\ c_3\ c_4\ c_5\ c_6]$ is a valid codeword for the code with parity-check matrix H if and only if it satisfies the matrix equation :

$$H\hat{c}^T = 0_m \tag{2.5}$$

More than one parity-check matrix can describe a particular code; two parity check matrices for the same code do not even have to have the same number of rows, but they must satisfy (2.3) for every codeword in the code.

In Examples 2 there are three linearly independent equations in six variables, so there are three dependent variables (one for each equation). These are the variables corresponding to the parity-check bits. The remaining independent variables correspond to the message bits. So, for the code in Example 2, the number of codeword bits $n = 6$, the number of message bits $k = 3$ and so we have a rate $R = 1/2$ code.

In general, a code can have any number of parity-check constraints but only $n-k$ of them will be linearly independent, where k is the number of code message bits. In matrix notation $n - k$ is the rank of H over $GF(2)$.

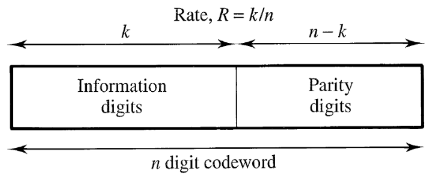


Figure 2.2: Discreption of parity check codeword

2.1.2 Representation of LDPC Codes

We assume that the output of an information source is a sequence of binary digits “0” or “1” in block coding as the LDPC codes, this binary information sequence is segmented into message blocks of fixed length; each message block, denoted by x, consists of k information digits. There are a total of 2^k distinct messages. The encoder, according to certain rules, transforms each input message x into a binary $n-tuple$ c with $n > k$. This binary $n-tuple$ c is referred to as the code word of the message x .

There are two ways to represent the LDPC code. As a linear block code, it can be described via matrices. The second way is via a graphical description.

A. Matrix Representation

The LDPC code can be described by two basic matrix the generator matrix G of dimension $k \times n$ and the parity check matrix H of dimension $m \times n$, 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 generator matrix G: This matrix describes the mapping from source words x to codewords c in the encoding part by the equation $c = G^T x$.

It is common to consider G in systematic form $G = [I_k|P]$ so that the first k transmitted symbols are the source symbols.The notation $[A|B]$ indicates the concatenation of matrix A with matrix B; I_k represents the $k \times k$ identity matrix. The remaining symbols are the parity-checks.

$$\begin{aligned} G = [I_3|P] &= \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \\ H = [P^T|I_3] &= \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{2.6}$$

Parity Check Matrix : The LDPC code is also described by a parity check matrix H of dimension $m \times n$, This matrix can be seen as a linear system of m parity check equations. The words c of the code defined by H simultaneously satisfy the m parity check equations. If the corresponding generator matrix is written in systematic form as above, then H has the form $[-P^T|I_m]$. Note that for codes over finite fields $GF(2^p)$, $-P \equiv P$.

Each row of the parity-check matrix describes a linear constraint satisfied by all code words HG^T and hence the parity-check matrix can be used to detect errors in the received vector:

$$H.r = H(c+e) = H.G^T.x + H.e = H.e = s \tag{2.7}$$

where e is the error vector and s is the syndrome vector. If the syndrome vector is null, we assume that there has been no error.

B. Graphical Representation

LDPC codes are usually defined in terms of a sparse bipartite graph, the so-called Tanner graph (Tanner, 1981)[17], in which we can represent the parity check matrix, in this graph branches link two different classes of nodes to each other:

- The first class of nodes called variable nodes (bit nodes), correspond to the bits of the codewords $(v_j, j = 1, ..., n)$, and therefore to the columns of H.
- The second class of nodes, called parity check nodes, correspond to the parity check equations $(c_i, i = 1, ..., m)$, and therefore to the rows of H.

Thus, to each branch linking a variable node v_j to a parity check node c_i corresponds the 1 that is situated at the intersection of the $j-th$ column and the $i-th$ row of the parity check matrix.

A cycle in a Tanner graph is a sequence of connected nodes which starts and ends at the same node in the graph and which contains other nodes no more than once. The length of a cycle is the number of edges it contains, and the girth of a graph is the size of its smallest cycle.

2.2 Proprieties of LDPC Codes

For an (n, k) LDPC we denote :

$$\mathbf{H} = \begin{array}{c} \\ \end{array}\begin{array}{ccccccc|c} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & \\ \hline 1 & 1 & 0 & 1 & 0 & 1 & 0 & c_1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & c_2 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 & c_3 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & c_4 \end{array}$$

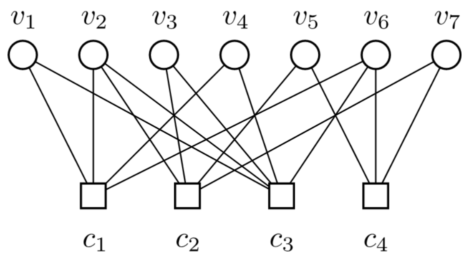


Figure 2.3: An example of H matrix and the corresponding Tanner graph

- Information word $x = (x_0, x_1, ..., x_{k-1})$ of length k.
- Code word $c = (c_0, c_1, ..., c_{k-1})$ of length n.
- The number of code words $M = 2^k$. Received word $r = (r_0, r_1, ..., r_{n-1})$ of length n.
- Decoded code word $\hat{c} = (\hat{c}_0, \hat{c}_1, ..., \hat{c}_{k-1})$ of length n.
- Decoded information word $\hat{x} = (\hat{x}_0, \hat{x}_1, ..., \hat{x}_{k-1})$ of length k .
- code rate $R = k/n$

The weight distribution

The so-called weight distribution $W(x)$ of an (n, k) LDPC block code $c = c_1, c_2, ..., c_M$ describes how many code words exist with a specific weight.

The weight of a vector or matrix is the number of non-zero symbols in it. The density of a source of random symbols is the expected fraction of non-zero symbols. 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' .

The minimum Hamming Distance

The distance between two code words $c = (c_0, c_1, ..., c_{n-1})$ and $c' = (c'_0, c'_1, ..., c'_{n-1})$ is given by the so-called Hamming distance:

$$dis(c, c') = |\{i : c_i \neq c'_i, 0 \leq i < n\}| \tag{2.8}$$

The Hamming distance $dist(c, c')$ provides the number of different components of c and c' and thus measures how close the code words c and c' are to each other. For a code C consisting of $M = 2^k$ code words $c_1, c_2, ..., c_M$, the minimum Hamming distance is given by:

$$d = \underset{\forall c \neq c'}{min}\ dist(c, c') \tag{2.9}$$

2.3 Classifications of LDPC codes

There are two type of classification for the LDPC codes the first one depends to the regularity of the code which divided into two groups, the regular code and the irregular codes . The second classification based on the representation of data over the Galois Field which we will discuss it in the next chapter, there are also two groups of codes in this classification the Binary codes and the Non-Binary codes .

2.3.1 Regular and Irregular LDPC Codes

Regular Codes : Regular codes were the first to be introduced when R. Gallager introduced the LDPC codes in 1962 . The regularity of these codes is specified by the constant number of "1" in the rows and columns of the matrix H It's mean that w_r and w_c are constant and connected by the following relation:

$$w_r = w_c.n/m \tag{2.10}$$

The regular LDPC codes are then described by (n, w_r, w_c) which n representing the length of the code word, wr the weight of the lines and wc the weight of the columns. It is clear that w_r (respectively w_c) are very small numbers in comparison of n (respectively m) so that H is sparse (low density).

As the ratio R have a relation with n and m we can rewrite it with other way in function of w_r and w_c .

$$R = 1 - w_c/w_r \tag{2.11}$$

Irregular codes : In case where the distribution of the non-zero elements is not uniform, these LDPC codes are called irregular codes . The irregularity of these codes is not characterized just by w_r and w_c , but also by two polynomials $\lambda(x)$ and $\sigma(x)$ which helps to create these codes [6].

The study shows that the irregular LDPC codes have better performance than the regular codes . In the other hand the irregular codes have more implementation complexity than the regular codes .

2.3.2 Binary and Non Binary LDPC Codes

The binary LDPC code is described by a binary-valued $m \times n$ parity-check matrix H in $GF(2)$, where the GF is abbreviation of Galois Field. This code may be generalized to finite fields $GF(q)$, where q is a prime number, The elements of $GF(q)$ will be called symbols and we use the term bits when referring to the binary representation of symbols (when $q = 2$). For a code over $GF(q)$, each received symbol can be any of the q elements in $GF(q)$.

The Non binary LDPC codes over $GF(q)$ can be seen as the generalization of binary LDPC codes over $GF(2)$ vector space projected over a finite field $GF(q)$, where $q = 2^m$ $(m \in Z^+)$. In this case, each symbol can be represented by a $m - bit$ binary tuple[18].

$$H = \begin{bmatrix} h_{00} & h_{01} & h_{02} & 0 & 0 & 0 \\ 0 & h_{11} & 0 & h_{13} & h_{14} & 0 \\ h_{20} & 0 & 0 & h_{23} & 0 & h_{25} \\ 0 & 0 & h_{32} & 0 & h_{34} & h_{35} \end{bmatrix} \tag{2.12}$$

In the parity check matrix H, of a NB-LDPC code over $GF(q)$, the nonzero entries are elements of $GF(q)$. Also each information and codeword symbol is an element of $GF(q)$. We will present the Non binary code and the Galois Field with details in the next chapter .

2.4 Construction of LDPC Codes

In contrast to other block codes, LDPC codes are constructed using the parity-check matrix not generator matrix. These codes can be constructed using either the matrix itself or its graphical representation, the Tanner graph.

The main process of LDPC code construction is simply a replacement of a small number of values in a zeros matrix by the $GF(q)$'s symbols (1's for binary code) in such a way that the rows and columns have the required degree distribution.

There are several algorithms to construct suitable LDPC codes. Gallager himself introduced one. Further, MacKay 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.

2.4.1 Gallager matrix construction for Regular Codes

The original LDPC codes presented by Gallager were regular and defined by a banded structure in H, Here is the basic constraints of Gallager code construction :

- The parity-check matrix has a fixed column weight w_c and a fixed row weight w_r.
- The parity-check matrix is divided into w_c sub-matrices, each containing a single 1 in each column.
- Without loss of generality, the first sub-matrix is constructed in some predetermined manner.
- The subsequent sub-matrices are random column permutations of the first sub-matrix.

$$H = \left[\begin{array}{cccccccccccc}
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
\hline
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
\hline
1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1
\end{array}\right].$$

Figure 2.4: Example of length-12 (3,4)-reguler Gallager parity-check matrix

Since H is not in systematic form, Gaussian elimination using row operations and reordering of columns needs to be performed to derive a parity-check matrix.$H = [P^T|I_m]$ Then the original H has to be redefined to include the column reordering as per the Gaussian elimination. The corresponding generator matrix is then $G = [I_k|P]$.

2.4.2 Mackay and Neal matrix construction

In The LDPC codes construction proposed by MacKay and Neal the columns of H are added one column at a time from left to right. The weight of each column is chosen so as to obtain the correct bit degree distribution and the locations of the non-zero entries in each column are chosen randomly from those rows that are not yet full. If at any point there are rows with more positions unfilled than there are columns remaining to be added, the row degree distributions for H will not be completely correct. Then the process can be started again or backtracked by a few columns until the correct row degrees are obtained.

$$H = \begin{bmatrix} 1&0&0&0&0&1&0&1&0&1&0&0 \\ 1&0&0&1&1&0&0&0&0&0&\mathbf{1}&0 \\ 0&1&0&0&1&0&1&0&1&0&0&0 \\ 0&0&1&0&0&1&0&0&0&0&\mathbf{1}&1 \\ 0&0&1&0&0&0&1&1&0&0&0&1 \\ 0&1&0&0&1&0&0&0&1&0&\mathbf{1}&0 \\ 1&0&0&1&0&0&1&0&0&1&0&0 \\ 0&1&0&0&0&1&0&1&0&1&0&0 \\ 0&0&1&1&0&0&0&0&1&0&0&1 \end{bmatrix}.$$

Figure 2.5: Example of a length-12(3,4)-regular MacKay-Neal parity-check matrix

In the example 2.5 when adding the $11th$ column, shown in bold, the unfilled rows were the second, fourth, fifth, sixth and ninth, from which the second, fourth and sixth were chosen.

2.5 Encoding operation

The objective of the encoding operation is to regroup a sequence of information symbols into words (or blocks) of equal length K which are independently encoded than each information word uniquely mapped onto a code word of length N . Encoding an LDPC code can turn

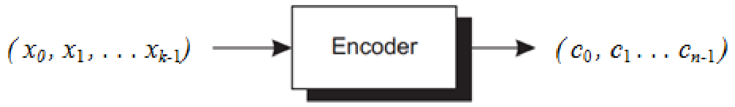
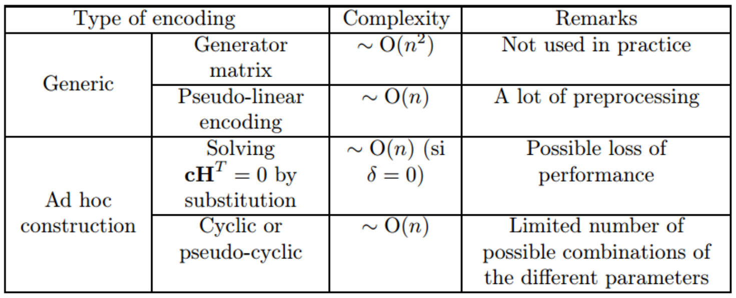


Figure 2.6: (n,k) LDPC Encoding

out to be relatively complex if matrix H does not have a particular structure. There exist generic encoding solutions, including an algorithm with high complexity, requiring complex preprocessing on the matrix H. Another solution involves directly building matrix H so as to obtain a systematic code very simple to encode[6].

Each one of the two encoding types have mentioned contain many algorithm or method, for the generic encoding solutions there are the encoding with a generator matrix method and the coding with linear complexity method in the other hand the second encoding type has also the coding with a sparse generator matrix, encoding by solving the system $c.H^T = 0$ obtained by substitution and the cyclic coding method. in our thesis we will give details for just two encoding methods. The table summarizes the different possible types of coding have mentioned .



Type of encoding		Complexity	Remarks
Generic	Generator matrix	$\sim$ O(n^2)	Not used in practice
	Pseudo-linear encoding	$\sim$ O(n)	A lot of preprocessing
Ad hoc construction	Solving $\mathbf{cH}^T = 0$ by substitution	$\sim$ O(n) (si $\delta = 0$)	Possible loss of performance
	Cyclic or pseudo-cyclic	$\sim$ O(n)	Limited number of possible combinations of the different parameters

Figure 2.7: Summary of different possible encodings

Encoding with a generator matrix:

The coding can be done via the generator matrix G of size $K \times N$ of the code, such as defined the section 2.1.2. As we have seen, LDPC codes are defined from their parity check matrix H, which is generally not systematic. P transformation of H into a systematic matrix H_{sys} =[P I_{N-K}] is possible, for example with the Gaussian elimination algorithm over the $GF(2)$. This relatively simple technique, however, has a major drawback: the generator matrix G_{sys} =[I_K P^T] of the systematic code is generally not sparse. The coding complexity increases rapidly . which makes this operation too complex for usual length codes.

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

$$(2) \qquad H_{\mathrm{rr}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

$$(3) \qquad H_{\mathrm{std}} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$(4) \qquad G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

Figure 2.8: An example of Gaussian elimination technique over the GF(2) for length-10 rate-1/2 LDPC code given by H matrix

Cyclic coding:

The classes of LDPC codes defined by finite geometry or by projective geometry enable cyclic or pseudo-cyclic codes to be obtained .The codes thus obtained can be encoded efficiently by using shift registers. In addition, they offer good properties in terms of the distribution of cycle length. The main drawback is that the cardinal of these classes of code is relatively small. These classes therefore offer only a very limited number of possible size–rate-irregularity profile combinations.

2.6 LDPC Decoding

2.6.1 Error Detection and Correction

Error detection is a simple matter: All one does is compute 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 goal of error correction is to form an estimate $\hat{c}$ of the transmitted codeword c, given the received word r. The most likely estimate will be the one that assumes the minimum number of errors occurred on the channel.

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

2.6.2 Decoding Algorithms

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.*

As it is known that a factor graph represents a factorization of the global code constraint into the local code constraints which are represented by the connection between variable and check nodes. Each variable node sends to the check nodes with which it is associated a message about the estimated value of the variable. The set of the messages received enables the check node to compute then return the extrinsic information. The successive processing of the variable then check nodes make up one iteration. At each iteration, there is therefore a bilateral exchange of messages between the parity nodes and variable nodes, on the arcs of the bipartite graph representing the LDPC code. One iteration is illustrated in the Figure 2.9 . It can be construed that the extrinsic message is a $soft - value$ for a symbol when the direct observation of the symbol is not considered in the computation of this specific value.

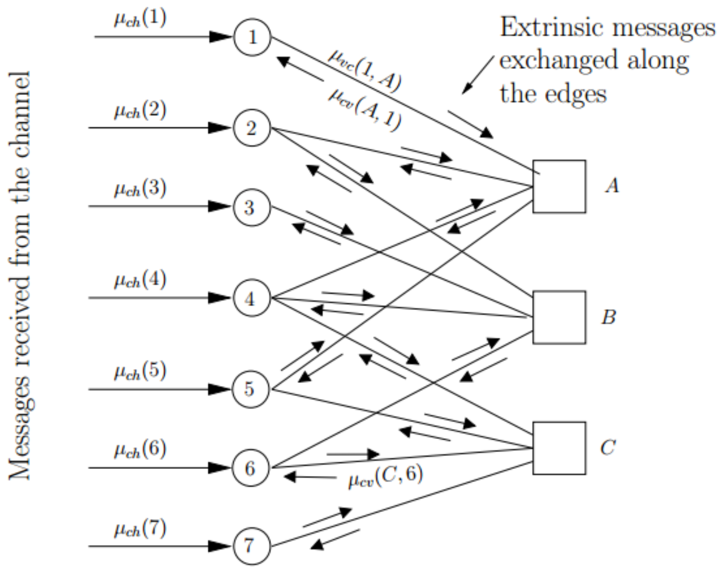


Figure 2.9: One complete iteration message passing example

The basic steps of the message passing algorithm are :

1. **Initialization:**

   The incoming messages received from the channel at the variable nodes are directly passed along the edges to the neighbouring check nodes because there are no incoming messages (extrinsic) from the check nodes in the first iteration.

2. **Updating the check nodes (CNs):**

   The check nodes perform local decoding operations to compute outgoing messages (extrinsic) depending on the incoming messages received from the neighbouring variable nodes. Thereafter, these new outgoing messages are sent back along the edges to the neighbouring variable nodes.

3. **Updating the variable node (VNs):**

   The variable nodes will perform the local decoding operations in the same way to compute the outgoing messages from the incoming messages received from both the channel and the neighbouring check nodes.

4. **Tentative Decoding :**

   After a complete iteration (updating of the CNs and VNs) the last operation is the calculation of hard decision messages and checking the codeword validity by using the syndrome .

   In this way, the iterations will continue to update the extrinsic messages unless the valid codeword is found or some stopping criterion is fulfilled (achieving the limit number of iterations)

   One important message-passing algorithm is the belief propagation algorithm which was presented by Robert Gallager in his PhD thesis. This algorithm has developed several times under different names. The most common ones are *sum-product algorithm* (SPA)[19], *min-sum*[20], extended *min-sum* (EMS) algorithms and the *min-max algorithm*[21], these algorithms will be presented in the next chapters .

Chapter 3

Chapter 3: Non Binary LDPC Code

The non-binary low density parity check (NB-LDPC) codes can achieve better error-correcting performance than binary LDPC codes when the code length is moderate. Despite the error-correcting performance advantages, NB-LDPC codes suffer from high decoding complexity. For a code over $GF(q)$, each received symbol can be any of the q elements in $GF(q)$. Hence, vectors of q messages representing the probability that the received symbol equals each element need to be computed, stored, and passed between the check and variable nodes during the decoding process.

3.1 Introduction to Galois Field

Many error-correcting codes, such as Reed-Solomon (RS) and low-density parity-check (LDPC) codes are defined over finite fields. Finite fields are also referred to as Galois Fields, which are named after a nineteenth-century French mathematician, Evariste Galois. The purpose of this section is to provide an elementary knowledge of algebra that will aid in the understanding of the material in the next sections. The treatment is basically descriptive and no attempt is made to be mathematically rigorous.

3.1.1 Basic Algebra Notions

Groups

A group is a set of elements G on which an operation $(\cdot)$ is defined such that the following properties are satisfied:

1. Closure: if $a; b \in G$ then $a \cdot b = c \in G$.
2. Associativity: for all $a; b; c \in G$, $(a \cdot b) \cdot \mathrm{c} = a \cdot (b \cdot c)$.
3. Identity: there exists a unique element $e \in G$ such that $a \cdot e = e \cdot a = a$.
4. Inverse: for all $a \in G$ there exists a unique element $a_0 \in G$ such that $\mathrm{a} \cdot a_0 = a_0 \cdot \mathrm{a} = \mathrm{e}$.

   A group is said to be commutative or abelian if it satisfies the above properties as well as:

5. Commutativity: for all $a; b \in G$, $a \cdot b = b \cdot a$.

The order of a group is the number of elements it contains, i.e. $ord(G)$, which is the cardinality of set G. If the order is finite, then the group is said to be finite group. A cyclic group is a group where all elements $\gamma \in G$ are obtained from the powers α_i of one element $\alpha \in G$. The powers are defined according to

$$\alpha_0 = e, \alpha_1 = \alpha^1, \alpha_2 = \alpha^2, ...$$

This particular element α is the primitive element of the group G.

Fields

A field is a set F, together with two operations: the additive operation denoted by $+$, and the multiplicative operation denoted by $\cdot$, such that the following conditions hold:

1. F forms an abelian group under "+" with (additive) identity element "0". The additive inverse of a is denoted $-a$.
2. $F - \{0\}$ forms an abelian group under " $\cdot$ " with (multiplicative) identity element "1". The multiplicative inverse of a is denoted $a - 1$.
3. " + " and " $\cdot$ " distribute: $a.(b + c) = (a.b) + (a.c)$, for all $a; b; c \in F$.

Subfields

Let F be a field under "+" and "$\cdot$". Let $K \subseteq F$. Then K is a subfield of F if it is also a field under "+" and "$\cdot$". If K is a subfield of F , then F is an extension field of K.

3.1.2 The finite Field GF(2)

The number of elements in a finite field can be a prime p . This field with p elements is denoted by $GF(p)$. The number of elements in a field is also called the order of the field. $GF(2)$ is the finite field with the least number of elements. One way to create a Galois field is to use the set of integers $\{0..., p - 1\}$ where p is prime and $(+)$ and $(\cdot)$ are $modulo - p$ addition and $modulo - p$ multiplication, respectively. The $modulo - p$ addition operation of a and b is by definition:

$$a \oplus_p b = (a + b) \ mod \ p$$

and the $modulo - p$ multiplication operation of a and b is by definition:

$$a \otimes_p b = (a.b) \ mod \ p$$

where The notation $b = a \ mod \ n$, means that b is the remainder of the division Euclidean of a by n.

For example when $p = 2$,the $GF(2)$ is the 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 $GF(2)$,the $GF(2)$ addition is the XOR operation and $GF(2)$ multiplication is the AND operation .

3.1.3 Extended Galois Field GF(2^m)

We know that the integers $\{0, ..., q-1\}$ cannot form a field under $modulo - q$ multiplication if q is not prime. However this does not mean that $GF(q)$ does not exist if q is not prime, it just means we can't use $modulo - q$ multiplication. and since we can create extension fields (subfields)of the $GF(q)$ one of these extension fields is based on power of p, the $GF(p^m)$ where m is a positive integer and p^m is the new order (cardinality) . The data in digital communication and storage systems is binary $\{0, 1\}$ as a result the $GF(2)$ is used. Therefore, the extension fields $GF(2^m)$ $(m \in Z^+)$ are usually used, since each field element can be represented by a binary vector [18].

In particular, the Galois Fields of order $q = 2^m$, has a great interest in coding theory and their applications . Indeed, an element of Galois field $GF(2^m)$ can be represented in a unique way in the form of a binary symbol of m bits. Reed Solomon (RS) codes, which are one of the most commercially successful code, uses Galois Fields of the form $GF(2^m)$ with m often equal to 8.

3.2 GF(2^m) Construction

3.2.1 Polynomial Method

Polynomials over GF (2)

A polynomial of shape :

$$f(X) = a_n X_n + a_{n-1} X_{n-1} + ... + a_1 X + a_0 \tag{3.1}$$

whose coefficients a_i are elements of a field $GF(2)$ ($\{0, 1\}$), is called a polynomial over $GF(2)$. The positive integer n is called the degree of the polynomial and is denoted by $deg(f)$. The $GF(2)[X]$ denote the collection of the polynomials over $GF(2)$ with coefficients a_i .

Irreducible polynomial

A polynomial $p(x)$ with degree m is irreducible in $GF(2)[x]$ if $p(x)$ cannot be factored into a product of lower-degree polynomials in $GF(2)[x]$, The $P(x)$ can be used to construct extension field $GF(2^m)$ [7] .

To construct the $GF(q)$ with $q = 2^m$, we will proceed in the same way as to construct the $GF(p)$ with p is a prime but by reasoning not on the relative integers but on the polynomials of $GF(2)[X]$. For this we suppose that we know a polynomial $p(x)$ of $GF(2)[X]$ of irreducible degree m. The set of polynomials modulo $p(x)$ with coefficients in $GF(2)$ is a set Finite: we can demonstrate that it is a field. To find to which class of polynomials modulo $p(x)$ belongs a polynomial $f(x)$ we do the Euclidean division of $f(x)$ by $p(x)$:

$$f(x) = q(x)p(x) + r(x) \tag{3.2}$$

Then $f(x)$ belongs to the class of $r(x)$. The set $GF(2)$ $[p(x)]$ is formed by the set of polynomial classes of degree less than or equal to $m - 1$ (since they are the Euclidean divides by $p(x)$). We say that the set $GF(2)[p(x)]$ is formed by the set of all polynomials of degree less than or equal to $m - 1$. Each polynomial contains m elements of $GF(2)$ which can take p values: therefore the number of elements of $GF(2)[p(x)]$ is 2^m.

Example : construction of $GF(2^m), m = 3, q = 8$: In the $GF(2)$ we chose the irreducible polynomial:

$$p(x) = x^3 + x + 1.$$

The set of values of x are $\{0, 1\}$ in this case the 0 and 1 are not roots for $p(x)$ so $x + 1$ and x are not dividers of $p(x)$. No polynomial of degree 2 divides $p(x)$ otherwise $p(x)$ would be the product of a polynomial of degree 2 and of a polynomial of degree 1 which is impossible since no polynomial of degree 1 divides $p(x)$. All polynomials of degree less than 3 form a class of polynomials modulo $p(x)$ They are presented on the table :

Polynomials	Binary representation x^2 x x^0
$f_0(x) = 0$	0 0 0
$f_1(x) = 1$	0 0 1
$f_2(x) = x$	0 1 0
$f_3(x) = x + 1$	0 1 1
$f_4(x) = x^2$	1 0 0
$f_5(x) = x^2 + 1$	1 0 1
$f_6(x) = x^2 + x$	1 1 0
$f_7(x) = x^2 + x + 1$	1 1 1

Table 3.1: Polynomial construction example (GF(8))

This set contains 2^3 elements, it is the $GF(2^3)$ or $GF(8)$. We see that the elements of the $GF(2^m)$ defined in this way are not very convenient to write. To use them, they can be represented as $m - tuples$ of the starting field $GF(2)$, i.e m bits in this case. We have defined the set which is still not very original. To define the field, we must Define addition and multiplication operations. This representation in the form of polynomials is quite limited for theoretical developments, we will see another method of constructing $GF(2^m)$.

3.2.2 Primitive element method

To define the $GF(2^m)$ by the first method, we have found an irreducible polynomial $p(x)$ of degree m, with coefficients in $GF(2)$. This polynomial being prime has no root in $GF(2)$ (in $GF(2)$ the element $\{0, 1\}$ are not root of $p(x)$). But in this method we imagine that there are roots "elsewhere" than in $GF(2)$, we can find m root, we denote by α one of them . This root is then the primitive element of the $GF(2^m)$. Indeed, we will find the elements of the Galois Field by considering all the powers of α : $\{\alpha^0, \alpha, ..., \alpha^{2^m-2}\}$.

Primitive polynomial

Given an irreducible polynomial of degree m, to test whether it is primitive, divide it from $x^n - 1$ where $m < n < p^m - 1$. If no such n gives 0 remainder, then it is primitive.

Binary extension field $GF(2^m)$, is usually adopted for hardware implementations, since each element can be represented by a $m - bit$ binary tuple.

Example : constructing of $GF(2^3)$:

Let $p(x) = x^3 + x + 1$ be the irreducible polynomial. We call α one of its roots, and we will express the different powers of α in function of $\alpha^0 = 1, \alpha, \alpha^2$. Indeed as soon as α^3 can be expressed in function of the lower powers because: $\alpha^3 + \alpha + 1 = 0$ therefore $\alpha^3 = \alpha + 1$. (There is no negative element in $GF(2)$) .

Continue for the other elements :

$$\alpha^4 = \alpha^2 + \alpha$$
$$\alpha^5 = \alpha^3 + \alpha^2 = \alpha + 1 + \alpha^2$$

In general we can use the Euclidian division, dividing α^n by $\alpha^3 + \alpha + 1$:
$\alpha^n = q(\alpha)(\alpha^3 + \alpha + 1) + r(\alpha)$ as $\alpha^3 + \alpha + 1 = 0$, $\alpha^n = r(\alpha)$. continue in this way:

$$\alpha^6 = (\alpha^3 + \alpha + 1)\alpha^3 + \alpha + 1) + \alpha^2 + 1 = \alpha^2 + 1$$
$$\alpha^7 = (\alpha^4 + \alpha^2 + \alpha)(\alpha^3 + \alpha + 1) + 1 = 1$$

We see that $\alpha^7 = 1$, so $\alpha^7 + 1 = 1 + 1 = 0$. (the addition is XOR operation). This is general : if the polynomial $p(x)$ is "correctly" chosen, $\alpha^{2^m-1} + 1 = 0$, also for a $GF(p^m)$ with $p > 2$, $\alpha^{q^m-1} - 1 = 0$.

3.2.3 Properties of the Galois Field $GF(2^m)$

- The $GF(2^m)$ is the set of primitive element α powers and the element 0

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

- The Field actually contains 2^m elements because $\alpha^{2^m-1} = 1$.

- α is a root of a polynomial $p(x)$ of degree m irreducible in $GF(2)$. This Polynomial must also be primitive.

  That means no other power j lower than $q^m - 1$ can be found such that $\alpha^j = 1$.

The correspondence between the notation of the elements using the power of α and the notation using polynomials is done by replacing α by x, we obtain the table of the $GF(8)$ for the previous example(Table 3.2) .

3.3 Galois Field (2^m) arithmetic

As explained previously, finite fields can be constructed differently, and the field elements can be represented as linear combinations of the elements in a basis (m element for the $GF(2^m)$), as well as powers of a primitive element. the hardware complexities of finite field operations are heavily dependent on the element representations.

Proper representations can be chosen based on the computations involved in a given application. Moreover, a single system can employ different representations in different units to minimize the overall hardware complexity.

	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 3.2: $GF(8)$ Power representation

3.3.1 Polynomial representation

Addition

The addition of 2 elements of the $GF(2^m)$ in general is the addition of the term polynomials to term. We must remember the property of the addition in the starting field where the coefficients are taken. In case of the $GF(2^m)$ the starting field is the $GF(2)$ the addition in this field is performed $modulo - 2$ or the XOR operation, so the $GF(2^m)$ has the same operation for each bit that means the bit-wise XOR .The figure represent the addition table of the $GF(8)$.for $x = (x_1x_2x_3)$ and $y = (y_1y_2y_3) \in GF(8)$,The addition operation of the $GF(8)$ is :

$$x + y = (x_1x_2x_3)\ XOR\ (y_1y_2y_3)$$

Multiplication

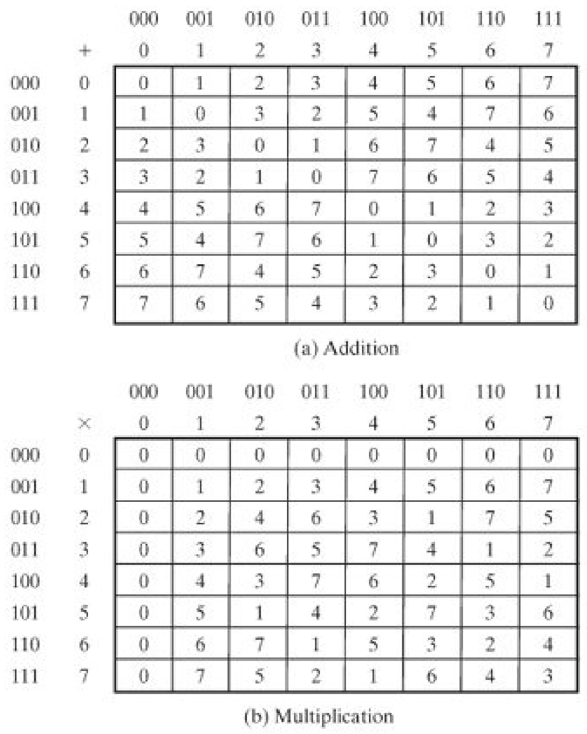
In this representation which based on the polynomials the Multiplication operation is performed by the polynomials multiplication than find the class of the obtained polynomial. The figure 3.1(b) represent the multiplication table of the $GF(8)$.

We can conclude from the operations of the polynomial (or basis) representation that the addition operation are simple and can easily implemented with no complexity as it's based just on the bit-wise XOR in the other hand the multiplication operation is more complicated and not easy to implement because of the the competitions complexities (multiplication and division of polynomials) .

3.3.2 Power representation

Addition

If we use the polynomial notation of α. The sum will be the same as we explained it in the polynomial representation, i.e bit by bit *Modulo-2* or the *bit-wise XOR*. So the addition cannot be done directly using power representation. Power representation has to be converted to a basis representation before the additions are done. An example for such a conversion in $GF(2^8)$ is shown in Table 3.5 After the addition is completed in a basis representation, a reverse mapping is needed to get the power representation of the sum. For $GF(2^3)$, the conversion or mapping table has 2^3 (2^m in general case) lines and each line has m bits. The



		000	001	010	011	100	101	110	111
	+	0	1	2	3	4	5	6	7
000	0	0	1	2	3	4	5	6	7
001	1	1	0	3	2	5	4	7	6
010	2	2	3	0	1	6	7	4	5
011	3	3	2	1	0	7	6	5	4
100	4	4	5	6	7	0	1	2	3
101	5	5	4	7	6	1	0	3	2
110	6	6	7	4	5	2	3	0	1
111	7	7	6	5	4	3	2	1	0

(a) Addition

		000	001	010	011	100	101	110	111
	×	0	1	2	3	4	5	6	7
000	0	0	0	0	0	0	0	0	0
001	1	0	1	2	3	4	5	6	7
010	2	0	2	4	6	3	1	7	5
011	3	0	3	6	5	7	4	1	2
100	4	0	4	3	7	6	2	5	1
101	5	0	5	1	4	2	7	3	6
110	6	0	6	7	1	5	3	2	4
111	7	0	7	5	2	1	6	4	3

(b) Multiplication

Figure 3.1: GF(8) Addition (a) and Multiplication (b) table using the polynomial representation

implementation of this table becomes quite hardware consuming when m is not small. the figure 3.2 represent the $GF(2^3)$ addition table for power representation.

Multiplication

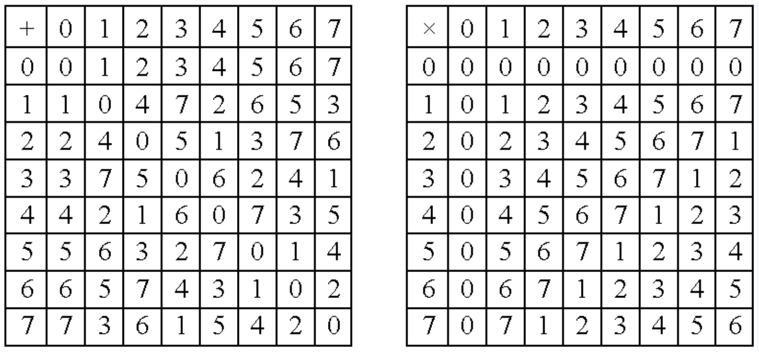
Using power representation, multiplication over $GF(2^m)$ can be performed by adding up the exponents of the operands modulo $2^m - 1$:

$$\alpha^i.\alpha^j = \alpha^{(i+j)mod(2^m-1)} \tag{3.3}$$

The figure 3.2 represent the multiplication table of the $GF(8)$ for the power representation. We can conclude that the power representation has low complexity and easy to implement in the case of the multiplication operation and using basis representations are more complicated than those based on power representation. but for the addition it's better to use the conversion to the basis presentation .

power representation	3-tuple form	polynomial representation
0	000	0
1	100	1
α	010	2
α^2	001	4
α^3	110	3
α^4	011	6
α^5	111	7
α^6	101	5

Table 3.3: $GF(8)$ power-polynomial representation conversion



+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	0	4	7	2	6	5	3
2	2	4	0	5	1	3	7	6
3	3	7	5	0	6	2	4	1
4	4	2	1	6	0	7	3	5
5	5	6	3	2	7	0	1	4
6	6	5	7	4	3	1	0	2
7	7	3	6	1	5	4	2	0

×	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	3	4	5	6	7	1
3	0	3	4	5	6	7	1	2
4	0	4	5	6	7	1	2	3
5	0	5	6	7	1	2	3	4
6	0	6	7	1	2	3	4	5
7	0	7	1	2	3	4	5	6

Figure 3.2: GF(8) power representation operations

3.4 Non Binary LDPC codes over GF(q)

The LDPC codes defined on the Galois Field $GF(q = 2^m)$, $m > 1$ whose symbols of the code words are elements of the Galois field $GF(q)$, $q > 2$. we called non-binary LDPC codes. In the parity check matrix H of a NB-LDPC code over $GF(q)$, the nonzero entries are elements of $GF(q)$. Also each information and codeword symbol is an element of $GF(q)$. For communication and data storage systems, $GF(2^m)$ is usually adopted. In this case, each symbol can be represented by a p-bit binary tuple.

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

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

Figure 3.3: Non-binary prity-check matrix

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 h_{ij} . Figure 3.4 illustrates the bipartite graph for equation 2 in Figure 3.3 by adding the permutation nodes that correspond to the elements h_{11},h_{13} and h_{14} .

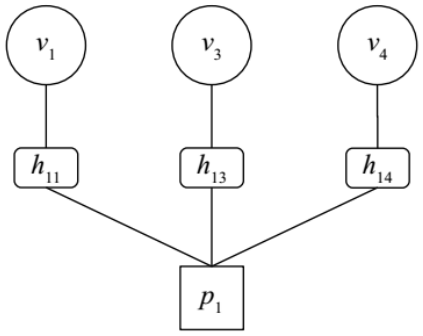


Figure 3.4: Graphical representation of non-binary parity-equation

The binary LDPC codes have asymptotic performances approaching the Shannon limit. However, for small or medium-sized code words, the performance of binary LDPC codes degrade considerably. It has been shown that this loss can be compensated by using high cardinality $GF(q)$-LDPC codes. Moreover, the high cardinality of the codes ensures better resistance to frame errors . This improvement in performance can be explained intuitively by the fact that several bits are grouped together in a single non-binary symbol. Therefore, erroneous bits are confined to fewer non-binary symbols, and subsequently parity constraints are affected by fewer errors. Nevertheless, the improvement of the performances by the increase of the order of the Galois field is accompanied by an exorbitant increase in the complexity of the decoding which constitutes a brake on the practical use of the $GF(q)$-LDPC codes.

Chapter 4

Chapter 4: Iterative Decoding Algorithms

The graphical representation of the NB-LDPC codes can be used in the implementation of algorithms whose effectiveness has been shown on graph models such as the Belief Propagation algorithm generally noted BP. The algorithm BP is an iterative algorithm which is part of the class messages passage algorithms (mentioned in chapter 2). They are called so because, at each iteration of the algorithm, messages are transmitted from the CNs to their related VNs, and reversely from the VNs to their related CNs . We distinguish two types of messages :

- The intrinsic messages : a priori information messages calculated only from the observations of the channel.
- The extrinsic messages: each branch of a node of the graph (VNs and CNs) circulates an incoming message and an outgoing message.

The decoder must be able to converge to a valid code word after a finite number of iterations, stop the decoding as soon as it converges to this word. To avoid an infinite execution a maximum number of iterations is fixed. In the algorithm BP, the messages exchanged are a posteriori probabilities calculated on the symbols of the code word. However, the BP algorithm suffers from a prohibitive computational complexity, which comes essentially from the calculations performed during the update of the parity constraints. By noting that the updating of CNs can be modeled by convolution products, The log-BP algorithm (Sum-Product Algorithm)[19] is an algorithm in which the four decoding steps are carried out in the logarithmic domain to allow a hardware implementation less sensitive to quantization errors, and therefore better suited to fixed-point arithmetic. However, updating the CNs always requires a large amount of calculation and the complexity of the decoder still high. The BP and log-BP algorithms are so-called optimal decoding algorithms because they do not use any mathematical approximation to reduce the complexity of the decoding. The BP algorithm and its variants guarantee optimal decoding performances but they have not great interest for a hardware implementation. Consequently, other algorithms based on approximations of the BP algorithm have been proposed with the aim of ensuring a reasonable performance / complexity compromise. We mainly cite the Min-Sum, its variant EMS (Extended Min-Sum), and the Min-Max algorithm which can be considered as an approximation of the Min-Sum algorithm.

4.1 Believe Propagation Algorithm

For a NB-LDPC code over $GF(q)$, the observation of the *nth* received symbol is r_n, the *nth* transmitted symbols, is c_n, they can be any of the q elements in $GF(q)$. Therefore, the messages passed between the check and variables nodes in the decoding process are vectors of q messages. By using the power representation of the $GF(q)$ elements, Let α be a primitive element of $GF(q)$. Then all the elements of $GF(q)$ are expressed as $0, 1, \alpha, \alpha^2, ..., \alpha^{q-2}$.

Let $c = [c_0, c_1, ..., c_{N-1}]$, $c_i \in GF(q = 2^m)$ the code word transmitted. The role of decoding is to converge to a valid code word $\hat{c} = [\hat{c_0}, \hat{c_1}, ..., \hat{c}_{N-1}]$ from a noisy signal $r = [r_0, r_1, ..., r_{N-1}]$, where r_i is the noisy version of c_i. The decoding is successful if $\hat{c} = c$.

In the BP algorithm, the intrinsic information of a VN v_i is a vector of q a posteriori probabilities defined by Equation 4.1:

$$I_i = [p(v_i = \beta_0|r_i), p(v_i = \beta_1|r_i), ..., p(v_i = \beta_{q-1}|r_i)] \tag{4.1}$$

where $p(a|b)$ is the conditional probability of a given b.

Let $i = 0, 1, ..., M-1$ and $j = 0, 1, ..., N-1$. If the element h_{ij} of the parity matrix H is nonzero then $M_{v_j c_i}$ denotes the message sent by the VN v_j to the CN c_i and $M_{c_i v_j}$ the message sent by the CN c_i to the VN v_j.

The set of steps of the BP algorithm are:

1. **Initialization:** Each VN v_j sends its intrinsic information to all of their related CNs.

2. **Update of the VNs:** A VN v_j receives w_c messages $\tilde{M}_{c_i v_j}$ and supplies w_c messages $M_{v_j c_i}$. Messages from v_j are computed by Equation 4.2 . Each outgoing message is a function of all incoming messages to v_j except that of c_i.

$$M_{v_j c_i}[\beta] = \mu_{v_j c_i} \cdot I_j[\beta] \cdot \prod_{\substack{s \neq i \\ h_{s,j} \neq 0}} \tilde{M}_{c_s v_j}(\beta) \quad \beta \in GF(q) \tag{4.2}$$

   With $\mu_{v_j c_i}$ is a normalization factor such that $\sum\limits_{\beta \in GF(q)} M_{v_j c_i}[\beta] = 1$

3. **Permutation:** Before entering the CN c_i, the message $M_{v_j c_i}$ is multiplied by the non-zero element h_{ij} of the parity matrix. The resulting message $\tilde{M}_{v_j c_i}$ is obtained by equation 4.3 .

$$\tilde{M}_{v_j c_i}[\beta] = M_{v_j c_i}[\beta \cdot h_{ij}^{-1}] \quad \beta \in GF(q) \tag{4.3}$$

4. **Updating the CNs:** Updating the CN p_i is done by equation 4.4 .

$$M_{c_i v_j}[\beta] = \sum_{\theta_s = \beta} \prod_{\substack{s \neq j \\ h_{is} \neq 0}} \tilde{M}_{v_s c_i}[\theta_s] \tag{4.4}$$

   With β and θ are variables belonging to $GF(q)$. The updating of the CN p_i consists in calculating the probability of all combinations of symbols verifying the parity equation.

5. **Inverse Permutation :** Before entering the VN v_j, the message $M_{c_i v_j}$ is divided by the non-zero element h_{ij} of the parity matrix. The resulting message $\tilde{M}_{c_i v_j}$ is obtained by Equation 4.5 .

$$\tilde{M}_{c_i v_j}[\beta] = M_{c_i v_j}[\beta \cdot h_{ij}] \beta \in GF(q) \tag{4.5}$$

6. **Estimation of the code word:** At the end of each iteration, each VN v_j updates an a priori probability vector denoted APPj as in equation 4.6 .

$$APP_j[\beta] = \mu_{v_j} \cdot I_j[\beta] \cdot \prod_{h_{s,j} \neq 0} \tilde{M}_{p_s v_j}(\beta) \quad \beta \in GF(q) \tag{4.6}$$

With μ_{v_j} is a normalization factor such that $\sum\limits_{\beta \in GF(q)} APP_j[\beta] = 1$. Then the decision is made by equation 4.7 which consists in selecting the symbol which has the greatest probability in APPj.

$$\hat{c}_j = \underset{\beta \in GF(q)}{argmax} \{APP_j[\beta]\} \quad j = 0, 1, ...N - 1 \tag{4.7}$$

If the set of symbols $\hat{c}_j$ form a valid code word then the decoding ends.

4.2 The Sum-Product Algorithm (log-BP)

It is now well known that reduced complexity decoding algorithms are built in the logarithm domain, and this is for two main reasons. First, it transforms the products in into simple sums . The second reason is that log-domain algorithms are usually more robust to the quantization effects when the messages are stored on a small number of bits . The practical decoding algorithms for LDPC codes (Sum-Product, Min-Sum, EMS and Min-Max) are all expressed in the logarithm domain.

The reliability of a symbol can be measured by the Log-Likelihood Ratio (LLR) defined by Equation 4.8.

$$LLR(\beta) = \ln \frac{p(v_j = \beta | r_i)}{p(v_j = 0 | r_i)} \quad \beta \in GF(q) \tag{4.8}$$

Replacing the probabilities by LLRs in equations 4.2, 4.4 and 4.6 makes it possible on the one hand to transform the multiplication operations into additions operations and on the other hand to reduce the quantification errors. Thus, in the Sum-Product algorithm, the intrinsic information of a VN v_j is defined by equation 4.9.

$$I_j = [0, \ln \frac{p(v_j = \beta_1 | r_i)}{p(v_j = \beta_0 | r_i)}, ..., \ln \frac{p(v_j = \beta_{q-1} | r_i)}{p(v_j = \beta_0 | r_i)}] \tag{4.9}$$

The messages that circulate on the bipartite graph are composed of LLRs. The Sum-Product algorithm retains the same decoding steps of the BP algorithm while modifying the update equations. Indeed, the update of a VN v_j is done by equation 4.10.

$$M_{v_j c_i}[\beta] = I_j[\beta] + \sum_{\substack{s \neq j \\ h_{is} \neq 0}} \tilde{M}_{c_s v_j}[\theta_s] \tag{4.10}$$

The update of a CN c_i is done by equation 4.11.

$$M_{c_i v_j}[\beta] = \ln \sum_{\sum_{\substack{s \neq j \\ h_{is} \neq 0}} \theta_s = \beta} exp(\sum_{\substack{s \neq j \\ h_{is} \neq 0}} \tilde{M}_{v_s c_i}[\theta_s]) \tag{4.11}$$

Finally, the updating of the a priori information is done by equation 4.12 .

$$APP_j[\beta] = I_j[\beta] + \sum_{h_{s,j} \neq 0} \tilde{M}_{c_s v_j}(\beta) \quad \beta \in GF(q) \tag{4.12}$$

4.3 The Min-Sum algorithm

The Min-Sum algorithm was proposed to reduce the complexity of the Log-BP algorithm using an approximation of equation 1.16. Indeed, in the Min-Sum algorithm, the update of a CN c_i is performed by equation 4.13 .

$$M_{c_i v_j}[\beta] \approx \min_{\substack{s\neq j \\ h_{is}\neq 0}\ \theta_s=\beta} \{ \sum_{\substack{s\neq j \\ h_{is}\neq 0}} \tilde{M}_{v_s c_i}[\theta_s] \} \tag{4.13}$$

The Min-Sum algorithm thus makes it possible to simplify the decoder by eliminating the tables of correspondences necessary for the implementation of the exponential functions and logarithms and by minimizing the number of arithmetic operations.

4.4 The EMS algorithm

To simplify the Min-Sum decoder, the EMS algorithm has been introduced in which the messages circulating on the bipartite graph are truncated by selecting the most reliable n_m symbols among the possible q symbols, with $n_m << q$. However, the value of n_m must be carefully chosen so that the decoding performance does not undergo significant degradation.

In the case of the Sum-Product algorithm, the messages are vectors composed of q unsorted reliability values. Moreover, it is not necessary to explicitly indicate the value of the symbol associated with each of the reliabilities, since it can be easily deduced by its position in the message. Due to truncation, messages from the EMS algorithm must be sorted and the values of the symbols must be explicitly mentioned. The messages that circulate on the graph are therefore of the form $M = [(LLR(\theta_k), \theta_k)]_{0\leq k<n_m}$,with θ_k a variable in $GF(q)$. The massage M can be divided into two sub-message, the$M : GF$ denotes the partial message containing the set of symbols of the message M and$M : L$ denotes the vector containing the set of $LLRs$ of the message M. The most reliable symbol in M is $M : (0)$ (first position for the smallest LLR value) and the least reliable symbol is $M : GF(n_m - 1)$. The truncation of the messages results in a performance degradation which can be compensated by using a constant reliability value denoted γ for the symbols not retained during truncation. The value of γ is calculated as follows:

$$\gamma = M.L(n_m - 1) - offset \tag{4.14}$$

Where offset is a scalar determined by simulation so as to obtain the best possible BER.

The EMS algorithms retains the same decoding steps of the Min-Sum algorithm while modifying the update process. A compensation scalar γ_i is associated with each message $\tilde{M}_{c_i v_j}$. The value of γ_i is determined by equation 1.19. The outgoing message $M_{v_j c_i}$ contains the n_m most reliable symbols by combining the intrinsic information with the incoming messages except that of c_i. The reliability of a symbol $M_{v_j c_i}$ is obtained by equation 1.20.

$$M_{v_j c_i}.L(k) = I_i[M_{v_j c_i}.GF(k)] + \sum_{\substack{s\neq j \\ h_{is}\neq 0}} W_s(k) \tag{4.15}$$

With

$$W_s(k) = \begin{cases} \tilde{M}_{p_s v_j}[M_{v_j c_i}.GF(k)] & if\ M_{v_j c_i}.GF(k) \in \tilde{M}_{p_s v_j} \\ \gamma_s \quad otherwise \end{cases}$$

In the other hand updating the CNs step still the same as the Min-Sum algorithm, except that the number of outgoing messages are n_m rather than q.

4.5 The Min-Max Algorithm

The logarithmic likelihood ratio (LLR) can take negative values. However, it would be simpler to deal only with positive values. Therefore, there was a proposition to define the LLRs as follows:

$$LLR(\beta) = -\ln \frac{p(c=\beta|r)}{\max\limits_{\theta \in GF(2^m)} \{p(c=\theta|r)\}} \quad \beta \in GF(2^m) \tag{4.16}$$

Where $r = (r_0, r_1, ...r_{M-1})$ is the observation of the channel and $c = (c_0, c_1, ...c_{M-1})$ is the transmitted symbol. In this definition, the normalization is done by the probability of the most reliable symbol. It follows that the LLR of this symbol is always zero and the LLRs of the other symbols are positive.

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 4.1: Exapmle of LLRs values of GF(8)

Min-Max algorithm is an approximation of the MS algorithm, a modifications have been done in which makes it possible to simplify the processing at the CNs by replacing the sum in equation 4.13 by the operator max.

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

- Initialization: $M_{v_j c_i}[\beta] = I_j[\beta]$
  Iterations :
- Check node processing

$$M_{c_i v_j}[\beta] \approx \min_{\sum_{\substack{s \neq j \\ h_{is} \neq 0}} \theta_s = \beta} \{\max_{\substack{s \neq j \\ h_{is} \neq 0}} \tilde{M}_{v_s c_i}[\theta_s]\} \tag{4.17}$$

- Variable node processing

$$M'_{v_j c_i}[\beta] = I_j[\beta] + \sum_{\substack{s \neq j \\ h_{is} \neq 0}} \tilde{M}_{c_s v_j}[\beta] \quad \beta \in GF(q) \tag{4.18}$$

$$M_{v_j c_i}[\beta] = M'_{v_j c_i}[\beta] - \min_{\beta \in GF(q)} (M'_{v_j c_i}[\beta]) \tag{4.19}$$

- A posteriori information computation

$$APP_j[\beta] = I_j[\beta] + \sum_{h_{s,j} \neq 0} \tilde{M}_{c_s v_j}(\beta) \quad \beta \in GF(q) \tag{4.20}$$

After each iteration, hard decision for the ith symbol can be made as:

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

The iterations can be carried out until $H[r_0, r_1, r_2, ...]^T = 0$ or the maximum iteration number has been reached.

To simplify The CN operations it can be implemented efficiently by forward-backward scheme.This scheme consists in constructing the outgoing messages by a set of elementary operations allowing not to repeat the same calculations and to reduce the latency of processing.We will present it with details in the next chapter.

4.5.1 Variable Node processing example

In this example (see fig 4.1(a)) we are working on the $GF(8)$ and let the $w_c = 2$, that means each variable node has two CNs connected to it, We denote by M_0 the Intrinsic message received from the channel.

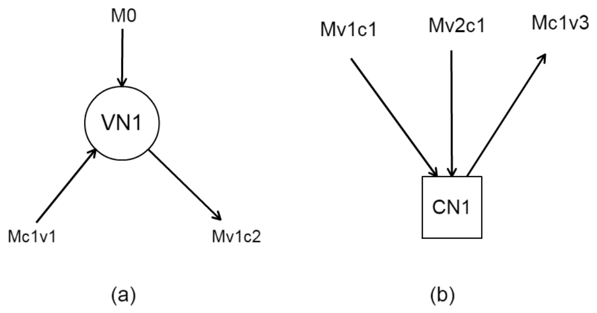


Figure 4.1: Variable and Check Node Messages processing

GF	LLR
α^2	5
α^5	9
α^3	15
α^1	0
0	12
α^6	3
α^0	16
α^4	5
M_0	

$+$

GF	LLR
α^4	10
α^3	13
0	4
α^2	0
α^0	17
α^1	9
α^6	6
α^5	8
$M_{c_1v_1}$	

$=$

GF	LLR
α^2	5
α^6	9
α^1	9
α^4	15
0	16
α^5	17
α^3	28
α^0	33
$M'_{v_1c_2}$	

$\rightarrow$

GF	LLR
α^2	0
α^6	4
α^1	4
α^4	10
0	11
α^5	12
α^3	23
α^0	28
$M_{v_1c_2}$	

Table 4.2: Varaible node processing example

As illustrated in the table (4.2) the output message $M_{v_1c_2}$ is calculated by doing the sum of the LLR values corresponding to the same GF element from $M_{c_1v_1}$ and the M_0, than an LLR normalization by subtracting the minimum from all the LLR values.

4.5.2 Check Node processing example

Here we have the $GF(4)$ and let $w_r = 3$ (see Fig 4.1(b)), so each CN connected to 3 VNs. let $M_{v_1c_1}$, $M_{v_2c_1}$ be the input messages and $M_{c_1v_3}$ the output message . This example illustrate the operations processed to compute the output message $M_{c_1v_3}$. The equation (4.21) present the check node min-max operation.

$$LLR(\alpha^k) = \underset{\alpha^k \in GF(q)}{MIN} \{ \underset{\substack{\alpha^i,\alpha^j \in GF(q)^2 \\ \alpha^k=\alpha^i+\alpha^j}}{MAX} (LLR(\alpha^i), LLR(\alpha^j))\} \tag{4.21}$$

$M_{v_1c_1}$

α^i	LLR(α^i)
0	7
α^0	0
α^1	3
α^2	11

$M_{v_2c_1}$

α^j	LLR(α^j)
0	9
α^0	6
α^1	4
α^2	0

Table 4.3: Input Data form for the CN processing example

The CN resulting message is presented in the table (4.4), and it's calculated by: First taking the maximum LLR values for all combination of two messages $M_{v_1c_1}$ and $M_{v_2c_1}$, which results 4^2 LLR values for 4 elements as illustrated in table (4.4), the GF element are obtained by doing the sum of the GF elements for each combination, at last we take the minimum LLR value for every GF element, these values and theirs corresponding GF(4) elements are the output message $M_{v_1v_3}$.

(GF, LLR)	(0,9)	(α^0,6)	(α^1,4)	(α^2,0)
(0,7)	(0,9)	(α^0,7)	(α^1,7)	(α^2,7)
(α^0,0)	(α^0,9)	(0,6)	(α^2,4)	(α^1,0)
(α^1,11)	(α^1,11)	(α^2,11)	(0,11)	(α^0,11)
(α^2,3)	(α^2,9)	(α^1,6)	(α^0,4)	(0,3)

$\Rightarrow$

α^k	LLR(α^k)
0	3
α^0	4
α^1	0
α^2	4

Table 4.4: Check Node min-max processing example

4.6 Performance comparison

4.6.1 Influence of iteration number on performance

For a QC-LDPC decoder with the systematic rate 1/2 and a block length K = 972 and thus a codeword length N = 1944. The simulation of BER performance for this code is illustrated in Figure 4.2, this simulation has been performed over an AWGN channel with BPSK modulation. Its object is to compare the BER performance of the sum-product algorithm for different number of iterations I. It can be observed that the BER performance improves considerably for small I and saturates roughly after 25 to 30 iterations. We can conclude that practical decoder implementations typically perform 5 to 10 iterations.

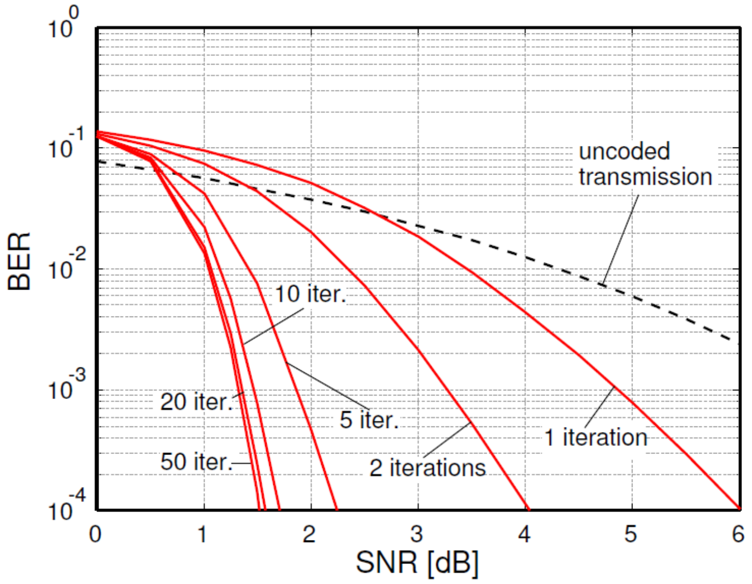


Figure 4.2: Iteration number influence on decoding performance [2]

4.6.2 EMS and Min-max algorithm comparison

The Min-max algorithm can be implemented by a more efficient architecture in comparison to the EMS, that's because of replacing the sum with the max operation in CN computations. Despite this, the performance of the Min-max algorithm is only slightly worse. Figure 4.3 show the bit error rates (BERs) of the EMS and Min-max algorithms for a (744, 653) QCNB-LDPC code over GF (2^5) with 15 decoding iterations under AWGN channel. This code is constructed using the method based on the primitive element (power representation). As

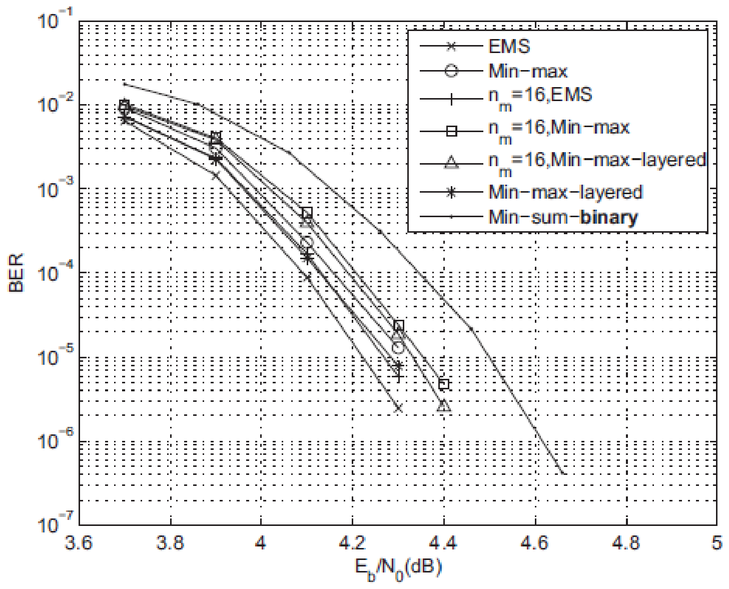


Figure 4.3: BERs of a (744, 653) NB-LDPC code over GF(25) and a (3683,3175) binary LDPC code under AWGN channel [3]

illustrated the Figure 4.3, the Min-max algorithm has only around 0.075 dB performance degradation at BER=10^{-4} compared to the EMS algorithm.

When using all q messages, this leads to vast memory requirement, especially when the order of the $GF(q)$ is large. In order to fix the memory problem, the proposition of keeping only the $n_m < q$ most reliable messages can be used. As it can be observed from Fig. 4.3, keeping only $n_m = 16$ messages from the $GF(2^5)$ elements leads to around 0.05dB performance loss in the Min-max algorithm and 0.04 dB performance loss in the EMS algorithm. Figure 4.3 also show the performance of a binary (3683, 3175) LDPC code using the Min-sum decoding. This binary code has similar code rate and code word length in terms of bits as the (744, 653) NB-LDPC code over $GF(2^5)$, As observed the performance degradation at BER=10^{-5} of Binary code compared to the Non Binary code is around 0.285 dB.

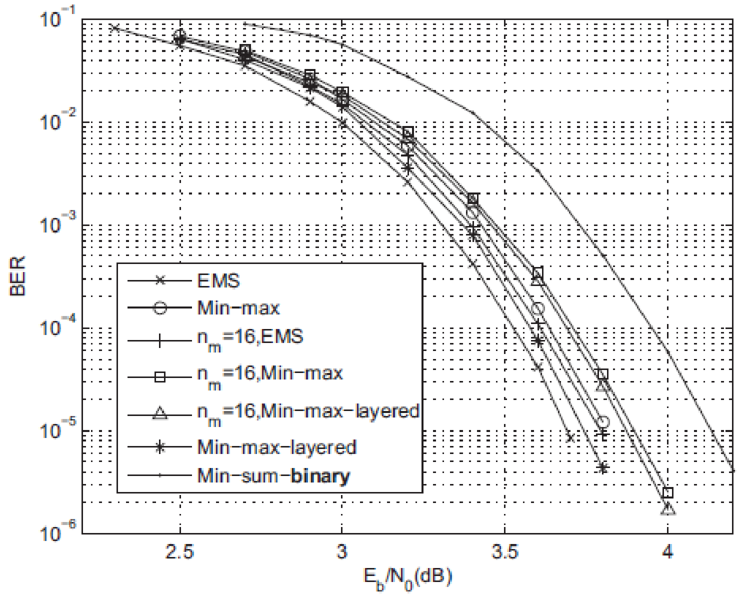


Figure 4.4: BERs of a (248, 124) NB-LDPC code over GF(25) and a (1270,635) binary LDPC code under AWGN channel [3]

Figure 4.4 illustrate the performance of a (248, 124) code over $GF(2^5)$ for EMS and Min-max decoding algorithms, also the performance of the Min-sum decoding for a (1270, 635) binary LDPC code. This code has a shorter length and lower rate than the (744, 653) code. As it can be observed, using the NB-LDPC code lead to benefit more coding gain than the Binary code .

Chapter 5

Chapter 5: NB-LDPC Decoder Architecture and Implementation

In this chapter, we will discuss the design of the decoder on hardware 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. Our design keeps all q messages on each edge of the Tanner graph. An optimized scheme and corresponding architecture are developed to compute the minimum of 'max' for the elementary step of the check node processing. There are two basic design strategies, the layered and the non-layered one. The non-layered one aims at higher throughput, with considerably higher area, while the layered one has lower throughput, with lower area requirement. Moreover, the computation units and the scheduling of the computations are optimized to reduce the area and decoding latency. Also a forward-backward scheme is proposed for the CN processing. Employing this scheme, the speed of CN processing can be almost doubled when the check node degree is not small. Our architecture is applied to a NB-LDPC code constructed over $GF(2^3)$ as an example. It has also been synthesized on a Xilinx Virtex-5 FPGA ML501 device.

5.1 High level architecture

5.1.1 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 [22] which is a direct mapping of the tanner graph.

The global architecture of decoder as illustrated in Figure 5.1 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 h_{ij}/h_{ij}^{-1} 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.

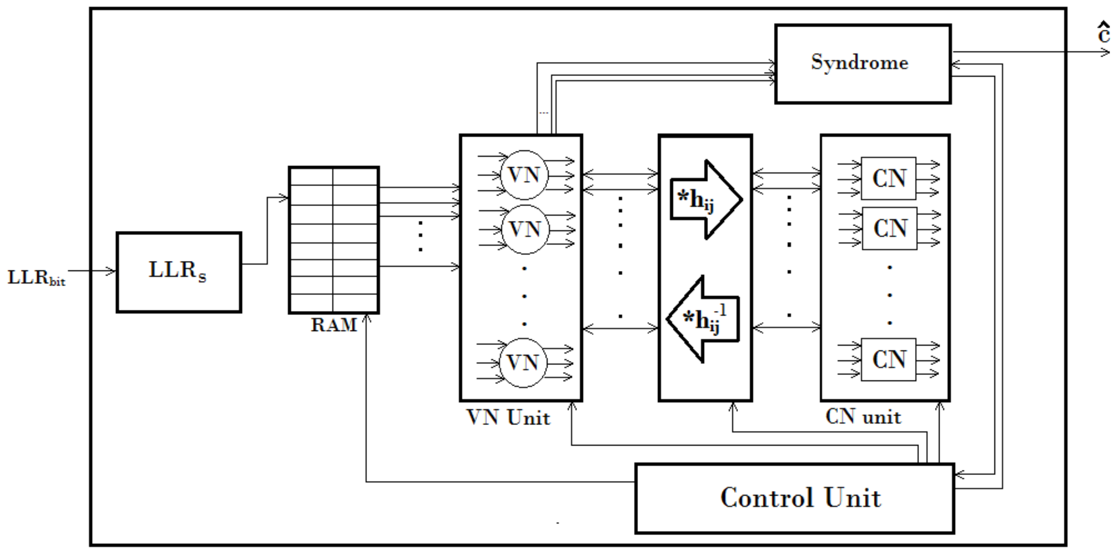


Figure 5.1: The top level NB-LDPC Decoder architecture

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.

5.1.2 Layered and Non-Layered architecture

The design can be partitioned into two basic architecture classes : the non-layered (*Full-parallel*) and the layered architecture [23]. The last one also can be divided into two strategies *row parallel*, and *block-parallel*. These architecture classes are depicted in Figure 5.2.

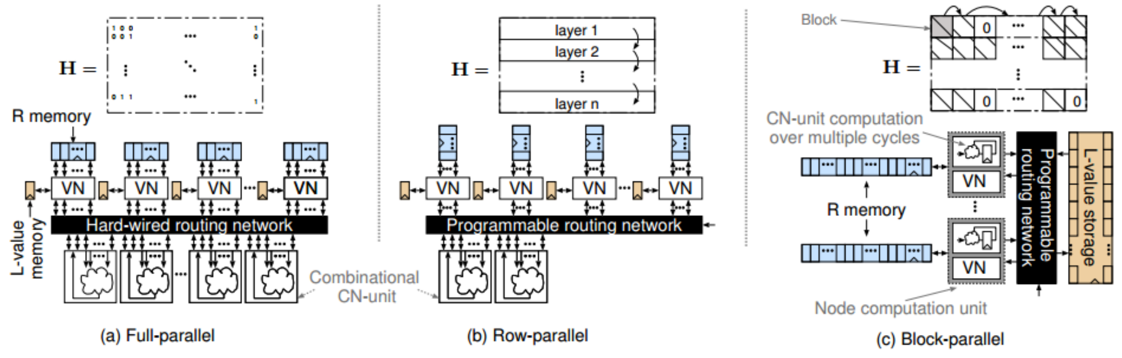


Figure 5.2: Layred and Non-layred Decoder architecture

A high-level block-diagram of a **non-layered** (full-parallel) design is shown in Figure 5.2(a). The update equations are mapped into individual VNUs and CNUs that exchange messages through a *hard-wired routing network* [2]. The parallel processing allows each iteration to be performed in a fewer number of clock cycles, by updating all VN units then all CN units. This architecture enables very high throughput since one iteration is performed per fewer number of clock cycles with simple computations that allow for high operating clock frequencies. Unfortunately, the complex routing network that connects CNUs and VNUs turns out to be a major implementation bottleneck for these designs.

Full-parallel architectures have been considered mainly for wire-line communication standards (e.g. 10GBASE-T). Since the full-parallel architecture represents the direct hardware mapping of a specific parity-check matrix, this class cannot provide any flexibility. Therefore, it is generally ill-suited for wireless communication standards that require the support for different parity-check matrices in order to tune code rate and block length.

Layered decoding has been widely adopted to reduce the memory requirement and increase the convergence speed of LDPC decoding.

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 complexity and the amount of bits required to control this programmable routing network heavily depend on the structure of the code and on its partitioning into layers.

The row-parallel architectures provide an area advantage over full-parallel designs. Note that additional storage to hold the LLR values computed in the previous iteration, while processing layers of the current iteration, can be avoided for a layered schedule with proper layer selection. In general, 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 designs rely on further resource sharing and further iterative decomposition. Figure 5.2(c) outlines the architectural principle, which is usually used in combination with the layered message-passing schedule. In essence, 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. The block-based processing in conjunction with structured codes significantly facilitates reconfigurability. Due to this reason, this architecture class has been widely employed for flexible decoder implementations.

5.2 System level architecture

5.2.1 Forward-Backward Based Architecture

Computing the *c-to-v* messages in a straight-forward manner requires a complexity of $O(q^{w_r-1})$, where w_r is the number of variable nodes connected to a check node. 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 not to repeat the same computations and to reduce the processing latency. These elementary operations are performed by Elementary Check Node (ECN). Each ECN receives two sorted messages M_1 and M_2 and generates a sorted message M_o. The message M_o is constructed by selecting the q most reliable symbols from all the possible combinations (as explained in min-max CN updating operation).

Figure 5.3 illustrates the Forward-Backward architecture of CN with degree $w_r = 4$. For clarity, the incoming CN messages are denoted by M_{v_jc} and the outgoing messages are denoted by M_{cv_j}, $j = 1, ..., 4$.

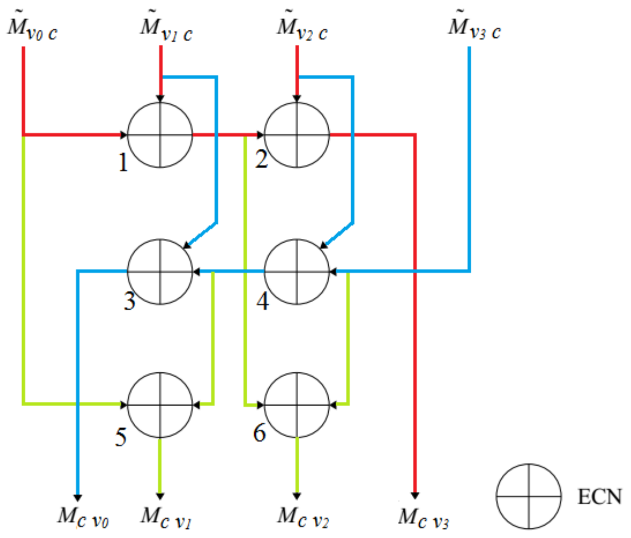


Figure 5.3: Forward-Backward Check-node architecture

The CN of degree w_r realization requires the implantation of $3(w_r - 2)$ ECNs distributed over 3 layers. In our case, the outgoing M_{cv_3} message is generated by the first layer of ECNs that adds incoming messages in the Forward direction (the red). The outgoing message M_{cv_0} is generated by the second layer of ECNs which adds incoming messages in the backward direction (the blue). The other outgoing messages are generated by the third layer which combines the two incoming messages M_{v_0c} and M_{v_3c} with the intermediate messages of the two upper layers (the green).

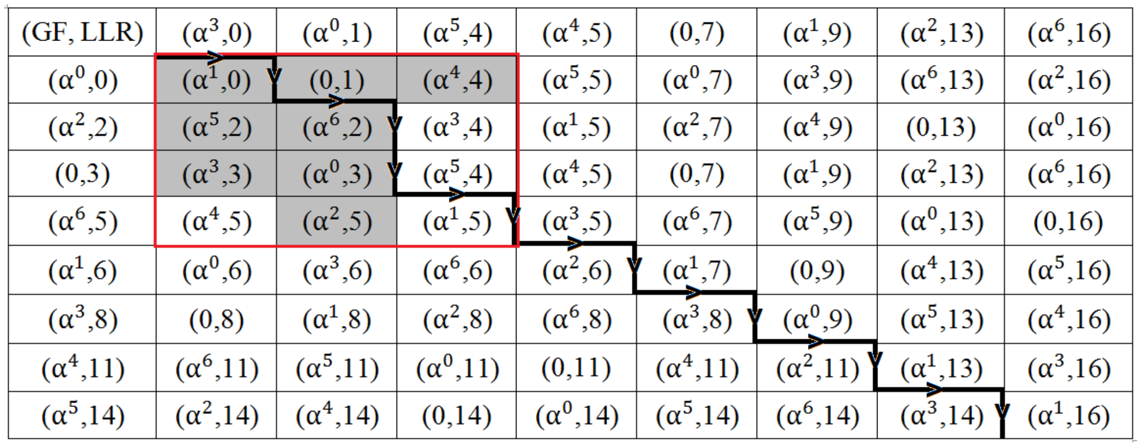
5.2.2 Check Node architecture

The CN receives w_r messages and generates also w_r messages which will be transmitted to the VNs. The CN architecture uses the Forward-Backward scheme illustrated in previous section, It consists of $3(w_r - 2)$ ECNs. in our project we are constructing CN with degree $w_r = 4$ that means we need to 6 ECNs. Figure 5.3 illustrate the scheme construction. We are also proposing an CN architecture that can be used in both cases, keeping all q messages in each vector or keeping only the $n_m < q$ most reliable messages in each vector. this 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 $n_m < q$ 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 $L_A = [L_A(0), L_A(1), ..., L_A(q-1)]$ and $L_B = [L_B(0), L_B(1), ..., L_B(q-1)]$, and the corresponding finite field element vectors by $GF_A = [GF_A(0), GF_A(1), ..., GF_A(q-1)]$ and $GF_B = [GF_B(0), GF_B(1), ..., GF_B(q-1)]$, also denote the output LLR and corresponding finite field element vectors by L_O and GF_O. The entries in the output LLR vector for the Min-max decoding are the q minimum values of $\max(L_A(i), L_B(j))$ with different $GF_A(i)+GF_B(j)$ for any combination of i and j less than q.

The traditional solution of computing the outgoing message consists on comparing q^2 pairs of messages from the two input vectors to find the ones with larger LLRs, then find the q minimums among them. Taking care of all these operations in parallel is hardware-demanding, also performing them serially make a latency problem. But the ECN developed in our project uses two incoming messages stored in the order of increasing LLR and generate sorted outgoing message of the most reliable messages, by using an efficient algorithm to find the q most reliable LLRs in minimum clock cycles using serial computation to reduce the hardware area.

Two example input vectors for a NB-LDPC code over $GF(8)$ are considered in the table in Figure 5.4. The finite field elements are denoted by the exponents in their power representations. As illustrated in the table the most reliable q symbols are essentially distributed in the top left corner (red rectangle), the grey cells are the entries of the output message vector.



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

Figure 5.4: Example of ECN computation with two sorted inputs

We can observe from the table in Figure 5.4 that there is a pattern for LLR values distribution , based on this pattern we can make the desired algorithm for the ECN computation .

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 in Figure 5.4 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 $L_A < L_B$ vector, and goes to the right otherwise. As observed , the LLRs above the horizontal segment of the boundary in the same column are the same, and they are smaller than those below the segment. Similarly, for the LLRs in the same row to the left and the right of the vertical segment. The testing result of whether $L_A(i) < L_B(j)$ 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.

ECN architecture

The previous algorithm can be implemented by the architecture illustrated in Figure 5.5. Since the minimum of 'max' LLRs need to be kept, the comparisons of the LLRs start from the first entries in the two input vectors. In the case that $L_A(i) < L_B(j)$ happens, $L_A(i)$ was the 'max' value when compared with any previous $L_B(k)$ with $0 \leq k < j$, since the LLRs in each vector are sorted. Hence, $L_A(i)$ should be inserted into the output vector together with $GF_A(i) + GF_B(k)$ for each $0 \leq k < j$. After this process is completed, i will increment so the next entry for the L_A vector is read out in the next clock cycle to be compared with $L_B(j)$. Similarly, in the case that $L_A(i) > L_B(j)$, $L_B(j)$ will be inserted into the output vector with field element $GF_A(k) + GF_B(j)$ for each $0 \leq k < i$.

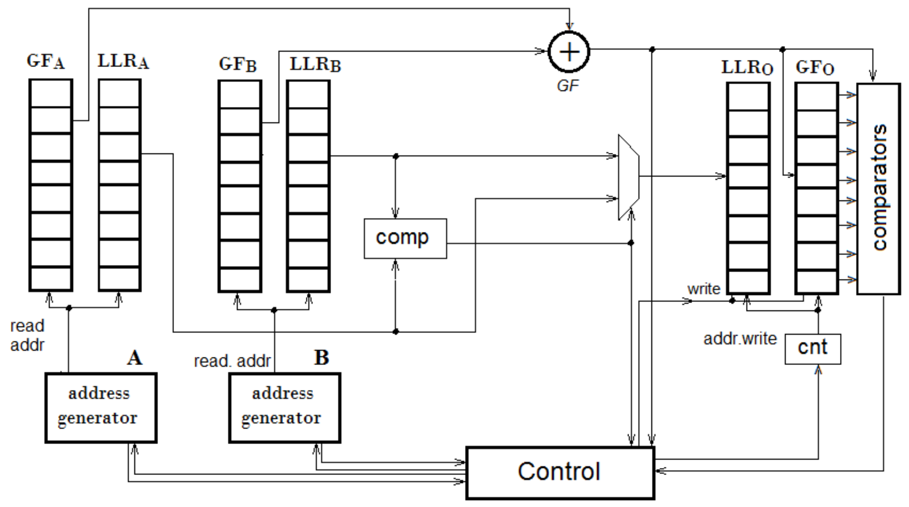


Figure 5.5: ECN architecture

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. The increase by one is pre-computed and the addresses are selected by multiplexor in the address generators. Figure 5.6.

If $L_A(i) < L_B(j)$, 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, $GF_B(k)$ is read out and added up with $GF_A(i)$ until the counter reaches $j - 1$. In the case that $L_A(i) > L_B(j)$, 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.

5.2.3 Variable Node architecture

The VNs are divided into two categories according to whether all q messages are kept for each vector or not. In our project we will design a VN with degree $w_c = 2$, that 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. It contains several Elementary Blocks that can make it capable to operates on three basic functions :

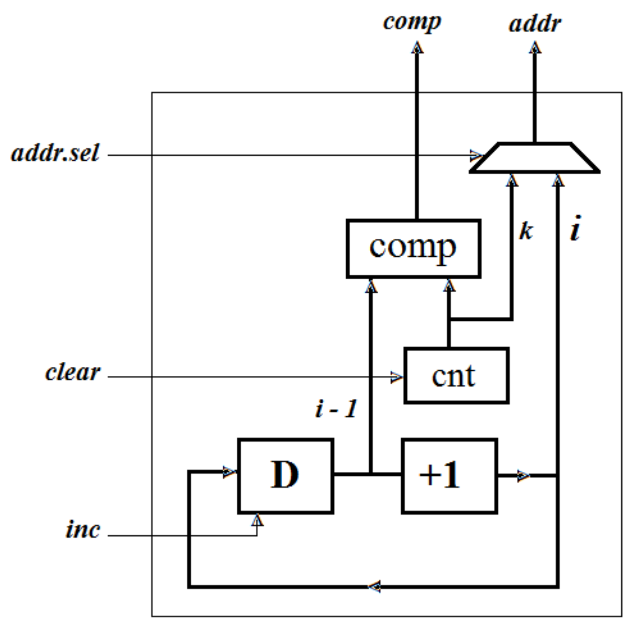


Figure 5.6: Address generator block

- Updating the VN: This operation is done by the EVN block, The EVN receives two incoming message from the CNs and generate an new outgoing message to the normalization block .
- Normalization : This operation is done by the Norm block, It 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: This operation is done by the Decision block. Firstly it calculates the APP by the sum of all VN input message vectors, then determine the estimated symbol $\hat{c}$ by taking the GF element corresponding to the minimum LLR value.

EVN Architecture

The goal of an EVN is to compute the outgoing message vector stored in the order of increasing LLR 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.

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 GF_A vector entry is compared with all those in the GF_B 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.

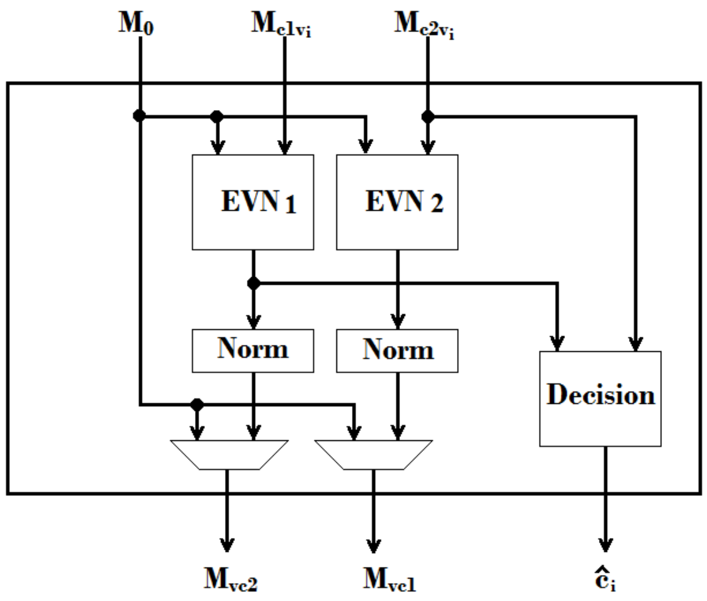


Figure 5.7: Variable Node Architecture

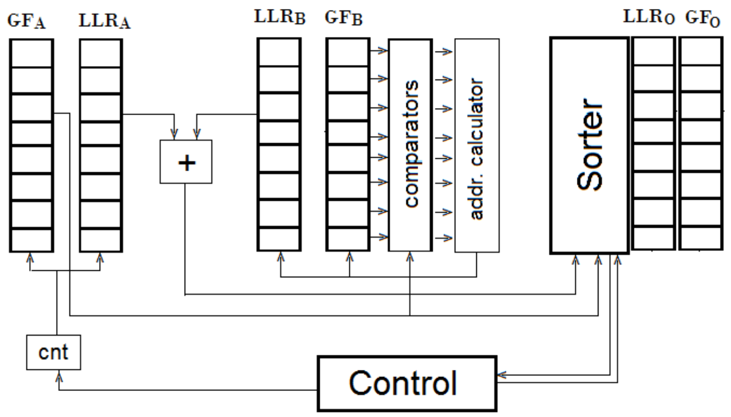


Figure 5.8: Elementary Variable Node Architecture

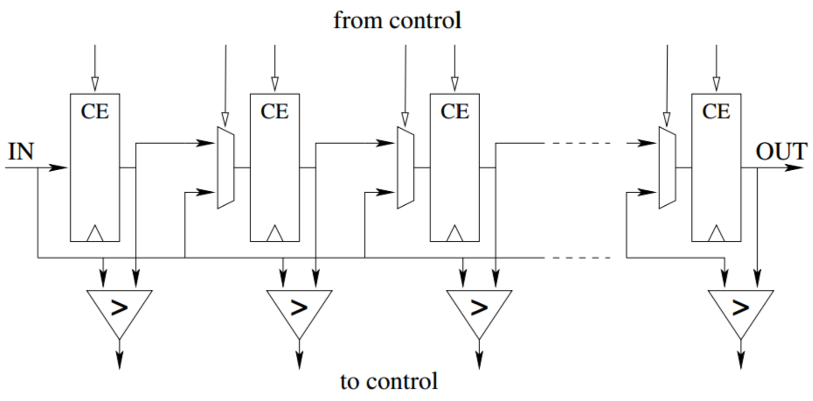


Figure 5.9: Sorter architecture

An architecture for such a sorter was proposed in [24] shown in Figure 5.9. 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.

Decision block

The decision operation is based on the APP calculation (explained in chapter 4) which is the sum of all VN input vectors (M_0, Mc_1v and M_{c_2v}), as a result we can use the output of EVN1 and M_{c_2v} as inputs of the Decision block, which is a modified EVN, that modification consist of taking as output the GF element corresponding to the minimum LLR value, which is the first element since the EVN integrate a sorter.

5.3 Implementation and simulation

In this section we will discuss the FPGA implementation of an efficient decoder architecture based on the Min-max algorithm for NB-LDPC codes, this architecture provides implementation flexibility and a compromise between decoding speed and implementation complexity. The architecture blocks are explained in details such as the CN and VN, they are implemented on the Xilinx ISE platform using the VHDL description language. After the synthesis of those codes, the schematics of the architectures from the top level description to the unit description are generated and illustrated in this section. Similarly, the blocks are checked and tested via the test-bench tool, and the results are also presented. Finally a summery consumption reports are presented and discussed at the end of each block implementation section.

5.3.1 Variable Node Implementation

In what follow, we will work on $GF(8)$, as a result the GF and LLR vectors contains 8 elements, the GF elements presented in 3 bits and the LLR values in 10 bits.

Elementary Variable Node block

Figure 5.10 illustrates the EVN block schematic generated in Xilinx ISE.

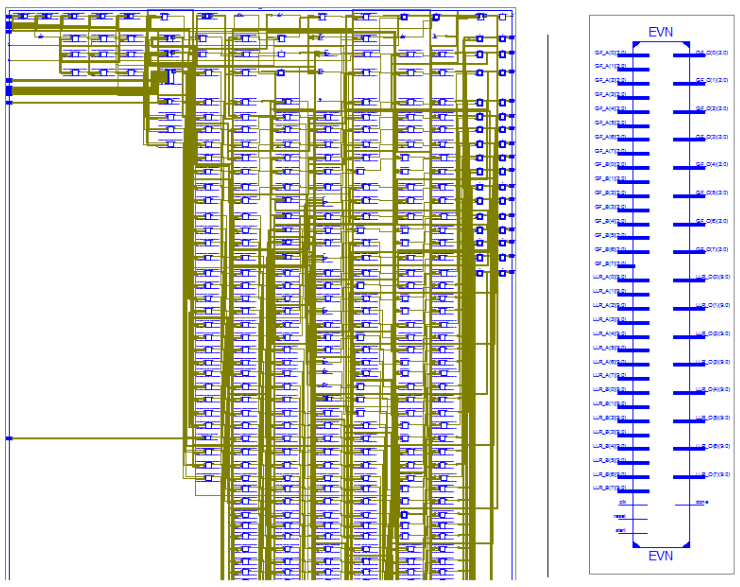


Figure 5.10: EVN schematics generated by Xilinx ISE

As illustrated in the figure the EVN block schematic has 3 type of inputs :

- Synchronization : which is the *clk* signal .
- Control signals : which are *reset* and *start* signals, these signals are generated by the global control unit of the decoder. *Start* signal indicates the input data availability to start the EVN process.
- Data messages : which are the incoming messages, there are 4 input vectors for the two incoming messages, GF_A, LLR_A, GF_B and LLR_B each one has 8 element.

The outputs of the block schematic : output message that contains two vectors LLR_O and GF_O, the control signal *done* to indicate the end of EVN process.

The EVN control unit implements a 4 stage state machine, generating all the internal control signal that controls its function. The state transition diagram is shown in figure 5.11.

INIT State : The EVN starts in this state, all the internal control signals initialized and

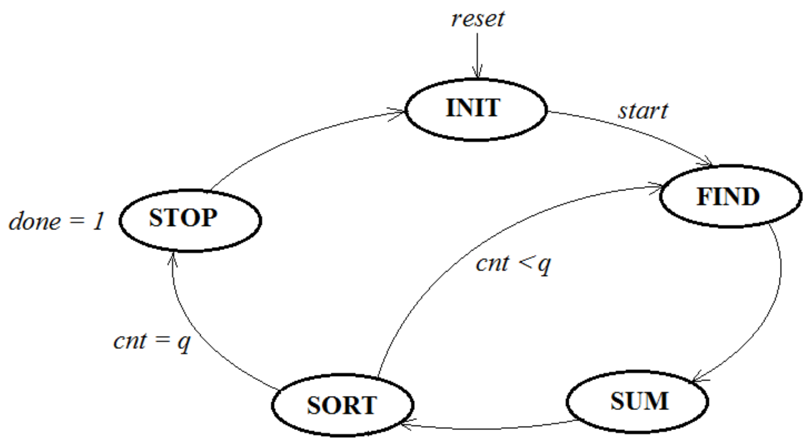


Figure 5.11: EVN state machine

remains here until a new data are available. When the start signal is high , a new data are available, the EVN load those data in the internal RAMs (RAM_A and RAM_B), and move to the new state "Find".

FIND State: Reading the GF element from the RAM_A, find the address of the same GF element in RAM_B, and move to the new state "SUM".

SUM State: Reading the LLR values corresponding to the GF element and add them . then move the new state "SORT".

SORT State : Compare the sum result LLR in the sorter to find its address (position) ,then load it and its corresponding GF on RAM_O. If all element are loaded next state is "STOP", else next state "FIND".

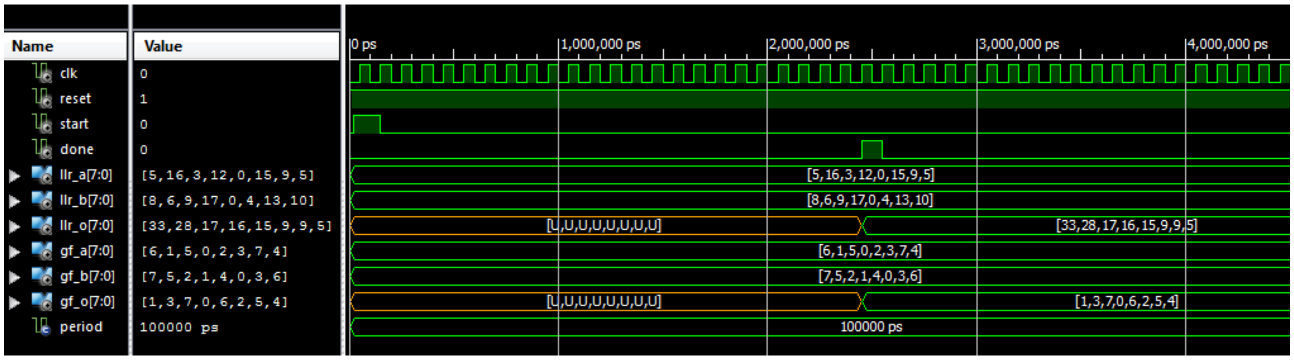
STOP State : In this state all q elements are loaded in RAM_O, the EVN complete its computations, make the done signal high and next state will be "INIT".

This state machine diagram spends 3 clock cycles to compute one element of the output vector, for all the 8 elements it takes 24 clock cycles. Another state diagram is proposed in order to reduce this processing cost using pipeline technique on the elementary operations so they can be working in parallel.

EVN Simulation Results

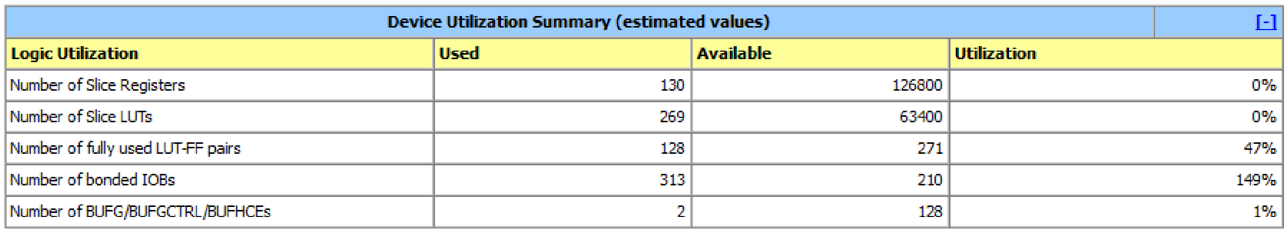
Using the Xilinx ISE platform and a test bench module, The behavioral simulation results for the EVN are shown in Figure 5.12 . The input data has the same values as the example illustrated in table 4.2 (section 4.5).

Since the results are the same obtained in the example in table 4.2 (chapter 4), the simulation result prove the functionality of the EVN block, as the figure shows, the EVN complete the computations after 24 clock cycles.



Name	Value
clk	0
reset	1
start	0
done	0
llr_a[7:0]	[5,16,3,12,0,15,9,5]
llr_b[7:0]	[8,6,9,17,0,4,13,10]
llr_o[7:0]	[33,28,17,16,15,9,9,5]
gf_a[7:0]	[6,1,5,0,2,3,7,4]
gf_b[7:0]	[7,5,2,1,4,0,3,6]
gf_o[7:0]	[1,3,7,0,6,2,5,4]
period	100000 ps

Figure 5.12: EVN simulation results using testbench in Xilinx ISE

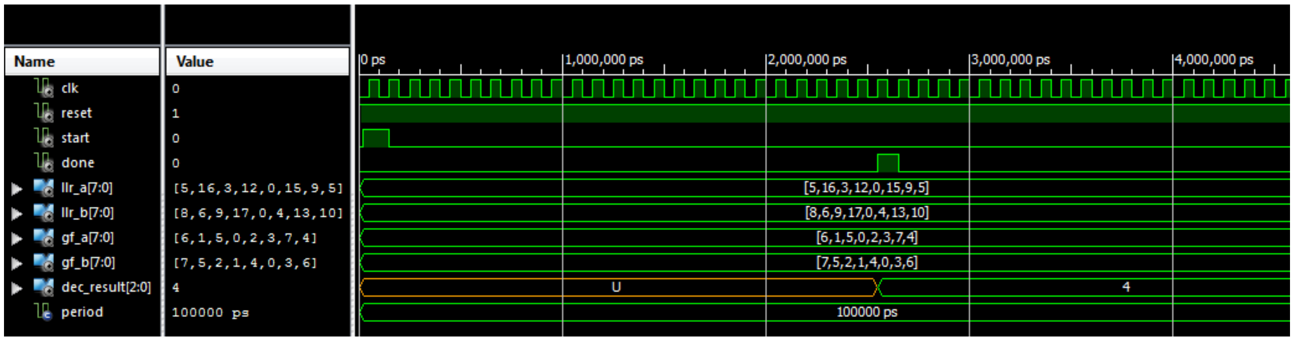


Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	130	126800	0%
Number of Slice LUTs	269	63400	0%
Number of fully used LUT-FF pairs	128	271	47%
Number of bonded IOBs	313	210	149%
Number of BUFG/BUFGCTRL/BUFHCEs	2	128	1%

Figure 5.13: EVN Utilization summary estimated by Xilinx ISE

Decision Unit block simulation

Using the Xilinx ISE platform and a test bench module, The behavioral simulation results for the Decision block are shown in Figure 5.14 . The input data has the same values as the example illustrated in table 4.2 (Chapter 4).



Name	Value
clk	0
reset	1
start	0
done	0
llr_a[7:0]	[5,16,3,12,0,15,9,5]
llr_b[7:0]	[8,6,9,17,0,4,13,10]
gf_a[7:0]	[6,1,5,0,2,3,7,4]
gf_b[7:0]	[7,5,2,1,4,0,3,6]
dec_result[2:0]	4
period	100000 ps

Figure 5.14: Decision Unit simulation result in Xilinx ISE

As shown in Figure 5.14 the resulting estimated GF element is '4' (α^2 in power representation) , which is the same result given in table 4.2 (Chapter 4).This process takes 25 clock cycles.

VN block

Figure 5.15 shows the block schematic of the VN block as generated in Xilinx ISE .

The VN block schematic has 3 data message inputs (each one has 2 vectors), synchronization input (clk) and control signals reset, start to indicate new available data then starting computations, and initial to indicate if it is the first iteration or not. The block schematic

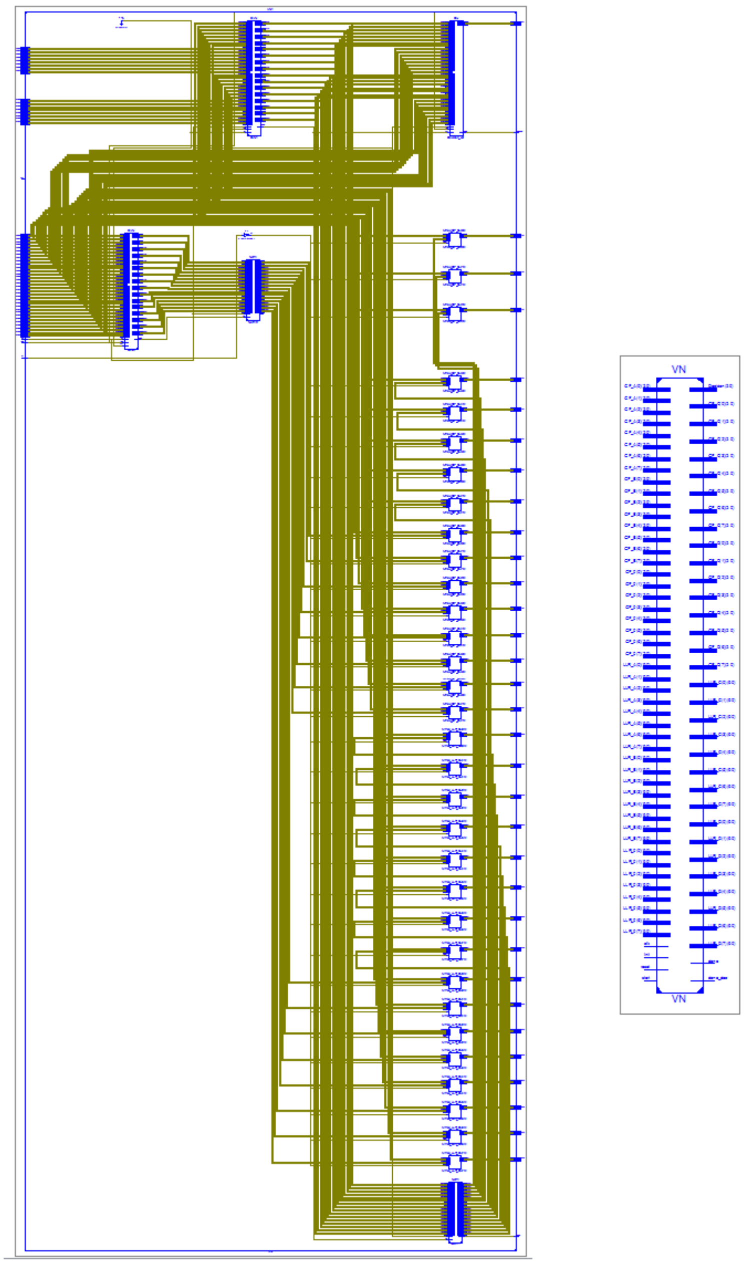


Figure 5.15: VN schematic as generated by Xilinx ISE

has as outputs two data messages (each one has 2 vectors), Decision message contain the estimated symbol and two control signals *done* and *done_decision* .

The VN operates on three basic functions update the v-to-c messages, normalizes the messages and compute the decision symbol. each function has a specific block. Updating the massages take 24 clock cycles and the normalization take 8 clock cycles so compute the VN output messages takes 32 clock cycles. The Decision operations will begin after the EVN computations is done and take 24 clock cycles, by adding the time spent in the two blocks the compute of the decision output takes 48 clock cycles. The VN output data and the decision output message are independent because the first ones will be sent to CNs and the second will be sent to the Syndrome unit.

5.3.2 Check Node Implementation

ECN block

Figure 5.16 illustrates the block schematic of the ECN block as generated in Xilinx ISE.

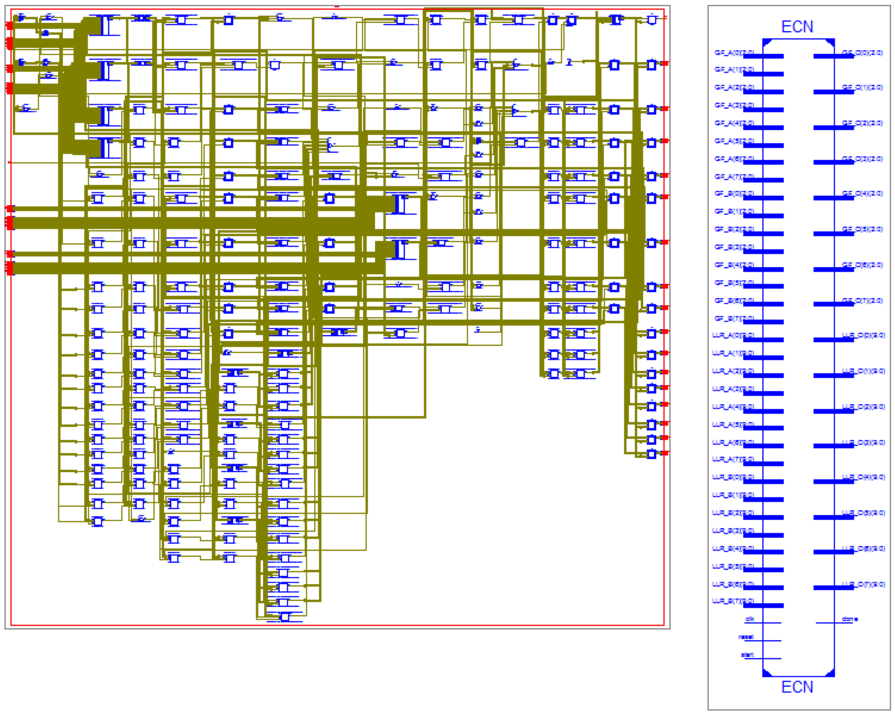


Figure 5.16: ECN schematic block generated using Xilinx ISE

The figure shows that ECN block schematic has the following inputs :

- *Synchronization* : which is the clk signal .
- *Control signals* : the reset and start signals. Start signal indicates the input data availability to start the ECN process.
- *Data messages* : There are 4 input data vectors for the two incoming messages, GF_A, LLR_A, GF_B and LLR_B each one has 8 element.

The outputs of the block schematic are: the output message contains tow vectors LLR_O and GF_O, the second one is the control signal *done*.

The ECN control unit state transition diagram is shown in figure 5.17.

START State : The ECN starts in this state, all the internal control signals initialized and remains here until a new data are available. When the start signal is high i.e. a new data are available, the ECN load those data in the internal RAMs (RAM_A and RAM_B), and move to the new state "COMP".

COMP State : Read then compare two LLR values of address i and j from RAM_A and RAM_B the comparison result determines the next state whether "SUM1" or "SUM2". initialize the k counter.

"SUM1" State (respectively "SUM2"): Perform the addition of GF elements corresponding to the addresses i in RAM_A and k in RAM_B (respectively k in RAM_A and j in

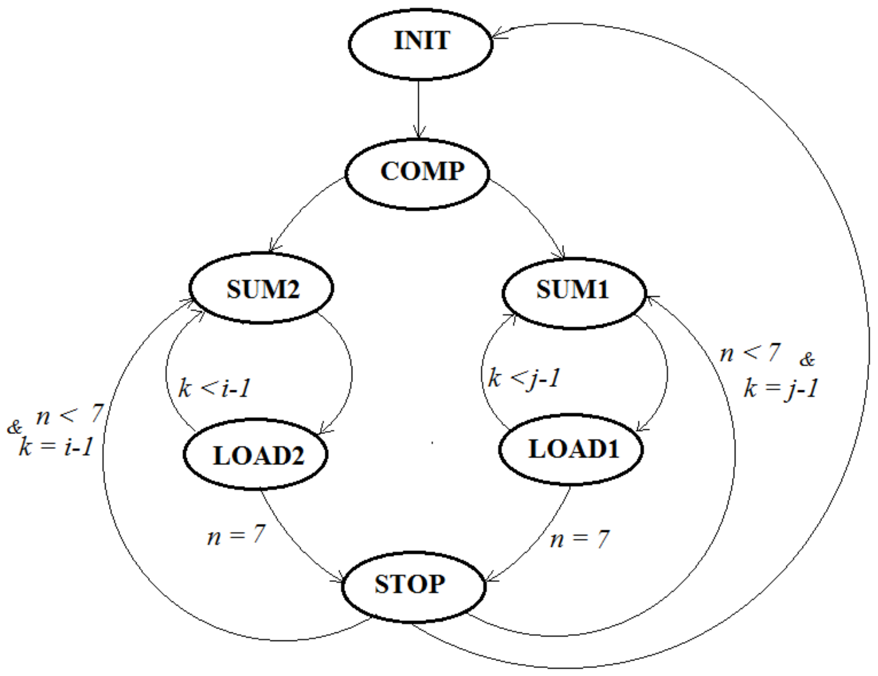


Figure 5.17: ECN state machine

RAM_B). The next state is "LOAD1" (respectively "LOAD2").
"LOAD1" State (respectively "LOAD2"): At first check whether the result GF element has been loaded before or not, if not load it and its corresponding LLR on RAM_O in the address n then check whether n reach 7 or not, if it did next state is "STOP", else increment n. Another test will be performed, check whether the k achieves j-1(respectively i-1) or not, if it did increment i (respectively j) and next state is " COMP ", else increment k and next state is "SUM1" (respectively "SUM2").
"STOP" State : In this state all q elements are loaded in RAM_O, the ECN complete its computation, make the done signal high and next state will be "START".

ECN Simulation Results

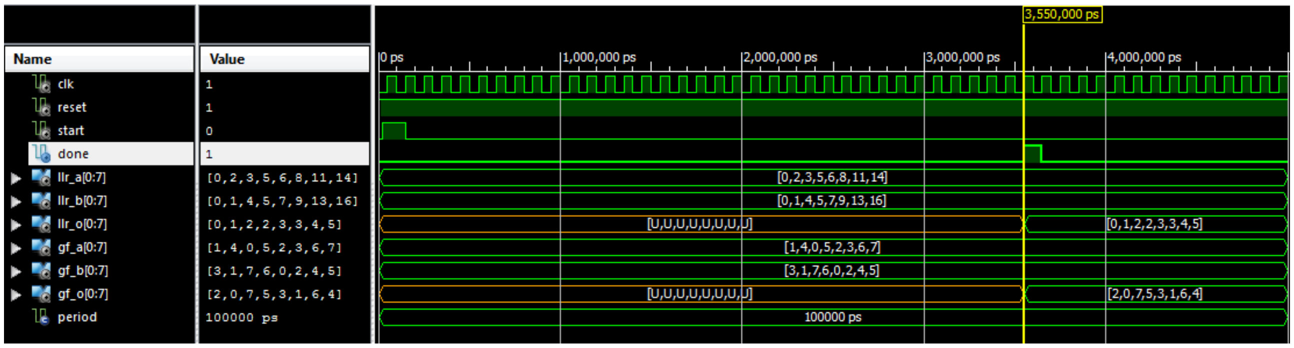
Using the Xilinx ISE platform and a test bench module, The behavioral simulation results for the ECN are shown in Figure 5.18. The input data has the same values as the table in figure 5.4 (chapter 5).

The obtained results are the same as the example table results (Figure 5.4), they prove the functionality of the ECN block, as the figure show, the ECN complete the computations after 35 clock cycles, which is not constant, since the iteration number of ECN algorithm changes by changing the input messages values.

CN block

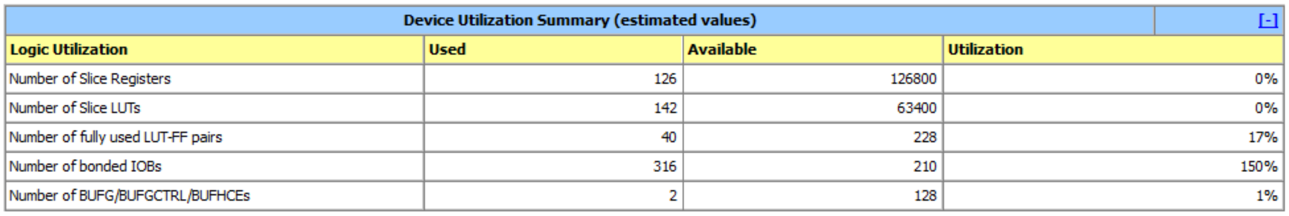
Figure 5.20 shows the block schematic of Forward-Backward CN block as generated in Xilinx ISE .

The CN architecture based on Forward-Backward scheme consists of 6 ECNs, At the beginning only the ECNs connected directly to the input data work (ECN number 1 and 4



Name	Value
clk	1
reset	1
start	0
done	1
llr_a[0:7]	[0,2,3,5,6,8,11,14]
llr_b[0:7]	[0,1,4,5,7,9,13,16]
llr_o[0:7]	[0,1,2,2,3,3,4,5]
gf_a[0:7]	[1,4,0,5,2,3,6,7]
gf_b[0:7]	[3,1,7,6,0,2,4,5]
gf_o[0:7]	[2,0,7,5,3,1,6,4]
period	100000 ps

Figure 5.18: ECN testbench results on Xilinx ISE



Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	126	126800	0%
Number of Slice LUTs	142	63400	0%
Number of fully used LUT-FF pairs	40	228	17%
Number of bonded IOBs	316	210	150%
Number of BUFG/BUFGCTRL/BUFHCEs	2	128	1%

Figure 5.19: ECN Utilization summary estimated by Xilinx ISE

in the Figure 5.3). After they complete their computation the other ECNs (2,3,5 and 6) can begin processing.

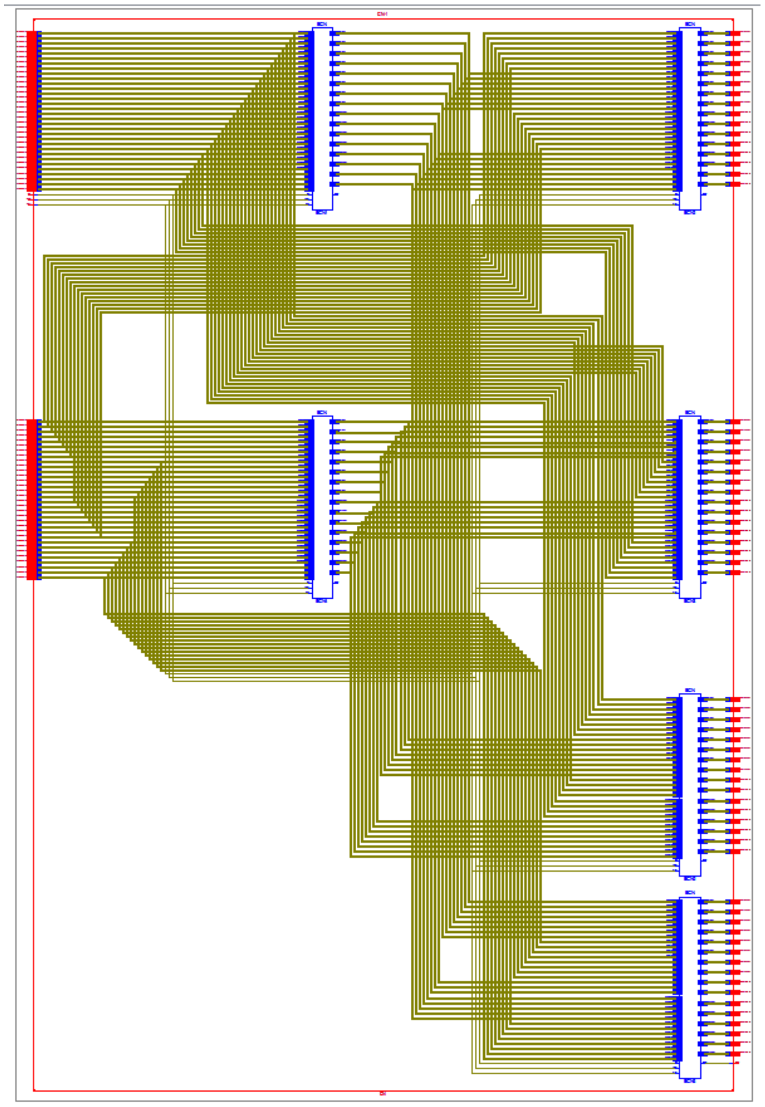


Figure 5.20: Backward-Forward CN block schematic as generated on Xilinx ISE

Conclusion

The project 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 others 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 the design of NB-LDPC decoders can be partitioned into two main architecture. The non-layered architecture achieve very high throughput in excess of 10 Gbps but cannot provide any flexibility in terms of code rate or block length. The layered architecture which consist of row-parallel class and block-parallel class designs reduce the area compared to the non-layered architecture with different degrees of resource sharing and iterative decomposition at the expense of a degradation in throughput. While both architecture classes offer flexibility, the block-parallel class is especially suitable for QC-LDPC codes as it naturally fits together with the block structure of these codes.

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 any given code, without changing the design of the decoder. Moreover, it can be adopted to reduce the memory resources.

In our project we have focused on the design and implementation of an efficient architecture for the NB-LDPC decoder basic blocks, since wireless communication systems decoders must support a wide range of different parity-check matrices,and because of that we provide flexible decoder which can works with different block lengths and code rates.

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 $n_m << q$ most reliable elements. We have also designed the variable node block using elementary blocks for the same reason.

The decoder components were implemented on a Xilinx ISE platform using VHDL language and tested using test benches. All the results and reports have been documented. Various details like block schematics, state machine diagrams, simulation and resource utilization have been documented.

Finally it was a great opportunity to us, to learn more about channel coding field and specially the robust NB-LDPC code and its hardware implementation.

Bibliography

[1] U. Vilaipornsawai M. Reza Soleymani, Yingzi Gao. *Turbo Coding for Satellite and Wireless Communications.* The Springer International Series in Engineering and Computer Science. Springer, 1 edition, 2002.

[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] Claude Berrou. *Codes and turbo codes.* Collection IRIS. Springer, 1st edition. edition, 2010.

[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-Interscience, 2007.

[8] Mylène Pischella Didier Le Ruyet. *Digital Communications 1: Source and Channel Coding.* Networks and Telecommunications. Wiley-ISTE, 1 edition, 2015.

[9] Masoud Salehi John Proakis. *Digital Communications, 5th Edition.* McGraw-Hill Science Engineering Math, 5th edition, 2007.

[10] Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.

[11] Alexandru Spataru. *Fondements de la théorie de la transmission de l'information.* PPUR presses polytechniques, 1987.

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

[13] Vijay K. Bhargava Stephen B. Wicker. *Reed-Solomon Codes and Their Applications.* 1 edition, 1999.

[14] Peter Elias. *Error-free coding.* Technical report (Massachusetts Institute of Technology. Research Laboratory of Electronics) ; 285. Massachusetts Institute of Technology. Research Laboratory of Electronics, 1954.

[15] C. Berrou, A. Glavieux, and P. Thitimajshima. Near shannon limit error-correcting coding and decoding: Turbo-codes. 1. In *Communications, 1993. ICC '93 Geneva. Technical Program, Conference Record, IEEE International Conference on*, volume 2, pages 1064–1070 vol.2, May 1993.

[16] Jean-Baptiste Doré. *Optimisation conjointe de codes LDPC et de leurs architectures de décodage et mise en œuvre sur FPGA*. PhD thesis, INSA de Rennes, 2007.

[17] R. Tanner. A recursive approach to low complexity codes. *IEEE Transactions on Information Theory*, 27(5):533–547, Sep 1981.

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

[19] C. Qian, W. Lei, and Z. Wang. Low complexity ldpc decoder with modified sum-product algorithm. *Tsinghua Science and Technology*, 18(1):57–61, Feb 2013.

[20] 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*, 58(11):3126–3139, 2010.

[21] Shu Lin, Shumei Song, Bo Zhou, Jingyu Kang, Ying Y Tai, and Qin Huang. Algebraic constructions of nonbinary quasi-cyclic ldpc codes: Array masking and dispersion. In *9th International Symposium on Communication Theory and Applications (ISCTA)*, 2007.

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

[23] M. M. Mansour and N. R. Shanbhag. A 640-mb/s 2048-bit programmable ldpc decoder chip. *IEEE Journal of Solid-State Circuits*, 41(3):684–698, March 2006.

[24] 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*, pages 1–2, Jan 2008.