**UNIVERSITE DE BLIDA 1**

**Faculté de Technologie**

Département d'Electronique

# THÈSE DE DOCTORAT

en Génie électrique

## DÉCODAGE LDPC STOCHASTIQUE SUR FPGA

Par

**Ghania ZERARI**

Devant le jury composé de :

| | | |
|---|---|---|
| A. BENALLAL | Professeur à l'Université de Blida | Président |
| A. MERAGHNI | Professeur à l'ENS-Kouba | Examinateur |
| B. FERGANI | Professeur à l'USTHB | Examinateur |
| N. BELKHAMSA | Maître de Conférences à l'Université de Blida | Examinateur |
| A. GUESSOUM | Professeur à l'Université de Blida | Directeur de Thèse |

Blida, 2018

# RESUME

Cette thèse a pour but principal l'étude des architectures de décodage LDPC stochastique existant et de développer une nouvelle approche moins complexe et plus performante implémentable sur FPGA.

Dans le contexte des décodeurs LDPC à complexité réduite non stochastique, nous avons réalisé une nouvelle étude architecturale et nous avons développé une nouvelle variante de décodage LDPC basé sur l' "Offset MIN-SUM algorithm" (OMS). L'architecture proposée offre une réduction de complexité supplémentaire et une diminution des temps de réponse, tout en conservant les performances du décodage.

De même pour l'approche stochastique, les récentes techniques du décodage LDPC et leurs complexités sont analysées sur le plan implémentation matérielle. Le but de cette nouvelle analyse est de déduire le rapport entre la performance et le taux d'utilisation des ressources logiques sur FPGA. Ainsi, nous avons conçu une nouvelle architecture de décodage LDPC stochastique complètement parallèle, qui permette la réduction de l'utilisation de la logique FPGA et l'amélioration de la convergence. La validation sur FPGA de Xilinx, de notre technique stochastique, a confirmé une réduction supplémentaire d'utilisation de la logique FPGA et une amélioration de la convergence, même pour les codes courts.

**Mots clés:** Décodage LDPC "Low-Density Parity-Check", Décodeur LDPC stochastique, FPGA "Field-Programmable Gate Array".

# ABSTRACT

The main target of this thesis is to study the existing stochastic LDPC decoding architecture and to develop a new efficient approach, with less complexity which can be implemented in FPGA devices.

For the non-stochastic reduced complexity LDPC decoders, we performed a new architectural study and we developed a new LDPC decoding variant based on the "Offset MIN-SUM algorithm" (OMS). The proposed architecture offers an additional complexity reduction and a latency decreasing, while maintaining the decoding performance.

For the stochastic approach, the recent techniques of decoding LDPC codes are treated with a new method to get the relationship between performance and FPGA logic utilization. Hence, we developed a new design of fully parallel stochastic LDPC decoding architecture, which allows the reduction of the FPGA logic utilization and the improvement of the convergence. The Xilinx FPGA validation of our stochastic technique confirmed a further reduction in FPGA logic utilization and improved the convergence, even for short codes.



**Keywords:** "Low-Density Parity-Check" LDPC codes, Stochastic LDPC Decoding, FPGA "Field-Programmable Gate Array".

# ملخص

الغرض الرئيسي من هذه الرسالة هو دراسة أنظمة فك الترميز LDPC بواسطة الطرق العشوائية (ستوكاستيك) الحالية وتطوير نهج فعال جديد، أقل تعقيدا و التي يمكن تنفيذها بواسطة FPGA.

بالنسبة إلى أجهزة فك الترميز LDPC قليلة التعقيد الغير عشوائية، قمنا بإجراء دراسة جديدة وقمنا بتطوير مقاربة فك الترميز LDPC جديدة استنادا إلى "خوارزمية Offset MIN-SUM" (OMS). وتقدم الهندسة المقترحة تخفيضا إضافيا في التعقيد وتقلل وقت الاستجابة، مع الحفاظ على أداء فك الترميز.

بالمثل للنهج الستوكاستيك الحديث، تم التعامل مع تقنيات فك الترميز LDPC مع هدف و طريقة جديدة. من ناحية أخرى، قدمنا تصميم متوازيي بشكل كامل، والذي يسمح للحد من مساحة استخدام FPGA و تحسين التقارب. أكد التجسيد بواسطة Xilinx FPGA من صحة تقنية الستوكاستيك لدينا و أدى إلى مزيد من انخفاض في استخدام FPGA وتحسين التقارب، حتى بالنسبة للرموز القصيرة.

**كلمات مفتاحية**: فك الترميز LDPC، الترميز الستوكاستيك، أنظمة FPGA.

**TABLE DES MATIERES**

# REMERCIEMENTS

Je tiens à adresser mes sincères remerciements à Monsieur Ahmed BENALLAL, Professeur à l'Université Blida 1 de l'honneur qu'il m'a fait en acceptant de présider le jury de ma thèse.

Je tiens également à exprimer ma profonde gratitude et ma reconnaissance envers Monsieur Abderrezak GUESSOUM, Professeur à l'Université Blida 1, directeur de cette thèse. Ses idées, ses conseils et ses critiques m'ont été d'une aide précieuse pour mener ce travail à bien.

J'adresse à Monsieur Abdelhamid MERAGHNI Professeur et directeur de l'Ecole Normale Supérieure de Kouba, à Messieurs, FERGANI Belkacem, Professeur à l'Université des Sciences et de la Technologie Houari Boumediene, Monsieur Noureddine BELKHAMSA, Maître de Conférences à l'Université de Blida, mes plus vifs remerciements pour l'intérêt qu'ils ont manifesté pour mon travail et pour avoir accepté la charge d'examinateurs.

Je remercie vivement Monsieur Rachid BEGUENANE, Professeur au Collège militaire royal du Canada à Kingston, pour son aide précieuse, pour sa disponibilité et pour ses conseils.

Mes remerciements vont également à tous les membres du Laboratoire LATSI de l'Université Blida 1. Je tiens à remercier tout particulièrement Monsieur Madjid AREZKI, Maître de conférences à l'Université Blida 1, Monsieur Samir DAHMANI, Maître de conférences à l'Université Blida 1, Rafik BRADAI, Maître de conférences à l'Université Blida 1, Monsieur Abderramane NAMANE, Professeur à l'Université Blida 1, Mountassar MAAMOUN, Maître de conférences à l'Université Blida 1, pour leurs aides précieuses, pour leurs disponibilités et pour leurs conseils.

Je suis très reconnaissant à Monsieur Mohamed AIDJA, Professeur à l'Université Blida 1, pour son aide précieuse, pour sa disponibilité et pour ses conseils.

Mes sincères remerciements vont aussi aux personnes qui m'ont aidé en contribuant, de près ou de loin, à l'aboutissement de ce travail. Qu'ils trouvent dans cette thèse une trace de ma reconnaissance. Je cite en particulier mes amis doctorants et jeunes docteurs du laboratoire LATSI de l'Université Blida 1.

Ces dernières lignes sont pour mes parents, ma famille ainsi que mes amis. Je tiens ici à leur exprimer toute ma reconnaissance pour tout le soutien et tous les encouragements qu'ils ont su me donner tout au long de ce travail et jusqu'au dernier instant.

Que ceux que j'oublie ici veuillent bien me pardonner.

Merci à tous

Ghania ZERARI

# LISTE DES ABREVIATIONS

ASIC : "application-specific integrated circuit"
ASMBG : Méthode stochastique avancée de génération de bits
AWGN : "Additive white Gaussian noise"
BER : "Bit Error Rate"
BPSK : "Binary phase shift keying"
CN : Nœuds de contrôle
CSS : "Controlled Start-up Stochastic decoding"
DC : Cycle de décodage
DS : "Delayed Stochastic"
DVB-S : "Digital Video Broadcasting – Satellite"
EM : "Edge Memories"
FPGA : "Field-Programmable Gate Array"
I-OMS : "Improved Offset Min-sum algorithm"
LDPC : "Low-Density Parity-Check"
LLR : "Log-Likelihood-Ratio"
SPA : "Sum-Product algorithm"
LUT : "Look-Up-Table"
MSA : "MIN-SUM algorithm"
MTFM : "Majority-Based Tracking Forecast Memories"
NMS : "Normalized MIN-SUM algorithm"
OMS : "Offset MIN-SUM algorithm"
RAM : "Random Access Memory"
SNR : "Signal-to-noise ratio"
TFM : "Tracking Forecast Memories"
VHDL : "Very High Speed Integrated Circuit (VHSIC) Hardware Description Language"
VHSIC : "Very High Speed Integrated Circuit"
VN : Nœud de variable

# LISTE DES FIGURES

# INTRODUCTION

Conjointement avec les progressions impressionnantes des ASICs, plusieurs nouvelles versions de circuits programmables et reprogrammables ont été développées, dans le but d'accélérer les processus de conceptions des systèmes électroniques et de réduire le temps et le coût de développement, tout en restant relativement compétitifs sur le plan de la rapidité d'exécution. Le problème est alors de pouvoir concevoir des systèmes, en fonction de différents paramètres tels que la complexité de l'architecture, le nombre de pièces à fabriquer ou le temps accordé. Les décodeurs LDPC ont profité de l'avancée de la technologie, pour renaître une deuxième fois.

Le décodage itératif est la nouvelle approche du décodage correcteur d'erreurs. Les codes LDPC "Low-Density Parity-Check" appartiennent aux codes correcteurs d'erreurs qui se rapprochent de l'efficacité de la limite de Shannon [1][2]. Le nom vient de la caractéristique de leur matrice de contrôle de parité qui contient un faible taux de "1" par rapport aux "0". Leur avantage principal réside d'une part, dans leur performance qui est très proche de la capacité du canal bruité, et d'autre part dans leur temps de décodage pour des algorithmes complexes.

Sa première présentation par Gallager dans sa thèse de PhD date des années 60 [3]-[4]. Mais en raison de manque de structure et la complexité de l'implémentation des codeurs et plus particulièrement l'implémentation des décodeurs, il a été ignoré, dans la plupart du temps [5]-[6]. Ces codes sont adoptés actuellement par différents standards de communications numériques (DVB-S2, IEEE 802.3an (10GBASE-T), IEEE 802.3ba, IEEE 802.16, the IEEE 802.11 (WiFi) [7]-[12].

Il a été clairement prouvé qu'un débit plus élevé est atteint par des solutions de décodage entièrement parallèle, cependant elles impliquent une augmentation de utilisation des ressources FPGA et une élévation de la complexité matérielle. De nombreuses architectures de décodage LDPC stochastique et à complexité réduite sont élaborées, pour contourner ce désagrément. Plusieurs implémentations de décodage LDPC ont été explorées pour atteindre des résultats à haut débit. Les récentes approches du décodage LDPC stochastique ont confirmé leurs adaptabilités pour le décodage parallèle. De plus, pour une réduction supplémentaire de la surface de silicium, différentes architectures et stratégies de décodage LDPC, utilisant les méthodes stochastiques, sont proposées.

Néanmoins, une architecture optimisée pour un décodeur LDPC stochastique basé sur ASIC ne peut pas générer systématiquement une utilisation efficace de logique FPGA. Il est évident que la mise en œuvre ASIC d'un compteur à six bits nécessite moins de surface de silicium par rapport à une mémoire de 64 bits. Cependant, un résultat contraire est obtenu par la mise en œuvre sur un FPGA. Une implémentation d'une mémoire de 64 bits sur un FPGA de Xilinx, peut être routée en utilisant une seule LUT, contrairement à la mise en œuvre du compteur à six bits.

Le but de ce travail est d'étudier le décodage LDPC stochastique existant et de développer une nouvelle approche moins complexe et plus performante implémentable sur FPGA.

Cette thèse est répartie en quatre chapitres :

Le premier chapitre est consacré à la présentation de l’état de l’art, le principe et l'architecture du décodage LDPC classique. Les notions et les calculs, accomplis par l'algorithme de décodage LDPC présentée par Gallager sur le domaine des probabilités et sur le domaine logarithmique, sont examinés.

Le deuxième chapitre expose une nouvelle étude et analyse architecturale des décodeurs LDPC à complexité réduite, afin de proposer une nouvelle stratégie. Nous présentons dans ce chapitre, premièrement les décodeurs à complexité réduite et deuxièmement notre nouveau décodeur. L'étude touchera, le décodage LDPC avec le "MIN-SUM algorithm" (MSA), le décodage LDPC avec le "Normalized MIN-SUM algorithm" (NMS) et le décodage LDPC avec la variante du "Offset MIN-SUM algorithm" (OMS). Notre nouvel algorithme de décodage LDPC est basé sur la variante du "Offset MIN-SUM algorithm" (OMS). L'objectif de l'approche développée est de modifier le traitement de mise à jour des nœuds de parité, en vue de réduire la complexité et de diminuer le temps de réponse, tout en conservant les performances du décodage.

Dans le troisième chapitre, les récentes techniques du décodage stochastique des codes LDPC sont traitées avec une nouvelle méthode et perspective. Ensuite, nous présentons une conception d'une nouvelle architecture de décodage LDPC stochastique qui permette la réduction de l'utilisation de la logique FPGA et l'amélioration de la convergence.

Dans le quatrième chapitre, nous présentons les résultats d'implémentation et les comparaisons entre les architectures de décodage LDPC stochastique basées sur un FPGA. De plus, nous présentons la validation de notre technique qui permet plus de réduction d'utilisation de la logique FPGA et une amélioration de la convergence, même pour les codes courts.

La thèse se termine par une conclusion générale où nous commentons le travail effectué et nous proposons des améliorations et perspectives.

# CHAPITRE 1

# PRINCIPE DU DECODAGE LDPC

## 1.1. Introduction

Le décodage itératif est la nouvelle approche du décodage correcteur d'erreurs. Les codes LDPC "Low-Density Parity-Check" appartient aux codes correcteurs d'erreurs qui se rapprochent de l'efficacité de la limite de Shannon. Le nom "Low-Density Parity-Check" vient de la caractéristique de leur matrice de contrôle de parité qui contient un faible taux de "1" par rapport au taux de "0". Leur avantage principal réside d'une part, dans leur performance qui est très proche de la capacité du canal bruité et d'autre part, dans leur temps de décodage.

Ces codes sont adoptés par différents standards de communications numériques. Plusieurs algorithmes et architectures associées ont été élaborés pour l'implémentation du décodage LDPC. Ce chapitre est une introduction pour le principe et l'architecture du décodage LDPC classique. Les deux approches de calculs accomplis par l'algorithme de décodage LDPC présentées par Gallager sont exposées. La première approche effectue les calculs sur le domaine des probabilités. Dans la deuxième approche, le concept du "Log-Likelihood-Ratio" (LLR) est introduit, et les calculs sont effectués sur un domaine logarithmique. Cette approche est appelée généralement l'algorithme de somme produit "Sum-Product algorithm" (SPA). Nous présentons dans la dernière partie de ce chapitre les performances de quelques codes de l'état de l'art du décodage itératif LDPC, avec l'utilisation de l'approche SPA.

## 1.2. Système de Communication et décodage

Une chaine de communication numérique peut être simplifiée par le synoptique de la Figure.1.1. Cette chaine est composée principalement de trois parties, l'émetteur, le récepteur et le canal de transmission. Le canal est appelé fréquemment un canal bruité.

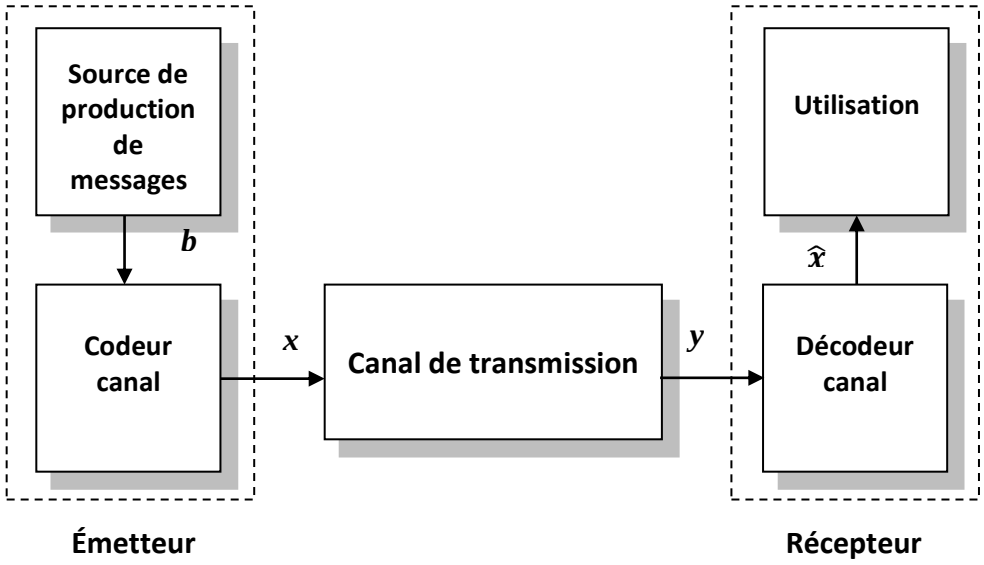


**Figure. 1.1.** Chaine de communication numérique

Dans cette configuration, l'émetteur est constitué d'une source de production de messages ou d'information binaires ***b*** connecté à un codeur canal, insérant des bits de contrôle, suivant une technique de codage canal bien précise.

Le codage peut être considéré comme la production du mot codé ***x***, utilisant l'équation suivante

$$x = b \times G$$

avec ***b*** vecteur d'information et ***G*** la matrice de codage.

Le mot codé $\boldsymbol{x}$ est transmis à travers un canal bruité, qui donnera à la sortie le mot bruité $\boldsymbol{y}$. Ce dernier est le résultat d'ajout d'erreurs, à la variante modulée de $\boldsymbol{x}$. Le décodeur canal du récepteur a pour fonction la correction de $\boldsymbol{y}$ pour récupérer le mot codé $\boldsymbol{x}$, et par conséquent la récupération de l'information avant codage. La sortie du décodeur est le code estimé $\widehat{\boldsymbol{x}}$ de $\boldsymbol{x}$.

## 1.3. Capacité du canal bruité

Depuis la publication des travaux de Shannon [1]-[2], jusqu'au début des années 90, on croyait que les seuls codes capables de se rapprocher de la capacité canal sont des codes extrêmement longs. Cette longueur exceptionnelle impliquait une impossibilité de codage et de décodage dans la pratique.
L'introduction des turbos-codes et des codes LDPC dans les années 90 a redémontré une deuxième fois l'exactitude des travaux de Shannon. Par conséquent, des performances proches de la capacité canal sont devenues possibles [13]-[15].

Shannon a démontré qu'il est théoriquement possible de transmettre l'information à travers un canal bruité à condition que le débit de la transmission soit inférieur à la capacité du canal. Pour un canal Gaussien avec un bruit blanc additif de densité de puissance $\sigma^2 = N_0/2$, la capacité canal exprimée en "bits/second" est donnée par

$$C_{Shannon} = W \times log_2(1 + \frac{P}{\sigma^2})$$

avec $\boldsymbol{P/\sigma^2}$ est le SNR et $\boldsymbol{W}$ est la bande passante en Hertz.

Cette expression analytique de capacité canal est obtenue pour une variable d'entrée qui suit une distribution Gaussienne (0,P).

Généralement, on s'intéresse à une capacité canal en "bits/channel symbol" plutôt qu'en "bits/second". Une telle capacité de canal peut facilement être obtenue via la division de $C_{Shannon}$ dans (1.1) par le débit de symboles de canal ***Rs*** (symboles par seconde). La capacité canal exprimée en "bits/channel symbol" est donnée par

$$C_{Shannon} = \frac{1}{2} \times log_2(1 + \frac{P}{\sigma^2})$$

La Figure.1.2. illustre la variation de la limite de la capacité canal en fonction de Eb/N0, pour une variable d'entrée qui suit une distribution Gaussienne (0,P) et une variable d'entrée binaire modulée en BPSK (-1, +1). La ligne continue indique la limite de Shannon pour la modulation BPSK. La ligne discontinue donne la limite pour l'entrée avec la distribution Gaussienne.

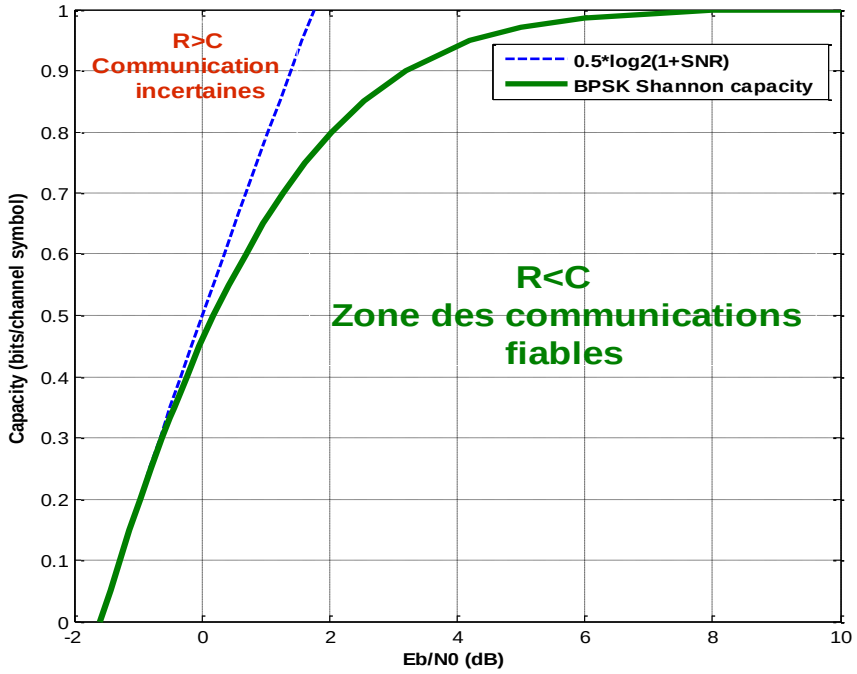


**Figure 1.2** Variation de la limite de la capacité (limite de Shannon) d'un canal avec un bruit blanc Gaussien additif pour une entrée non contrainte et est une entrée binaire modulée en BPSK.

Le tableau 1.1 énumère les limites de Shannon $E_b/N_0$, correspondantes aux rapports de codage sélectionnés, pour deux types de variables d'entrées. La première variable, est dite non contrainte, suit une distribution Gaussienne. La deuxième est une variable binaire modulée en BPSK.

**Tableau 1.1** Les limites $E_b/N_0$ pour différents rapports de codage

| **Rapport de Codage R** | **$E_b/N_0$ (dB) pour variable d'entrée avec distribution Gaussienne** | **$E_b/N_0$ (dB) pour variable d'entrée binaire modulée en BPSK** |
|---|---|---|
| 0.05 | -1.440 | - 1.440 |
| 0.10 | -1.284 | - 1.285 |
| 0.15 | -1.133 | - 1.126 |
| 0.20 | - 0.976 | - 0.963 |
| 0.25 | - 0.817 | - 0.793 |
| 0.30 | - 0.657 | - 0.616 |
| 1/3 | - 0.550 | - 0.497 |
| 0.35 | - 0.495 | - 0.432 |
| 0.40 | - 0.333 | - 0.236 |
| 0.45 | - 0.166 | - 0.030 |
| 0.50 | 0 | 0.187 |
| 0.55 | 0.169 | 0.423 |
| 0.60 | 0.339 | 0.682 |
| 0.65 | 0.511 | 0.960 |
| 2/3 | 0.569 | 1.059 |
| 0.70 | 0.686 | 1.275 |
| 0.75 | 0.860 | 1.626 |
| 0.80 | 1.037 | 2.039 |
| 0.85 | 1.215 | 2.545 |
| 0.90 | 1.396 | 3.199 |
| 0.95 | 1.577 | 4.190 |

Si $E_b$ est l'énergie moyenne par bit d'information et $R$ et le rapport de codage, l'expression de la capacité en fonction de $E_b/N_0$ , exprimée en "bits/channel symbol", sera donnée par

$$C_{Shannon} = \frac{1}{2} \times log_2(1 + \frac{2RE_b}{N_0})$$

Il faut noter qu'une communication fiable est possible à condition que $R < C$. La limite est pour $R = C$. Ainsi, la limite inférieure de $E_b/N_0$ est exprimée par

$$\frac{E_b}{N_0} > \frac{2^{2C} - 1}{2C}$$

Les nouvelles techniques de décodage utilisées actuellement, tels que les turbos-codes et surtout les codes LDPC, sont capables de fonctionner très près des limites de Shannon. Pour certains types de communications, telles que la transmission audio et vidéo, une communication sans erreur n'est pas vraiment nécessaire et un taux d'erreur de bits "BER" avoisinant le $10^{-3}$ est totalement toléré.

Les paragraphes qui suivent sont consacrés aux principes et performances du décodage LDPC.

## 1.4. Représentation matricielle et graphique des codes LDPC

Les codes LDPC, comme tous les codes en bloc linéaires, peuvent être décrits par une représentation graphique ou par une représentation matricielle [16]. Le codage LDPC utilise la matrice génératrice $(k,n)$ $G$, avec

$$G \times H^T = 0$$

$H$ est la matrice de contrôle de parité. Pour les codes LDPC, la circuiterie de codage est relativement plus simple comparant à la circuiterie de décodage.

Pour un code LDPC binaire ***(n, k),*** la taille de la matrice de contrôle de parité ***H*** est donnée par ***(n-k)×n***. Le mot codé *x* doit vérifier l'équation suivante

$$x \times H^T = 0$$

***x*** est composé de ***k*** bits d'informations et ***(n-k)*** bits de parité. La matrice ***H*** est caractérisée par deux nombres. Le premier nombre $w_r$ définit le nombre de ***1*** dans chaque ligne. Le deuxième nombre $w_c$ définit le nombre de ***1*** dans chaque colonne.

Les codes LDPC originaux présentés par Gallager étaient réguliers et suivaient une construction particulière. Un exemple d'un code Gallager de 12 bits de long, avec $w_r = 4$ et $w_c = 3$, est donné par la figure 1.3.

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

**Figure. 1.3.** Matrice H d'un code Gallager de 12 bits de long
($w_r = 4$ et $w_c = 3$)

Un exemple d'un code régulier MacKay Neal de 12 bits de long, avec $w_r = 4$ et $w_c = 3$, est donné par la figure 1.4. Pour qu'une matrice de contrôle ***H*** soit à basse densité, il faut satisfaire simultanément deux conditions $w_c << n$ et $w_r << (n-k)$.

La représentation graphique est généralement nommée graphe de bipartie ou graphe de ''Tanner''. Un graphe de bipartie possède deux groupes de nœuds de traitement, les nœuds de variable ***VNs*** et les nœuds de parité ou de contrôle ***CNs***.

Le graphe de bipartie d'un code LDPC ***(n, k)*** est généralement composé de **n** nœuds de variable et de ***(n-k)*** nœuds de contrôle et des connexions reliant ces deux groupes.

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

**Figure. 1.4.** Matrice H d'un code régulier MacKay Neal de 12 bits de long ($w_r = 4$ et $w_c = 3$)

La figure 1.5 et la figure 1.6 exposent successivement une représentation matricielle et un graphe de ''Tanner'' d'une petite matrice ***H*** pour un code de 6 bits de long avec $\boldsymbol{w_c = 2}$ et $\boldsymbol{w_r = 3}$.

Le nombre de connexions reliées à un nœud représente le degré du nœud. Nous désignons $\boldsymbol{d_v}$ pour le degré des ***VNs*** et $\boldsymbol{d_c}$ pour le degré des ***CNs***, ce qui signifie que $\boldsymbol{w_c = d_v}$ ***et*** $\boldsymbol{w_r = d_c}$ . Un "1" sur la matrice ***H*** est l'équivalent d'une connexion bidirectionnelle entre un ***VN*** et un ***CN***.

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

**Figure. 1.5.** Matrice H avec $d_v = 2$ et $d_c = 3$

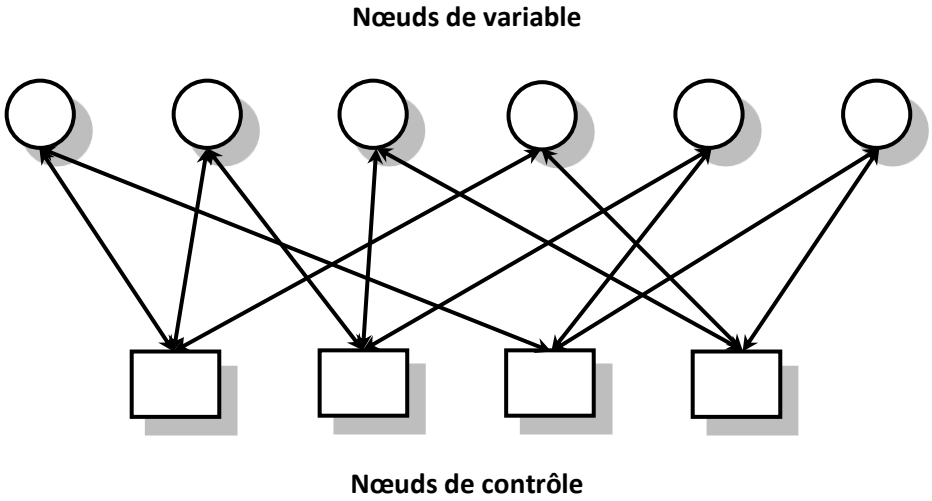


**Figure. 1.6.** Graphe de bipartie ou graphe de ''Tanner''

## 1.5. Code LDPC régulier et irrégulier

Le code LDPC est considéré régulier si ***dc*** est constant pour chaque ligne et ***dv*** est constant pour chaque colonne, ce qui signifie que tous les ***VN***s sont identiques et de même pour les ***CNs***. Pratiquement pour un code régulier la relation entre ***dv*** et ***dc*** est exprimée par

$$dv = dc \times \frac{(n-k)}{n}$$

La matrice ***H*** de la figure 1.4 correspond à un code régulier avec ***dv=2*** et ***dc=3***
Si ***dv*** n'est pas constant le code est considéré comme un code LDPC irrégulier.

## 1.6. Décodage LDPC

Plusieurs travaux et études, sur l'amélioration des performances et sur la réduction de complexité des algorithmes employés dans les décodeurs LDPC, ont

été réalisés indépendamment. C'est pourquoi, en trouve des différents noms pour le décodage LDPC. Le plus commun est l'algorithme de propagation de croyance, l'algorithme de message passant et l'algorithme de somme produit SPA.

Les algorithmes de décodage correcteur d'erreurs LDPC sont basés sur le même principe, d'échange d'informations entre les nœuds de variable et les nœuds de parité, jusqu'à la récupération d'un code correcte ou l'accumulation du nombre maximum d'itérations [17]-[18]. La figure 1.7. présente un synoptique d'un décodeur correcteur d'erreurs LDPC. La partie des nœuds de variable est constituée d'un nombre ***n*** de nœuds, égal à la longueur du code utilisé.

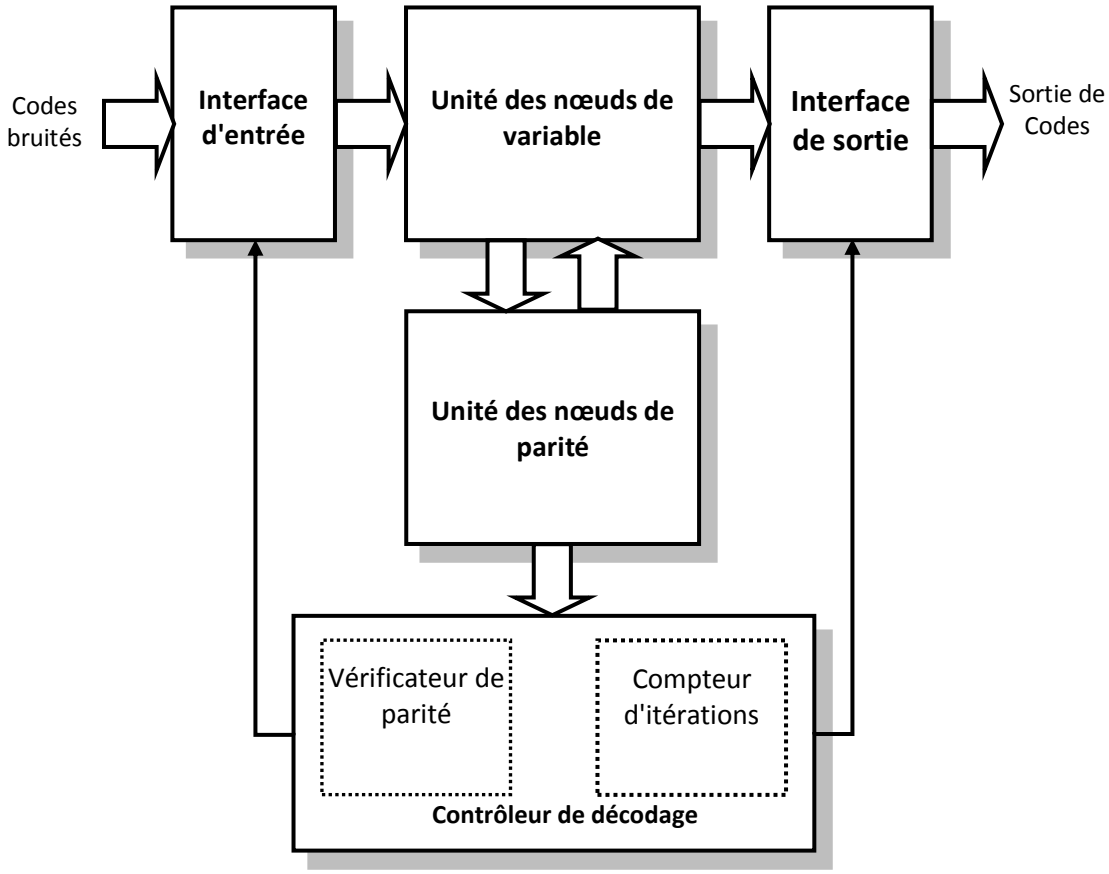


**Figure. 1.7.** Synoptique d'un décodeur correcteur d'erreurs LDPC

La partie des nœuds de parité est constituée d'un nombre (***m = n-k)*** de nœuds, égal à la longueur des bits de parité. Le vérificateur de parité possède ***m*** entrées reliées à la sortie des nœuds de parité. Il active l'interface de sortie en cas de présence d'un code correct.

L'algorithme LDPC se résume par les étapes suivantes :

**1er étape** : le décodeur reçoit, après un passage par le canal de transmission, un code bruité.
L'interface d'entrée alimente les nœuds de variables.

**2éme étape :** Les nœuds de variable et de parité traitent et échangent les informations afin de corriger les erreurs causées par le bruit du canal de transmission.

**3éme étape** : L'interface de sortie fait sortir le code résultant (état des nœuds de variable) dans deux cas.
**1** : Le vérificateur de parité détecte un code correct
**2** : Le décodeur atteint le maximum des itérations.

La 2éme étape de l'algorithme LDPC d'échange d'information et de traitement, effectuée par les nœuds de variable et de parité, représente la partie la plus importante de l'algorithme. De plus, elle occupe plus de 80 % de l'architecture dans l'implémentation matérielle. C'est valable pour les architectures sur ASIC et sur FPGA. Ce taux occupation est lié à l'architecture des nœuds et à la nature des interconnexions entre les nœuds de variable et de parité.

Les deux types de nœuds fonctionnent en deux modes, réception et émission. Une itération dans le décodage LDPC est l'équivalent d'un couple émission-réception entre les nœuds de variable et les nœuds de parité. Dans une architecture totalement parallèle, si les nœuds de variable sont actifs en mode émission les nœuds de parité doivent être actifs en mode réception.

Dans ce qui suit, nous supposons que le vecteur transmis $x = (x_1, \dots, x_N)$ est modulé en BPSK. Le vecteur reçu $y = (y_1, \dots, y_N)$ est obtenu par $y_n = \big((1 - 2x_n) + GN_n\big)$, dont $GN_n$ est le bruit Gaussien, ajouté par le canal.

Nous considérons maintenant un exemple d'un nœud de variable ***VN*** de degré ***dv=3***. Alors, le nœud ***VN*** est connecté à trois nœuds de parité. Le signal bruité $y$ est adapté par un étage de mise en forme afin d'initialiser le nœud ***VN*** avec ***L***. Cette adaptation est propre à chaque type d'algorithmes. La valeur de ***L*** est considérée constante pendant tout le cycle de décodage d'un vecteur $y_n$. Les figures 1.8. et figure 1.9. illustrent les deux modes de fonctionnement du nœud ***VN***. Un nœud de variable recevra les signaux du types $\boldsymbol{L_i}$ de la part des nœuds de parité et répondra par l'envoi des signaux de types $\boldsymbol{Z_j}$. La figure 1.8. présente un nœud de variable en mode réception, connecté à $CN_1$, $CN_2$ et $CN_3$.

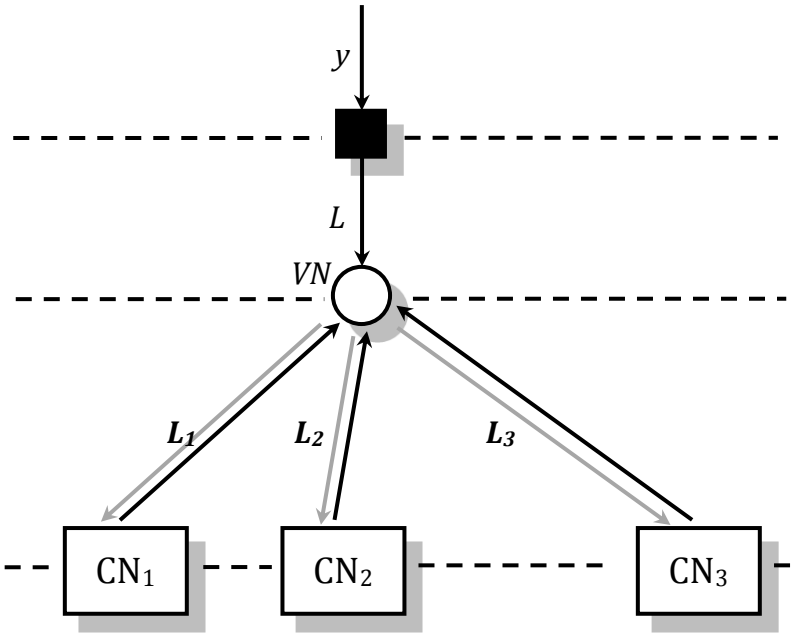


**Figure. 1.8.** Un nœud de variable ***VN*** de degré 3 en mode réception

Pour chaque itération le nœud ***VN*** recevra les trois signaux $\boldsymbol{L_1}$, $\boldsymbol{L_2}$ et $\boldsymbol{L_3}$. Les valeurs de ces signaux sont mises à jour, à chaque itération.

Le mode émission du nœud ***VN*** est illustré sur la figure 1.9. Les trois signaux de sortie $Z_1$, $Z_2$ et $Z_3$ sont calculés sur la base des signaux $L_1$, $L_2$ et $L_3$. Pour plus d'exactitude, un signal $Z_j$ utilise tous les signaux reçus $L_i$ excepté le signal $L_{i=j}$ . Pour notre cas, la sortie $Z_1$ est évaluée sur la base de $L_2$ et $L_3$. Avec la même règle, $Z_2$ est évaluée sur la base de ($L_1$ , $L_3$) et $Z_3$ est évaluée sur la base de ($L_1$ , $L_2$).

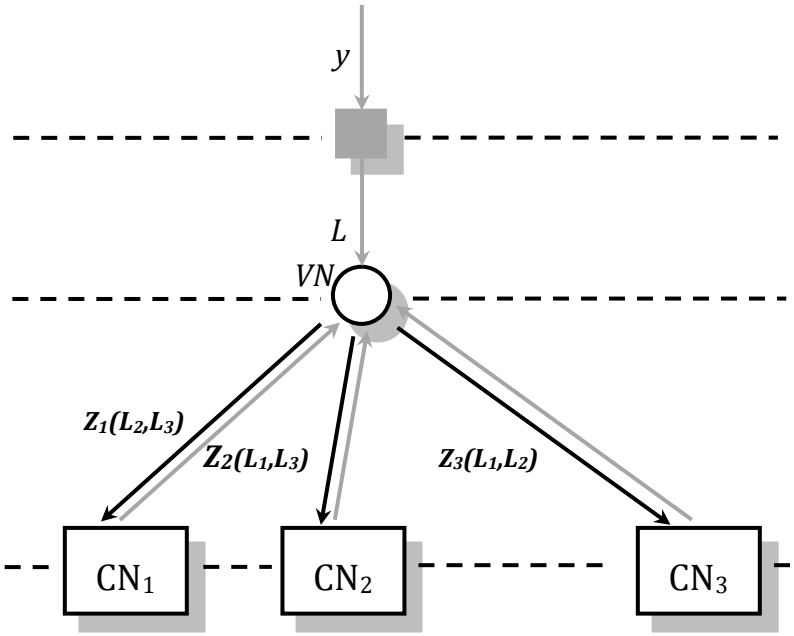


**Figure. 1.9.** UN nœud de variable ***VN*** de degré 3 en mode émission

L'exemple de nœud de parité examiné dans cette partie est d'un nœud de parité de degré ***dc=4***. Ainsi, le nœud ***CN*** est connecté à quatre nœuds de variable. Les figures 1.10. et figure 1.11. exposent les deux modes de fonctionnement du nœud ***CN***. Un nœud de parité reçoit les signaux du type $Z_j$ provenant des nœuds de variable et envoie les signaux du type $L_i$. La phase de réception accomplie par un nœud de parité, connecté aux nœuds de variable ($VN_1$, $VN_2$, $VN_3$, $VN_4$,), est présentée sur la figure 1.10. Dans cette phase le nœud de parité reçoit les quatre signaux $Z_1$, $Z_2$, $Z_3$ et $Z_4$ . Il est clair que cette tâche s'exécute simultanément avec l'envoi des nœuds de variable.

La figure 1.11 résume le mode d'émission du nœud ***CN***. Les quatre signaux de sortie $L_1$, $L_2$, $L_3$ et $L_4$ sont calculés en fonction des signaux $Z_1$, $Z_2$, $Z_3$ et $Z_4$.

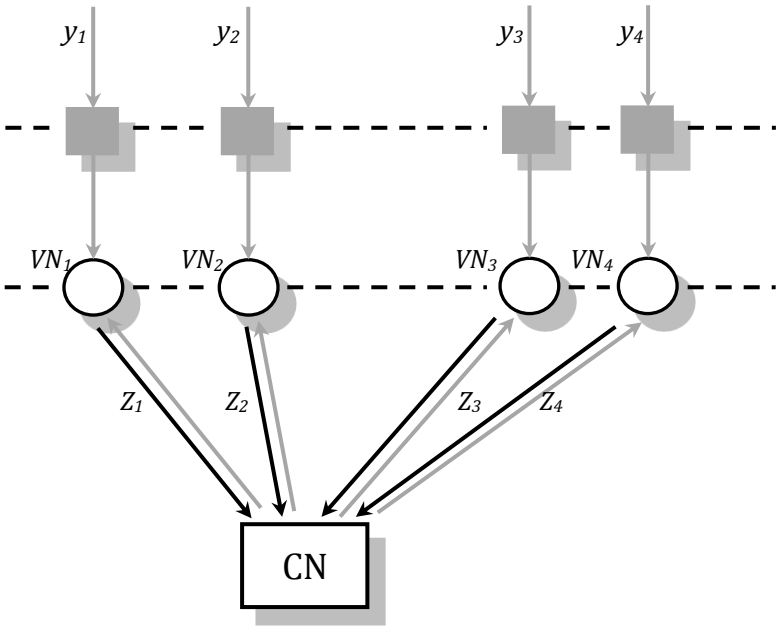


**Figure. 1.10.** Un nœud de parité ***CN*** de degré 4 en mode réception

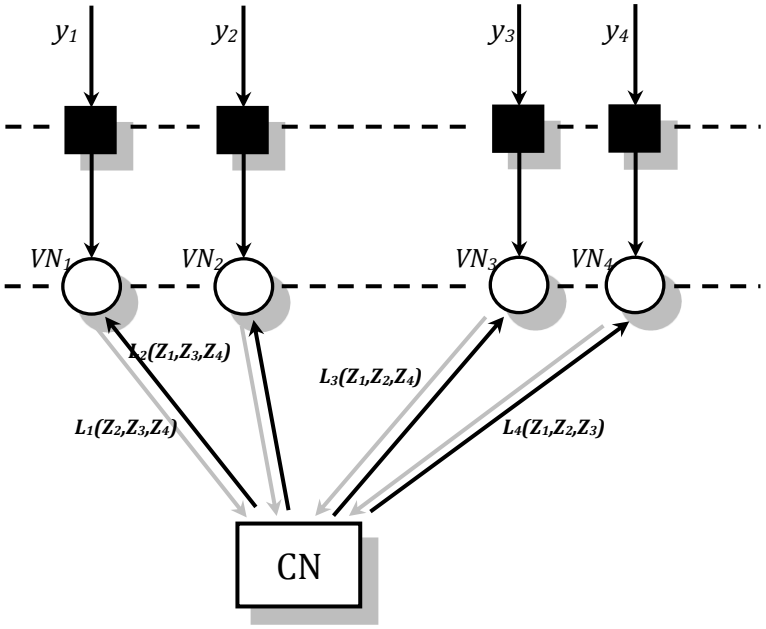


**Figure. 1.11.** Un nœud de parité ***CN*** de degré 4 en mode émission

Avec le même principe utilisé dans les nœuds de variable, un signal $L_i$ utilise tous les signaux reçus $Z_j$ sauf $Z_{j=i}$. La sortie $L_1$ est évaluée sur la base de ($Z_2$, $Z_3$, $Z_4$). De la même façon, $L_2$ utilise ($Z_1$, $Z_3$, $Z_4$), $L_3$ utilise ($Z_1$, $Z_2$, $Z_4$) et $L_4$ utilise ($Z_1$, $Z_2$, $Z_3$).

## 1.6.1. Décodage LDPC dans le domaine des probabilités

La première variante de l'algorithme de décodage LDPC présentée par Gallager sur le domaine des probabilités est basée sur trois étapes. La première repose sur l'initialisation. La deuxième qui englobe la partie la plus complexe, repose sur le calcul des itérations. La troisième étape concerne la décision.

Les détails du décodage LDPC et des calculs, réalisés par les nœuds de parité et les nœuds de variable sur le domaine des probabilités de la première approche donnée par Gallager, sont synthétisés dans l'algorithme que nous dénotons "**Algorithme 1:** Décodage LDPC dans le domaine des probabilités". L'algorithme 1 est décrit comme suit:

---

**Algorithme 1** Décodage LDPC dans le domaine des probabilités

---

Entrée: $y = (y_1, \dots, y_N)$
Sortie: $\hat{x} = (\hat{x}_1, \dots, \hat{x}_N,)$

**Initialisation:**

1. pour tout $n = 1, \dots, N$ faire $L_n = P_n = \Pr(x_n = 1 | y_n)$
2. pour tout $n = 1, \dots, N$ $et$ $m \in M(n)$ faire $Z_n = L_n$

**Itérations:**

3.Mise à jour des nœuds de parité

pour tout $m = 1, \dots, M$ $et$ $n \in N(m)$ faire

$$L_{m\to n} = \frac{1}{2} - \frac{1}{2}\left(\prod_{n'\in N(m)\backslash n}\left(1 - 2\left(Z_{n'\to m}\right)\right)\right)$$

4. Mise à jour des nœuds de variable

Pour tout $n = 1, \dots, N \; et \; m \in M(n)$ faire

$$Z_{n\to m} = \frac{L_n \times \prod_{m'\in M(n)\backslash m}\left(L_{m'\to n}\right)}{L_n \times \prod_{m'\in M(n)\backslash m}\left(L_{m'\to n}\right) + \; (1 - L_n) \times \prod_{m'\in M(n)\backslash m}\left(1 - L_{m'\to n}\right)}$$

5. Mise à jour des états des nœuds variable

Pour tout $n = 1, \dots, N$ faire

$$Z_n = \frac{L_n \times \prod_{m'\in M(n)}\left(L_{m'\to n}\right)}{L_n \times \prod_{m'\in M(n)}\left(L_{m'\to n}\right) + \; (1 - L_n) \times \prod_{m'\in M(n)}\left(1 - L_{m'\to n}\right)}$$

6. Décision:

On détermine à partir de $Z_n$ la séquence décodée $\hat{x}$ .

Pour tout $n = 1, \dots, N$ faire $\hat{x}_n = \begin{cases} 1, & si \; Z_n > 0.5 \\ 0, & si \; non \end{cases}$

Si l'équation de syndrome,

$$\hat{x} \times \mathrm{H}^{\mathrm{T}} = 0$$

est vérifiée le code est validé, sinon on retourne à l'étape 3.

Le code est déclaré incorrect si le nombre d'itérations atteint $It_{max}$.

## 1.6.2. Décodage LDPC avec "Sum-Product algorithm" SPA

Vu la complexité du calcul introduit par le premier algorithme, Gallager a présenté une deuxième approche, dans le domaine logarithmique. Nous parlons ici du concept du "Log-Likelihood-Ratio" (LLR). En pratique, l'utilisation des messages LLRs offre des avantages d'implémentations comparant à l'utilisation dans le domaine des probabilités. Dans cette approche les multiplications sont remplacées par des additions. De plus, l'étape de normalisation est éliminée. Cet algorithme est appelé généralement "Sum-Product algorithm" SPA.

Ce deuxième algorithme que nous dénotons "**Algorithme 2:** Décodage LDPC dans le domaine logarithmique " est décrit comme suit:

---

**Algorithme 2** Décodage LDPC dans le domaine logarithmique

---

Entrée: $y = (y_1, \dots, y_N)$

Sortie: $\hat{x} = (\hat{x}_1, \dots, \hat{x}_N,)$

**Initialisation:**

1. pour tout $n = 1, \dots, N$ faire $L_n = 4y_n/N_0$
2. pour tout $n = 1, \dots, N \; et \; m \in M(n)$ faire $Z_n = L_n$

**Itérations:**

3. Mise à jour des nœuds de parité

Pour tout $m = 1, \dots, M \; et \; n \in N(m)$ faire

$$L_{m \to n} = \left( \prod_{n' \in N(m) \backslash n} sgn\left(Z_{n' \to m}\right) \right) \times f\left( \sum_{n' \in N(m) \backslash n} f\left(\left|Z_{n' \to m}\right|\right) \right)$$

Avec

$$f(x) = \log\left(\frac{e^x + 1}{e^x - 1}\right)$$

4. Mise à jour des nœuds de variable

Pour tout $n = 1, \dots, N$ $et$ $m \in M(n)$ faire

$$Z_{n\to m} = L_n + \sum_{m'\in M(n)\setminus m} L_{m'\to n}$$

5. Mise à jour des états des nœuds variable

Pour tout $n = 1, \dots, N$ faire

$$Z_n = L_n + \sum_{m\in M(n)} L_{m\to n}$$

6. Décision:

On détermine à partir de $Z_n$ la séquence décodée $\hat{x}$ .

Pour tout $n = 1, \dots, N$ faire $\hat{x}_n = \frac{1-sgn(Z_n)}{2}$

Si l'équation de syndrome

$$\hat{x} \times H^T = 0$$

est vérifiée, le code est validé, sinon on retourne à l'étape 3.

Le code est déclaré incorrect si le nombre d'itérations atteint $It_{max}$.

## 1.7. Performance du décodage LDPC

Dans cette section, nous présentons les performances de quelques codes LDPC de l'état de l'art, avec décodage itératif en utilisant l'approche de l'algorithme SPA [20]. Pour la comparaison des performances, nous adoptons une modulation BPSK et une transmission sur un canal à bruit Gaussien additif.

Le premier exemple est un code LDPC régulier (2048, 1723) avec *dv=6* et *dc=32*. Le rapport de décodage est R= 0,841. Deux méthodes de construction sont utilisées pour créer la matrice de décodage H, la méthode Gallager et la méthode MacKey. Les performances d'erreur de bit et de trame avec le décodage SPA sont représentées sur la figure 1.12.

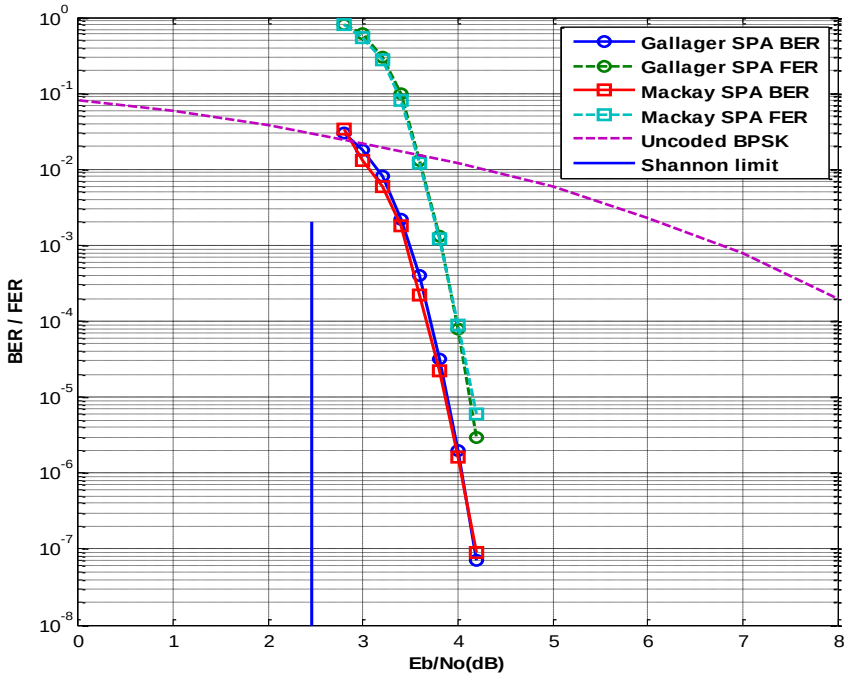


**Figure. 1.12.** Les performances du code (2048, 1723) avec un décodage LDPC-SPA

Nous pouvons aisément voir que les deux codes ont presque la même performance d'erreur de bit. Pour un BER de $10^{-5}$, le code n'accomplisse qu'un

écart de 1,35 dB à partir de la limite de Shannon. Un gain de codage de 6 dB par rapport au BPSK non codé est atteint, pour un BER de $10^{-6}$.

La figure 1.13 représente la convergence du décodage SPA pour le code Gallager-LDPC de l'exemple de figure 1.12. Nous apercevons que, pour un BER de $10^{-5}$, l'écart de performance entre 2 et 100 itérations est inférieur à 1,33 dB, et l'écart de performance entre 10 et 100 itérations est inférieur à 0,21 dB.

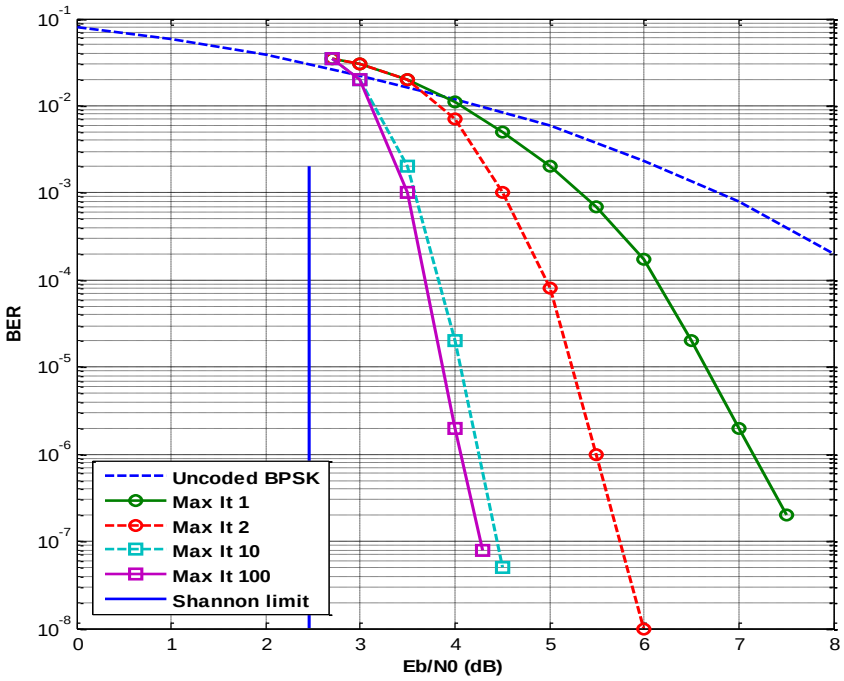


**Figure. 1.13.** La convergence du décodage SPA pour le code (2048, 1723) Gallager-LDPC

La figure 1.14 représente la convergence du décodage SPA pour le code MacKay-LDPC. Nous apercevons que ce code a une convergence légèrement plus lente que le code Gallager-LDPC.

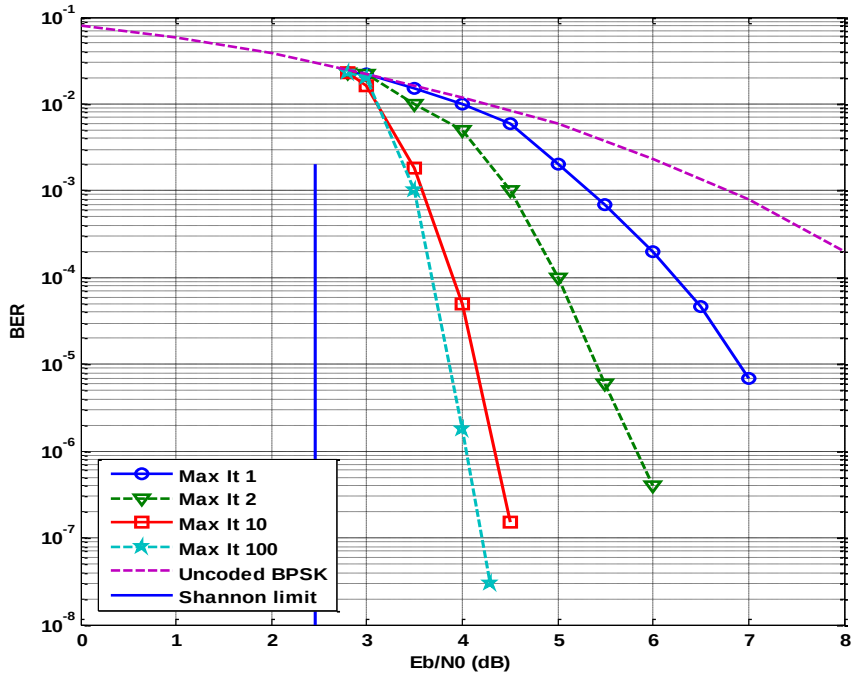


**Figure. 1.14.** La convergence du décodage SPA pour le code (2048, 1723) MacKay-LDPC

Le deuxième exemple est un code Gallager LDPC régulier (8192, 6754) avec *dv=6* et *dc=32*. Le rapport de décodage est R= 0,824. Les performances d'erreur de bit et de trame avec le décodage SPA sont représentées sur la figure 1.15. Pour un BER de $10^{-5}$, le code n'accomplisse qu'un écart de 1,21 dB à partir de la limite de Shannon. Un gain de codage de 3,6 dB par rapport au BPSK non codé est atteint, pour un BER de $10^{-3}$.

La figure 1.16 présente les performances d'un code Gallager-LDPC régulier (12288, 10845) avec *dv=6, dc=48* et un rapport de codage R=0,8825. Pour un BER de $10^{-5}$, le code effectue seulement un écart de 1 dB à partir de la limite de Shannon. À titre de comparaison, la performance de la matrice de décodage générée par la méthode MacKay de même longueur est également donnée sur la figure 1.16. Nous remarquons que le code MacKay est meilleur de 0,2 dB du code Gallager, pour un BER de $10^{-5}$ et BER de $10^{-6}$.

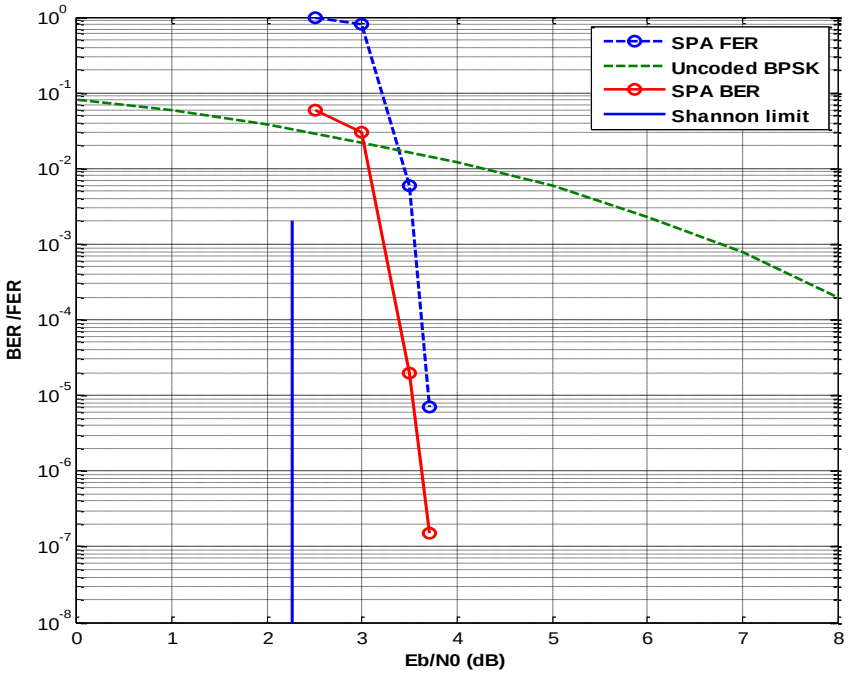


**Figure. 1.15.** Les performances du code $(8192, 6754)$ avec un décodage LDPC-SPA

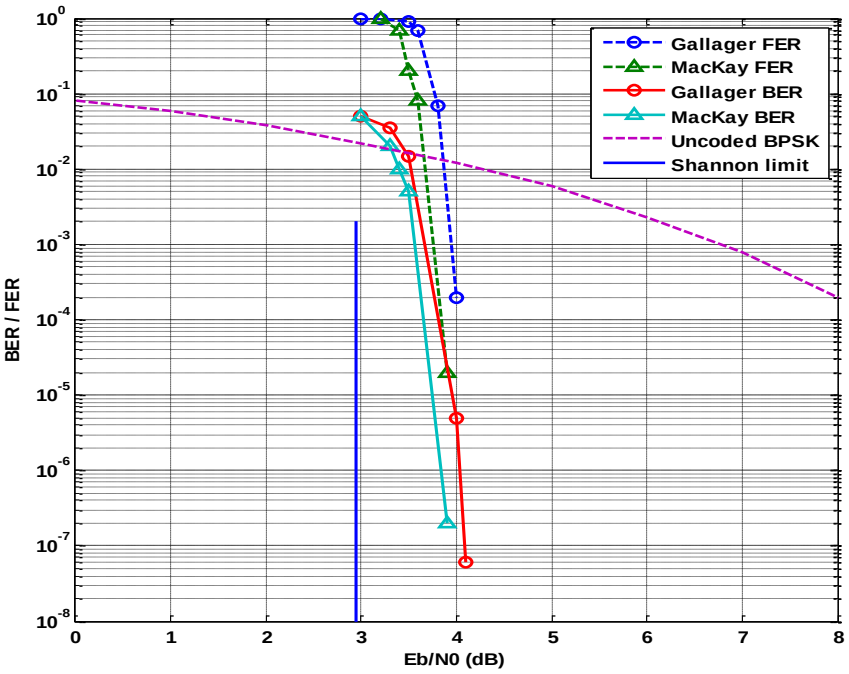


**Figure. 1.16.** Les performances du code $(12288, 10845)$ avec un décodage LDPC-SPA

## 1.8. Conclusion

Dans ce premier chapitre, nous avons introduit les concepts de base des codes LDPC qui sont adoptés par les différents standards de communications numériques (DVB-S2, IEEE 802.3an (10GBASE-T), IEEE 802.3ba, IEEE 802.16, the IEEE 802.11 (WiFi). Vu la complexité du calcul introduit par le premier algorithme, Gallager a présenté une deuxième approche, dans le domaine logarithmique. Néanmoins, cette deuxième approche reste relativement complexe. Ce qui a poussé d'une part, à développer plusieurs variantes de décodage LDPC à complexité réduite, et d'autre part de réduire davantage la complexité et d'améliorer les performances.

De ce fait, le deuxième chapitre sera consacré à une nouvelle analyse détaillée des architectures LDPC à complexité réduites et de proposer un nouveau décodeur LDPC à complexité réduite et hautement performant.

# Chapitre 2

# LES DECODEURS LDPC A COMPLEXITE REDUITE ET DEVELOPPEMENT D'UNE NOUVELLE APPROCHE

## 2.1. Introduction

Motiver par les hautes performances du décodage LDPC avec l'algorithme SPA, les simplifications se concentrent sur les mises à jour des nœuds de parité pour obtenir des dérivées à complexité réduite qui atteignent des performances quasi optimales [21]-[29]. Cette approche est liée à la complexité élevée des mises à jour des nœuds de parité, par rapport à la complexité des mises à jour des nœuds de variable,

Le but de ce chapitre est de faire une nouvelle étude et analyse architecturale des décodeurs LDPC à complexité réduite, afin de proposer une nouvelle stratégie. L'étude touchera premièrement, le décodage LDPC avec le "MIN-SUM algorithm" (MSA), deuxièmement le décodage LDPC avec le "Normalized MIN-SUM algorithm" (NMS) et troisièmement le décodage LDPC avec la variante du "Offset MIN-SUM algorithm" (OMS). L'introduction de la nouvelle stratégie de décodage est développée après la première partie de ce chapitre consacrée aux décodeurs à complexité réduite. Le nouvel algorithme de décodage LDPC est basé sur la variante du "Offset MIN-SUM algorithm" (OMS).

L'objectif de l'approche développée est de modifier le traitement de mise à jour des nœuds de parité, en vue de réduire la complexité et de diminuer le temps de réponse, tout en conservant les performances du décodage. Pour effectuer cette modification, la mise à zéro, des messages LLR inférieures à la constante de décalage (Offset), est remplacée par une approximation intelligente. Par conséquent, les comparateurs sont écartés des unités de sortie des nœuds de parité. Ce qui implique une réduction supplémentaire des temps de réponse des nœuds.

Les résultats des simulations Monte-Carlo montrent que le nouvel algorithme de décodage atteint une performance très proche de l'algorithme de décodage somme-produit (SPA), avec une complexité inférieure à la complexité du décodage OMS classique.

## 2.2. Décodage LDPC avec le "MIN-SUM algorithm"

Vu que la complexité des mises à jour des nœuds de parité est beaucoup plus conséquente relativement à la complexité des mises à jour des nœuds de variable, les simplifications se concentrent sur les mises à jour des nœuds de parité pour obtenir des dérivées à complexité réduite qui atteignent des performances quasi optimales.

Il a été constaté que $\left|f\left(\sum_{n'\in N(m)\backslash n} f\ \ (|Z_{n'\to m}|)\right)\right|$ suit principalement le $\left|\min_{n'\in N(m)\backslash n}|Z_{n'\to m}|\right|$, de plus nous avons:

$$\left|f\left(\sum_{n'\in N(m)\backslash n} f\,(|Z_{n'\to m}|)\right)\right| \leq \left|\min_{n'\in N(m)\backslash n}|Z_{n'\to m}|\right| \qquad (2.1)$$

alors les mises à jour des nœuds de parité peuvent être approximées par

$$L_{m\to n} \approx \left( \prod_{n' \in N(m)\setminus n} sgn\left(Z_{n'\to m}\right) \right) \times \left( \min_{n' \in N(m)\setminus n} \left|Z_{n'\to m}\right| \right) \qquad (2.2)$$

qui est la mise à jour des nœuds de parité dans le nouvel algorithme de décodage LDPC.

Cet algorithme est nommé "Min-sum algorithm" (MSA), que nous dénotons "**Algorithme 3:** Décodage LDPC avec le MIN-SUM algorithm "
L'algorithme 3 est décrit comme suit:

---

**Algorithme 3** Décodage LDPC MIN-SUM

---

Entrée: $y = (y_1, \dots, y_N)$
Sortie: $\hat{x} = (\hat{x}_1, \dots, \hat{x}_N,)$

**Initialisation:**

1. pour tout $n = 1, \dots, N$ faire $L_n = 4y_n/N_0$
2. pour tout $n = 1, \dots, N \; et \; m \in M(n)$ faire $Z_n = L_n$

**Itérations:**

3. Mise à jour des nœuds de parité

Pour tout $m = 1, \dots, M \; et \; n \in N(m)$ faire

$$L_{m\to n} = \left( \prod_{n' \in N(m)\setminus n} sgn\left(Z_{n'\to m}\right) \right) \times \left( \min_{n' \in N(m)\setminus n} \left|Z_{n'\to m}\right| \right)$$

4. Mise à jour des nœuds de variable

Pour tout $n = 1, \dots, N \; et \; m \in M(n)$ faire

$$Z_{n \to m} = L_n + \sum_{m' \in M(n) \setminus m} L_{m' \to n}$$

5. Mise à jour des états des nœuds variable

Pour tout $n = 1, \dots, N$ faire

$$Z_n = L_n + \sum_{m \in M(n)} L_{m \to n}$$

6. Décision:

On détermine à partir de $Z_n$ la séquence décodée $\hat{x}$ .

Pour tout $n = 1, \dots, N$ faire $\hat{x}_n = \frac{1 - sgn(Z_n)}{2}$

Si l'équation de syndrome

$$\hat{x} \times \mathrm{H}^{\mathrm{T}} = 0$$

est vérifiée, le code est validé, sinon on retourne à l'étape 3. Le code est déclaré incorrect si le nombre d'itérations atteint $It_{max}$.

---

La complexité de calcul de l'étape 3 de l'algorithme 1, estimée à ***2dc*** additions plus ***2dc*** fonctions spéciales ***(f(x))***, est réduite à ***dc +(log$_2$dc) - 2*** additions (comparaisons).

En pratique, il est nécessaire de déterminer les deux LLR ayant les plus petites amplitudes, ainsi que les signes des messages sortants. Alors, seulement ces

deux valeurs résultantes doivent être stockées, car les branches ***($d_c$-1)*** partagent les mêmes messages de sorties [30]-[33].

Il a été constaté que l'amplitude de $L_{m\to n}$ de l'algorithme 3 est toujours supérieure à celle de l'algorithme 1.

$$\left|\min_{n'\in N(m)\setminus n}|Z_{n'\to m}|\right| > \left|f\left(\sum_{n'\in N(m)\setminus n} f\left(\left|Z_{n'\to m}\right|\right)\right)\right| \tag{2.3}$$

Ceci suggère un traitement ou une correction supplémentaire de $L_{m\to n}$ de l'algorithme 3 pour obtenir des valeurs plus précises.

## 2.3. Décodage LDPC avec le "Normalized MIN-SUM algorithm" NMS

L'approximation basée sur MSA peut être améliorée en employant une mise à jour des nœuds de parité qui utilise une constante de normalisation $\boldsymbol{\alpha}$ inférieure à un. Les nouvelles mises à jour des nœuds de parité peuvent être approximées par

$$L_{m\to n} \approx \alpha\left(\prod_{n'\in N(m)\setminus n} sgn\left(Z_{n'\to m}\right)\right)\times\left(\min_{n'\in N(m)\setminus n}\left|Z_{n'\to m}\right|\right) \tag{2.4}$$

Cet algorithme est nommé " Normalized Min-sum algorithm " (NMS), que nous dénotons " **Algorithme 4 :** Décodage LDPC avec le Normalized MIN-SUM algorithm ".

L'algorithme 4 est décrit comme suit:

**Algorithme 4** Décodage LDPC Normalized MIN-SUM

Entrée: $y = (y_1, \dots, y_N)$

Sortie: $\hat{x} = (\hat{x}_1, \dots, \hat{x}_N,)$

**Initialisation:**

1. pour tout $n = 1, \dots, N$ faire $L_n = 4y_n/N_0$
2. pour tout $n = 1, \dots, N \; et \; m \in M(n)$ faire $Z_n = L_n$

**Itérations:**

3. Mise à jour des nœuds de parité

Pour tout $m = 1, \dots, M \; et \; n \in N(m)$ faire

$$L_{m \to n} = \alpha \left( \prod_{n' \in N(m) \setminus n} sgn(Z_{n' \to m}) \right) \times \left( \min_{n' \in N(m) \setminus n} |Z_{n' \to m}| \right)$$

$$avec \qquad 0 < \alpha \leq 1$$

4. Mise à jour des nœuds de variable

Pour tout $n = 1, \dots, N \; et \; m \in M(n)$ faire

$$Z_{n \to m} = L_n + \sum_{m' \in M(n) \setminus m} L_{m' \to n}$$

5. Mise à jour des états des nœuds variable

Pour tout $n = 1, \ldots, N$ faire

$$Z_n = L_n + \sum_{m \in M(n)} L_{m \to n}$$

6. Décision:

On détermine à partir de $Z_n$ la séquence décodée $\hat{x}$ .

Pour tout $n = 1, \ldots, N$ faire $\hat{x}_n = \frac{1 - sgn(Z_n)}{2}$

Si l'équation de syndrome

$$\hat{x} \times \mathrm{H}^{\mathrm{T}} = 0$$

est vérifiée, le code est validé, sinon on retourne à l'étape 3. Le code est déclaré incorrect si le nombre d'itérations atteint $It_{max}$.

---

Bien que $\boldsymbol{\alpha}$ doit varier avec différents rapports signal sur bruit (SNR) et différentes itérations pour obtenir les performances optimales, il est gardé constant pour la simplicité architecturale.

La nouvelle complexité de calcul de l'étape 3 de l'algorithme 4 est évaluée à $\boldsymbol{dc + (\log_2 dc) - 2}$ additions (comparaisons) plus $\boldsymbol{2}$ multiplications.

## 2.4. Décodage LDPC avec le "Offset MIN-SUM algorithm" OMS

L'approximation basée sur MSA peut être améliorée en utilisant une mise à jour des nœuds de parité qui utilise une correction par un offset $\beta$ positif. Les nouvelles mises à jour des nœuds de parité peuvent être approximées par

$$L_{m\to n} \approx \left( \prod_{n' \in N(m)\setminus n} sgn\left(Z_{n'\to m}\right) \right) \times \max\left\{ \left( \min_{n' \in N(m)\setminus n} \left|Z_{n'\to m}\right| \right) - \beta, 0 \right\} \qquad (2.5)$$

Cet algorithme est nommé "Offset Min-sum algorithm" (OMS), que nous dénotons "**Algorithme 5:** Décodage LDPC avec l'Offset MIN-SUM algorithm".

L'algorithme 5 est décrit comme suit:

---

**Algorithme 5** Décodage LDPC Offset MIN-SUM

---

Entrée: $y = (y_1, \dots, y_N)$

Sortie: $\hat{x} = (\hat{x}_1, \dots, \hat{x}_N,)$

**Initialisation:**

1. pour tout $n = 1, \dots, N$ faire $L_n = 4y_n/N_0$
2. pour tout $n = 1, \dots, N$ *et* $m \in M(n)$ faire $Z_n = L_n$

**Itérations:**

3. Mise à jour des nœuds de parité

Pour tout $m = 1, \dots, M$ *et* $n \in N(m)$ faire

$$L_{m\to n} = \left( \prod_{n'\in N(m)\setminus n} sgn\left(Z_{n'\to m}\right) \right) \times \max\left\{ \left( \min_{n'\in N(m)\setminus n} \left|Z_{n'\to m}\right| \right) - \beta, 0 \right\}$$

$$avec \qquad \beta > 0$$

4. Mise à jour des nœuds de variable

Pour tout $n = 1, \dots, N \; et \; m \in M(n)$ faire

$$Z_{n\to m} = L_n + \sum_{m'\in M(n)\setminus m} L_{m'\to n}$$

5. Mise à jour des états des nœuds variable

Pour tout $n = 1, \dots, N$ faire

$$Z_n = L_n + \sum_{m\in M(n)} L_{m\to n}$$

6. Décision:

On détermine à partir de $Z_n$ la séquence décodée $\hat{x}$ .

Pour tout $n = 1, \dots, N$ faire $\quad \hat{x}_n = \frac{1-sgn(Z_n)}{2}$

Si l'équation de syndrome

$$\hat{x} \times \mathrm{H}^{\mathrm{T}} = 0$$

est vérifiée, le code est validé, sinon on retourne à l'étape 3. Le code est déclaré incorrect si le nombre d'itérations atteint $It_{max}$.

Une méthode pour obtenir l'offset de correction $\beta$ a été exposée dans [21], où il a également été montré que l'utilisation d'un $\beta$ constant n'entraîne qu'une perte négligeable de performance par rapport au décodage SPA. Une approche similaire a également été présentée dans [22].

La complexité de calcul de l'étape 3 de l'algorithme 5 est évaluée à $dc + (\log_2 dc) - 2$ additions (comparaisons) plus $2$ soustractions signées et $2$ comparaisons. Nous pouvons facilement constater que l'approche OMS et moins complexe de l'approche NMS, car les deux multiplications sont remplacées par des soustracteurs et comparateurs.

Il a été prouvé par plusieurs travaux que les performances des décodeurs LDPC avec architecture OMS à virgule fixe, dépasse les performances des décodeurs LDPC avec architecture SPA [25]-[26].

## 2.5. Nouvelle approche de décodage LDPC à complexité réduite

Plusieurs travaux de recherche ont démontré que les performances des décodeurs LDPC à complexité réduite et à base d'architecture rigoureusement développée, avoisinent et dépassent même les performances des décodeurs LDPC avec une architecture basée sur un SPA standard. Cette dernière particularité nous a poussé à entamer et de canaliser une partie de nos recherches vers le développement d'un décodeur LDPC hautement performant non stochastique et à complexité réduite.

Il a été noté que la valeur de $\left|\min_{n'\in N(m)\backslash n}|Z_{n'\to m}|\right|$ est légèrement surestimée par rapport à $\left|f\left(\sum_{n'\in N(m)\backslash n} f\quad(|Z_{n'\to m}|)\right)\right|$.

L'algorithme du "Offset Min-sum algorithm" (OMS) est une variété améliorée du MSA, avec une augmentation limitée de la complexité. Les modifications réalisées par l'OMS compensent la surestimation du traitement MSA, en introduisant un facteur d'offset $\boldsymbol{\beta > 0}$.

L'amplitude de la fonction $L_{m\to n}$ de l'étape 3 de l'algorithme peut être illustrée comme suit:

$$|L_{m\to n}| = \begin{cases} \left(\min_{n'\in N(m)\backslash n}|Z_{n'\to m}|\right) - \beta & if \ \min_{n'\in N(m)\backslash n}|Z_{n'\to m}| > \beta \\ 0 & if \ \min_{n'\in N(m)\backslash n}|Z_{n'\to m}| \leq \beta \end{cases} \tag{2.6}$$

Nous pouvons clairement observer de l'équation précédente que les messages LLR inférieurs à $\boldsymbol{\beta}$ sont approchés par zéro. Par conséquent, les contributions des messages mentionnés sont exclues de la prochaine mise à jour. La section suivante présente une nouvelle approximation de l'OMS, dans le but de réduire la complexité et de maintenir les performances du décodage.

Il est évident que les approximations précises, des amplitudes de la fonction $L_{m\to n}$ sont comprises entre zéro et $\boldsymbol{\beta}$. Par conséquent, une correction supplémentaire et une $|L_{m\to n}|$ plus précise peut être donnée par:

$$|L_{m\to n}| = \begin{cases} \left(\min_{n'\in N(m)\setminus n}|Z_{n'\to m}|\right)-\beta & if \ \min_{n'\in N(m)\setminus n}|Z_{n'\to m}| > \beta \\ \min_{n'\in N(m)\setminus n}|Z_{n'\to m}| - \alpha \min_{n'\in N(m)\setminus n}|Z_{n'\to m}| & if \ \min_{n'\in N(m)\setminus n}|Z_{n'\to m}| \le \beta \end{cases}$$
$$with \qquad 0 \le \alpha \le 1 \qquad (2.7)$$

Le facteur $\boldsymbol{\alpha}$ devrait être variable, pour accomplir plus de performance. Néanmoins, il sera défini constant pour la simplicité architecturale. À partir de (2.6) et (2.7), on peut estimer l'erreur d'approximation, associée aux messages avec une amplitude inférieure à $\boldsymbol{\beta}$ par

$$Er_1 = \min_{n'\in N(m)\setminus n}|Z_{n'\to m}| - \alpha \min_{n'\in N(m)\setminus n}|Z_{n'\to m}| \qquad (2.8)$$

L'objectif de l'approche proposée est de modifier l'approximation des amplitudes de la fonction $\boldsymbol{L_{m\to n}}$ inférieure à $\boldsymbol{\beta}$. Tout d'abord, nous introduisons un nouveau traitement des sorties des CNs, dans lequel la mise à zéro est remplacé par une approximation positive et avec une forme similaire aux amplitudes supérieures à $\boldsymbol{\beta}$. L'amplitude est exprimée comme suit:

$$|L_{m\to n}| = \begin{cases} \left(\min_{n'\in N(m)\setminus n}|Z_{n'\to m}|\right)-\beta & if \ \min_{n'\in N(m)\setminus n}|Z_{n'\to m}| > \beta \\ \gamma \times \left(\left(\min_{n'\in N(m)\setminus n}|Z_{n'\to m}|\right)-\beta\right) & if \ \min_{n'\in N(m)\setminus n}|Z_{n'\to m}| \le \beta \end{cases}$$
$$with \qquad -1 \le \gamma \le 0 \qquad (2.9)$$

L'utilisation de (2.7) et (2.9), les nouvelles approximations des erreurs d'amplitudes inférieures à $\boldsymbol{\beta}$ sont estimées par

$$Er_2 = \gamma \times \left( \left( \min_{n' \in N(m)} |Z_{n' \to m}| \right) - \beta \right) - \left( \min_{n' \in N(m)} |Z_{n' \to m}| - \alpha \min_{n' \in N(m)} |Z_{n' \to m}| \right) \qquad (2.10)$$

Pour conserver les performances de décodage OMS, la nouvelle erreur d'approximation OMS $\boldsymbol{Er_2}$ ne doit pas dépasser l'erreur d'approximation OMS $\boldsymbol{Er_1}$. Au moins, nous devons avoir

$$Er_2 = Er_1 \qquad (2.11)$$

La constante négative $\boldsymbol{\gamma}$ peut être déduite en utilisant la valeur moyenne inférieure à $\boldsymbol{\beta}$ de $\boldsymbol{min_{n' \in N(m) \setminus n} |Z_{n' \to m}|}$ dans (2.11).

Laissez-nous estimer la valeur moyenne par

$$\left\langle \min_{n' \in N(m)} |Z_{n' \to m}| \right\rangle \cong \frac{\beta}{2} \qquad (2.12)$$

Avec l'utilisation de (2.12) dans (2.11), la valeur de la constante négative $\gamma$ sera obtenue par

$$\gamma = 2 \times (\alpha - 1) \qquad (2.13)$$

Si nous fixons $\boldsymbol{\alpha = 0.5}$, nous pouvons obtenir $\boldsymbol{\gamma = -1}$. Par conséquent, la nouvelle amplitude de la fonction $\boldsymbol{L_{m \to n}}$ peut être représentée comme suit

$$|L_{m\to n}| = \begin{cases} \left(\min_{n'\in N(m)\setminus n} |Z_{n'\to m}|\right) - \beta & if \ \min_{n'\in N(m)\setminus n} |Z_{n'\to m}| > \beta \\ \beta - \left(\min_{n'\in N(m)\setminus n} |Z_{n'\to m}|\right) & if \ \min_{n'\in N(m)\setminus n} |Z_{n'\to m}| \leq \beta \end{cases} \quad (2.14)$$

Par conséquent, le nouveau traitement des sorties des CNs de la nouvelle approche du OMS proposée sera donné par

$$L_{m\to n} = \left(\prod_{n'\in N(m)\setminus n} sgn(Z_{n'\to m})\right) \times \left|\left(\min_{n'\in N(m)\setminus n} |Z_{n'\to m}|\right) - \beta\right| \quad (2.15)$$

À partir de la comparaison de la nouvelle approche du décodage amélioré (2.15) avec le décodage OMS classique (Algorithme 5), nous pouvons affirmer que la mise en œuvre matérielle de la nouvelle CN nécessite moins de complexité matérielle.

## 2.5.1. Le nouvel algorithme I-OMS

Ce nouvel algorithme nous le nommons "Improved Offset Min-sum algorithm" (I-OMS), que nous dénotons "**Algorithme 6:** Décodage LDPC avec L' Improved Offset MIN-SUM algorithm ". L'algorithme 6 est décrit comme suit:

**Algorithme 6** Décodage LDPC Improved Offset MIN-SUM

Entrée: $y = (y_1, \dots, y_N)$

Sortie: $\hat{x} = (\hat{x}_1, \dots, \hat{x}_N,)$

**Initialisation:**

1. pour tout $n = 1, \dots, N$ faire $L_n = 4y_n/N_0$
2. pour tout $n = 1, \dots, N \; et \; m \in M(n)$ faire $Z_n = L_n$

**Itérations:**

3. Mise à jour des nœuds de parité

Pour tout $m = 1, \dots, M \; et \; n \in N(m)$ faire

$$L_{m \to n} = \left( \prod_{n' \in N(m) \setminus n} sgn(Z_{n' \to m}) \right) \times \left| \left( \min_{n' \in N(m) \setminus n} |Z_{n' \to m}| \right) - \beta \right|$$

$$avec \qquad \beta > 0$$

4. Mise à jour des nœuds de variable

Pour tout $n = 1, \dots, N \; et \; m \in M(n)$ faire

$$Z_{n \to m} = L_n + \sum_{m' \in M(n) \setminus m} L_{m' \to n}$$

5. Mise à jour des états des nœuds variable

Pour tout $n = 1, \dots, N$ faire

$$Z_n = L_n + \sum_{m \in M(n)} L_{m \to n}$$

6. Décision:

On détermine à partir de $Z_n$ la séquence décodée $\hat{x}$ .

Pour tout $n = 1, \dots, N$ faire $\hat{x}_n = \frac{1 - sgn(Z_n)}{2}$

Si l'équation de syndrome

$$\hat{x} \times H^T = 0$$

est vérifiée, le code est validé, sinon on retourne à l'étape 3.

Le code est déclaré incorrect si le nombre d'itérations atteint $It_{max}$.

---

L'unité de sortie des OMS CNs est représentée sur la figure. 2.1.a. Elle combine un soustracteur et un multiplexeur. La sélection du multiplexeur entre les deux entrées utilise le signal de sortie signe de la soustraction. L'unité améliorée supprime le multiplexeur et introduit un soustracteur absolu comme dans la figure 2.1.b. Par conséquent, le temps de réponse des sorties des CNs diminue à la suite de l'élimination des temps de propagation du signal signe de soustraction et des temps de réponse du multiplexeur.

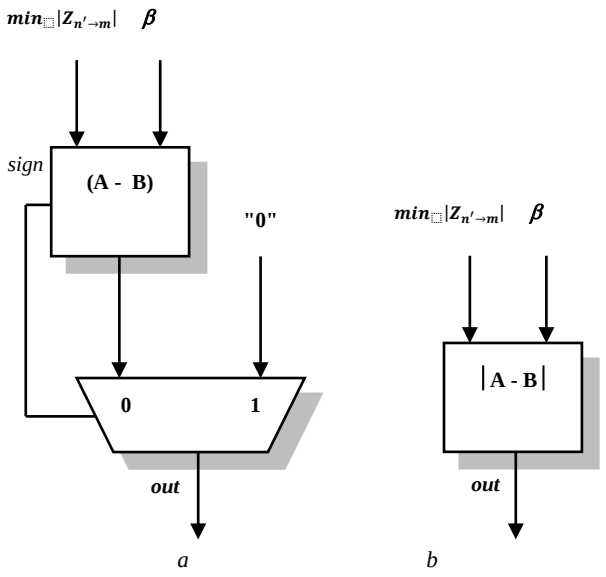


**Figure. 2.1** Structure des unités de sortie des CNs

a- Unité de sortie de la CN pour l'OMS classique, b- Nouvelle unité de sortie de la CN

La complexité de calcul de l'étape 3 de l'algorithme 6 est évaluée à $dc + (\log_2 dc) - 2$ additions (comparaisons) plus $2$ soustractions non signées.

## 2.5.2. Les résultats de simulation

Pour confirmer les performances du décodage de la nouvelle méthode, nous réalisons la comparaison avec la méthode MSA, avec la méthode OMS et avec la méthode bien connue SPA. Deux codes LDPC réguliers, (200, 100) et (1024, 512), sont utilisés sur un canal avec un bruit blanc gaussien additif (AWGN). Les deux codes ont un degré-3 pour les VNs et un degré-6 pour les CNs. Le nombre maximum d'itérations est fixé à 64 pour le code (200, 100) et à 32 pour le code (1024, 512).

La figure 2.2. montre les résultats de simulation MATLAB à virgule flottante de taux d'erreur de bit (BER) et le taux d'erreur de trame (FER) du code (200, 100) sur le canal AWGN avec une modulation BPSK. Les valeurs de l'offset $\beta$ peuvent être fixées au moyen de la procédure de Monte-Carlo.

Comme illustré sur la figure 2.2., le nouvel algorithme de décodage proposé réalise des performances comparables aux celles de l'algorithme de décodage SPA. De plus, le nouvel algorithme atteignait de meilleures performances au SNR supérieurs à 3 dB.

La figure 2.3. illustre la même comparaison avec code LDPC (1024, 512). Les résultats affichent visiblement les mêmes avantages. En outre, nous observons également des performances supérieures aux celles du SPA pour des SNR supérieurs à 2,25 dB.

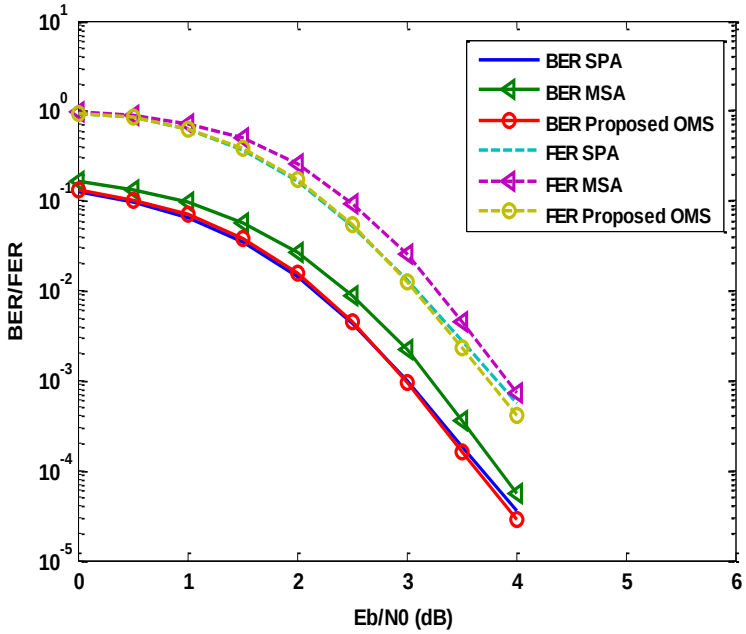


**Figure 2.2.** Comparaison des performances du MSA, SPA et OMS proposé avec le code LDPC (200, 100) sur le canal AWGN

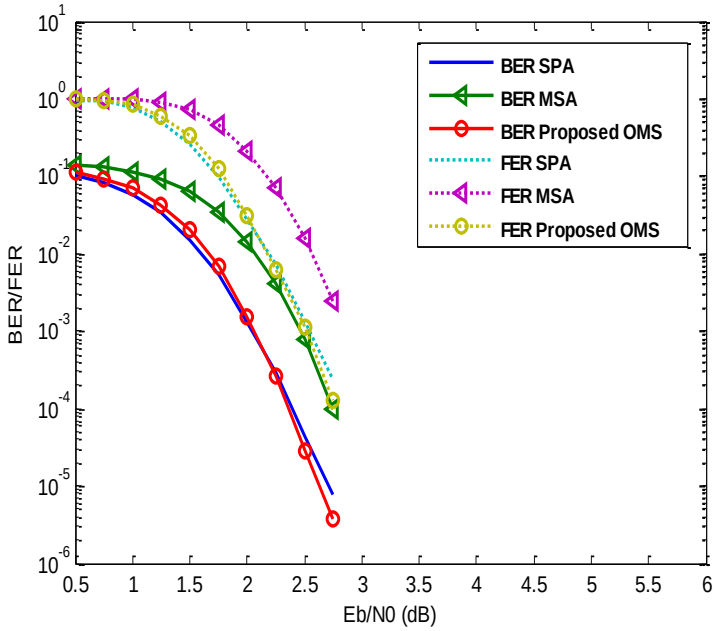


**Figure 2.3.** Comparaison des performances de l'OMS proposé, du SPA et du MSA avec le code LDPC (1024, 512) sur le canal AWGN

## 2.6. Conclusion

Dans ce premier chapitre, nous avons présenté dans la première partie, les principes et les architectures des décodeurs LDPC à complexité réduite et leurs algorithmes associés. Le premier décodage LDPC non stochastique à complexité réduite est le décodage avec Le "MIN-SUM algorithm" (MSA). La deuxième variante est le décodage avec le "Normalized MIN-SUM algorithm" (NMS). La troisième variante présentée est la variante du décodage avec l'"Offset MIN-SUM algorithm" (OMS).

Une nouvelle amélioration de l'algorithme de décodage OMS a été présentée. L'algorithme proposé est conçu pour réduire la complexité matérielle tout en conservant les performances du décodage. Le nouvel algorithme a été appliqué en utilisant deux différents codes LDPC réguliers (200, 100) et (1024, 512). Les résultats des simulations montrent que les performances du nouvel algorithme sont proches du SPA à virgule flottante pour un faible rapport signal sur bruit (SNR) et peut surpasser le SPA à virgule flottante pour un SNR élevé, avec une complexité inférieure à la complexité du OMS classique.

# Chapitre 3

# NOUVEAU DECODAGE LDPC STOCHASTIQUE SUR FPGA

## 3.1. Introduction

Les variantes du décodage stochastique présentent une alternative réaliste est moins complexe. Dans ce chapitre, les récentes techniques du décodage stochastique des codes LDPC sont traitées avec une nouvelle méthode et nouvelle perspective. Nous commençons par la présentation des principes du calcul stochastique, la représentation de l'information stochastique et les opérations arithmétiques.

La deuxième partie de ce chapitre est réservée pour une nouvelle étude du décodage LDPC stochastique sur le plan architectural et sur le plan algorithmique. Les approches EM, TFM, MTFM et DS sont exposées.

La troisième partie présente la conception d'une nouvelle approche de décodage LDPC stochastique très performante, complètement parallèle et implémentable sur FPGA. Les détails de cette architecture seront bien décrits, ainsi que les calculs et les étapes de l'algorithme associé.

## 3.2. Principe du calcul stochastique

### 3.2.1. Séquence de Bernouli

Avant de réaliser des opérations sur le domaine stochastique, nous devons premièrement convertir les variables d'entrées, c'est-à-dire les probabilités d'entrées, en séquences discrètes aléatoires des valeurs numériques {*ai*} où *i* = 1...m. La valeur de *m* représente la longueur de la séquence. Cette séquence est dite la séquence de Bernouli. Lors de sa génération, la probabilité du symbole "1" est équivalente à la probabilité [34].

Si dans une séquence de Bernouli de longueur m = 100 bits, 32 bits possèdent un état haut "1", alors la probabilité correspondante est $\mathbf{P_r = 0,32}$. Également, si 12 bits sont à "1" dans une séquence de Bernouli de longueur m = 20 bits, la probabilité est estimée à $\boldsymbol{P_r = 0,6}$. Toutefois, plusieurs séquences peuvent schématiser la même probabilité. La figure 3.1 expose plusieurs séquences de 20 bits schématisant la même probabilité ***0.6***.

1 0 0 0 1 1 1 0 0 0 1 0 0 0 1 1 0 0 1 0 1 1 0 0 0 1 1 0 0 0

0 0 1 0 0 1 0 0 0 1 1 0 0 1 0 1 1 1 1 0 0 1 0 0 0 0 1 0 0 1 1

0 1 0 0 1 1 0 0 1 0 1 1 0 0 0 1 1 0 0 0 1 0 0 0 1 1 0 0 1 0

1 0 1 0 0 1 0 0 1 0 1 0 0 1 0 1 0 0 0 1 1 0 1 0 0 1 0 1 0 0

1 0 0 1 0 1 0 0 0 0 1 0 0 1 0 1 0 0 1 0 1 0 0 1 0 1 0 1 0 1

**Figure 3.1.** Représentation d'une probabilité $\boldsymbol{P_r = 0,6}$ sur une séquence de Bernouli de longueur m = 20 bits

Différentes séquences peuvent être générées pour la même probabilité. Dans une séquence de Bernouli ***{ai}*** de ***m*** bits, dont $a_i \in \{0, 1\}$, la probabilité $P_r$ estimé est calculée par l'expression suivante.

$$P_r(m) = \frac{\sum_{i=1}^{m} a_i}{m} \tag{3.1}$$

Un générateur de valeurs aléatoires selon une distribution uniforme associé à un comparateur peuvent être utilisé, afin de convertir une probabilité en séquence de Bernouli (flux stochastique binaire). Le circuit de conversion est illustré sur la figure 3.2. Le comparateur possède une sortie logique et deux entrées de bus de même taille. Une de ces deux entrées, ***R***, est reliée à la sortie du générateur de valeurs aléatoires. La deuxième est reliée à $P_r$ . La sortie du comparateur est égale à "1" si $P_r$ est supérieure à ***R***. Elle est égale à "0" dans le cas contraire. Dans ce cas, la probabilité $P_r$ est mise à l'échelle à ***m*** bits de largeur.

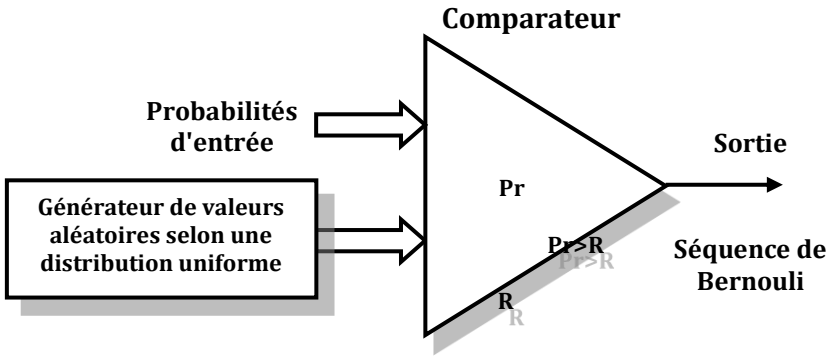


**Figure 3.2.** Génération de séquences de Bernouli en utilisant un comparateur et un générateur aléatoire à distribution uniforme.

## 3.2.2. Opérations arithmétiques stochastiques sur les probabilités

L'approche stochastique donne la possibilité de réaliser des opérations arithmétiques sur les probabilités telles que la multiplication, l'addition ou la division avec des opérations logiques simples utilisant des circuits logiques combinatoires et séquentielles. Si deux séquences stochastiques de Bernouli {ai} et {bi} symbolisent les probabilités d'entrée Pa et Pb, avec Pa = Pr(ai = 1) et Pb = Pr(bi = 1), la séquence de Bernouli {ci} symbolise la probabilité de sortie Pc, avec Pc = Pr(ci = 1). Pc est le résultat de l'opération arithmétique stochastique entre les entrées Pa et Pb.

La multiplication :

La multiplication stochastique de deux probabilités est effectuée par une porte logique AND. Nous pouvons facilement déduire de la table de vérité de l'opération AND, que la sortie est égal à "1" si les deux entrées sont à "1". La circuit de calcul Pc, avec de Pc = Pr(C = 1) = Pr(A) ∗ Pr(B) est présentée par la figure 3.3. Le résultat est plus valable si les séquences d'entrée stochastiques de Bernouli sont totalement indépendantes et suffisamment longues .

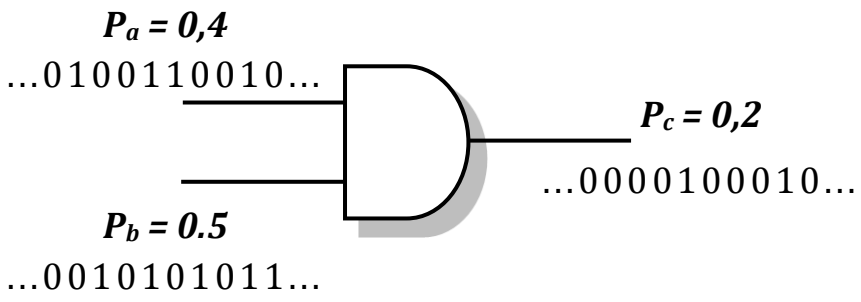


**Figure 3.3.** Multiplication de deux séquences de Bernouli.

Pour réaliser une multiplication d'une séquence de Bernouli avec elle-même Pc =Pa∗Pa=$Pa^2$ , il faut décorréler les deux séquences d'entrée. La décorrélation peut être effectuée par une introduction d'un retard sur la séquence {ai} d'une ou de plusieurs tops d'horloge. Nous pouvons utiliser une ou plusieurs bascules du type D.

La division :

La division entre deux valeurs numériques est relativement complexe. Généralement, il est impossible d'exécuter une opération de division sur une seule période d'horloge. C'est aussi le cas de la division stochastique entre deux séquences de Bernoulli. Néanmoins, des solutions adaptés, basées sur des structures simples sont proposées. Une de ces solutions est de considérer la division entre les deux séquences de Bernoulli {ai} et {bi} avec Pa≪Pb, alors la division Pc est approximée par :

$$P_c = \frac{P_a}{P_b} \approx \frac{P_a}{P_a + P_b} \tag{3.2}$$

Cette opération peut être exécutée sur une seule période d'horloge, avec une bascule JK simple. Le principe est illustré par la figure 3.4. Le bit sortant de la bascule est égal au bit entrant J si les deux entrées J et K sont distinctes. Si les deux bits d'entrées sont égaux à "0", la sortie retient l'état précédent. Et si les deux bits d'entrées sont égaux à "1", la sortie retient l'inverse de l'état précédent.

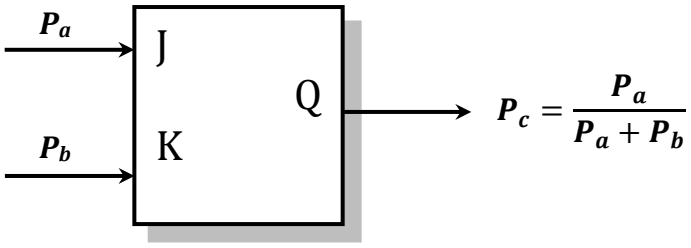


**Figure 3.4.** Division stochastique.

L'addition :

L'addition stochastique de deux séquences de Bernoulli est une opération très différente de la multiplication et de la division car le résultat n'est pas obligatoirement inclus dans l'intervalle de probabilité [0, 1]. De ce fait, il n'est pas possible de les exécuter indépendamment d'une opération de mise à l'échelle. Une addition avec la mise à l'échelle est très simple à effectuer, sur des séquences stochastiques. Un multiplexeur qui sélectionne aléatoirement une entrée donnée $\boldsymbol{k}$ avec une certaine probabilité $\boldsymbol{S_k}$ telle que

$$\sum_k S_k = 1 \tag{3.3}$$

La sortie $\boldsymbol{P_c}$ générée suit une probabilité $\boldsymbol{P_r(P_c)}$ qui est une somme échelonnée des probabilités d'entrée $\boldsymbol{P_r(A_k)}$. Un tel multiplexeur est représenté sur la figure 3.5, où RS est une sélection aléatoire et est fournie par un générateur de valeurs aléatoires selon une distribution uniforme. Le résultat de sortie est donnée par

$$P_r(P_c) = \sum_{k=1}^{N} S_k * P_r(A_k) \tag{3.4}$$

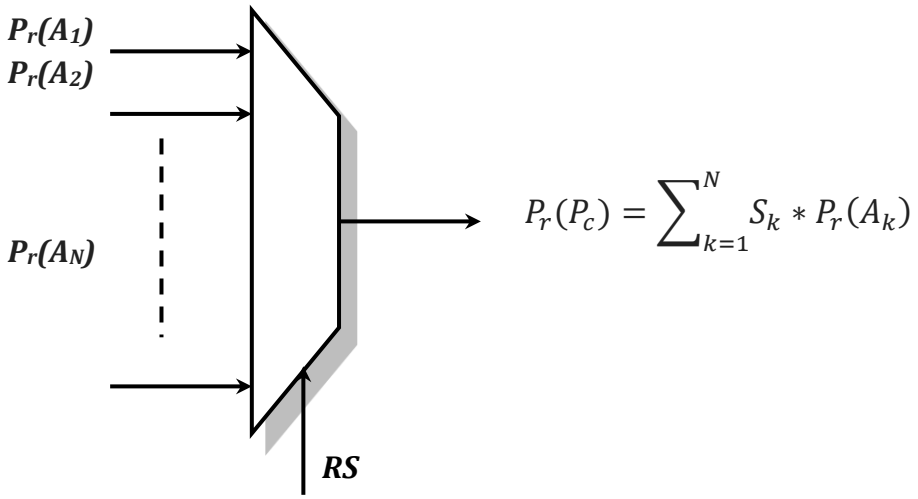


**Figure 3.5.** Additionneur stochastique.

## 3.3. Le décodage LDPC stochastique

Le décodage stochastique des codes LDPC débute par la conversion des bits reçus en des séquences de Bernoulli. Les figures 3.6. et 3.7 présentent les architectures stochastiques des nœuds de variable et de parité. Cet avantage simplifie considérablement la complexité des nœuds et réduit également le problème de placement et de routage. Pendant chaque itération, le mécanisme de décodage stochastique effectue des échanges d'un bit entre un nœud de variable et un nœud de parité [35]-[43].

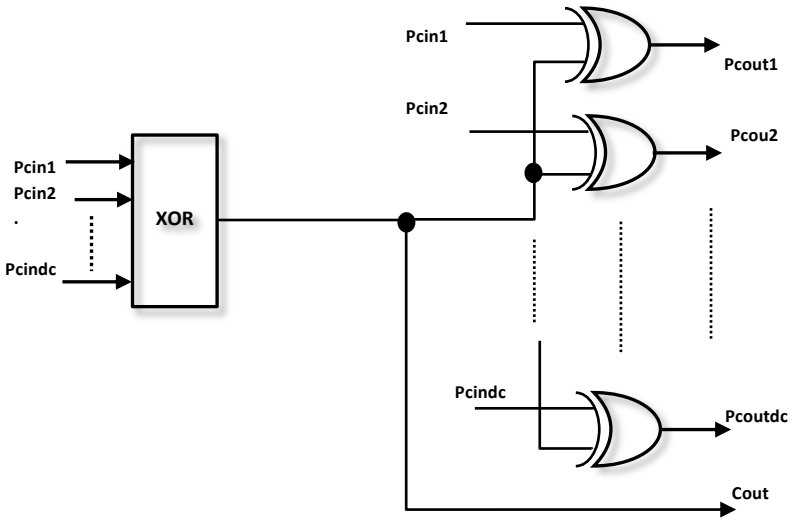


**Figure 3.6.** La structure du nœud de parité

Différentes séquences peuvent être générées pour la même probabilité. Dans une séquence de Bernouli ***{ai}*** de ***m*** bits, dont $a_i \in \{0, 1\}$, la probabilité estimée est calculée par l'expression suivante.

$$P_{ch}(m) = \frac{\sum_{i=1}^{m} a_i}{m} \tag{3.5}$$

Pour ***Cout***, ***Pcout$_i$*** et ***Pcin$_i$*** présentant les sorties des nœuds de parité, les probabilités de sorties et les probabilités d'entrées des nœuds de parité respectivement, dont ***Pcin$_i$ = Pr(cin$_i$=1)*** est la probabilité de chaque entrée CN, avec ***$i \in \{1, 2, ....., dc\}$*** et ***dc*** est le degré de CN. La probabilité de sortie ***Pcout$_i$*** est calculée par l'expression suivante.

$$\left.\begin{aligned} Pcout_1 &= Pcin_2 \oplus \ldots \oplus Pcin_{dc} \\ Pcout_l &= Pcin_1 \oplus \ldots \oplus Pcin_{l-1} \oplus Pcin_{l+1} \oplus \ldots \oplus Pcin_{dc} \\ Pcout_{dc} &= Pcin_1 \oplus \ldots \oplus Pcin_{dc-1} \end{aligned}\right\} \quad (3.6)$$

L'état du nœud de parité ***Cout*** est calculée par :

$$Cout = Pcin_1 \oplus \ldots \oplus Pcin_{dc} \quad (3.7)$$

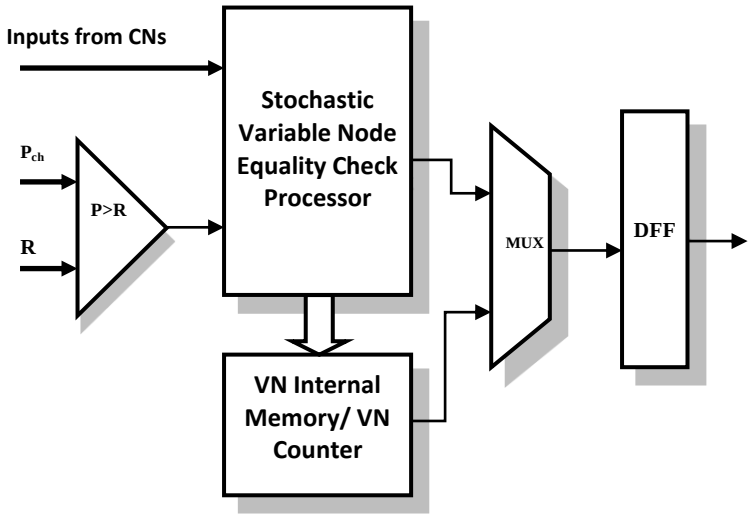


**Figure 3.7.** Structure du nœud de variable stochastique récent

***Pvin$_1$*** et ***Pvin$_2$*** sont les probabilités d'entrées dans VN avec ***dv=2***. La probabilité de sortie *Pvout* est calculée par :

$$Pvout = \frac{Pvin_1 . Pvin_2}{Pvin_1 . Pvin_2 + (1 - Pvin_1).(1 - Pvin_2)} \tag{3.8}$$

Si les entrées de VN sont identiques, cet état est appelé état d'accord "the agreement state", l'un des bits d'entrée sera transmis à la sortie. Lorsque les entrées ne sont pas identiques, le nœud variable nécessite une méthode avancée pour générer le bit de sortie. Cet état est appelé état de blocage ou état de désaccord. Une des méthodes stochastiques avancées de génération de bits (ASMBG) peut être utilisée.

$$Pvout(N) = \begin{cases} \dfrac{Pvin_1 . Pvin_2}{Pvin_1 . Pvin_2 + (1 - Pvin_1).(1 - Pvin_2)} & if\ Pvin_1 = Pvin_2 \\ Bit\ using\ \text{ASMBG} & otherwis \end{cases} \tag{3.9}$$

La figure 3.7. présente la structure principale de nœud de variable stochastique récente. L'algorithme de décodage LDPC stochastique peut être résumé comme suit

**Algorithme 7** Décodage LDPC stochastique

Entrée: $y = (y_1, \dots, y_N)$

Sortie: $\hat{x} = (\hat{x}_1, \dots, \hat{x}_N,)$

**Initialisation:**

1. Pour tout $n = 1, \dots, N$
   Charger les probabilités Pch correspondantes à $L_n$ (un DC)
   Transformer Pch en séquence de Bernouli ai (chaque DC).

2. Initialisez les mémoires internes des nœuds de variable (16 à 32 DC pour la mémoire 32 bits) ou le compteur interne à saturation (un DC).

Itérations:

3. Du nœud de variable vers nœud de parité:

A chaque cycle de décodage, le nœud de variable calcule ses bits d'entrée en utilisant

$$Pvout(N) = \begin{cases} \dfrac{Pvin_1 . Pvin_2}{Pvin_1 . Pvin_2 + (1 - Pvin_1).(1 - Pvin_2)} & if\ Pvin_1 = Pvin_2 \\ Bit\ using\ \text{ASMBG} & otherwis \end{cases}$$

et envoie ses bits de sortie aux nœuds de parité correspondants.

4. Du nœud de parité vers nœud de variable:

A chaque cycle de décodage, le nœud de contrôle calcule ses bits d'entrée en utilisant

$$\left.\begin{array}{l} Pcout_1 = Pcin_2 \oplus \ \ldots\ldots\ldots\ldots \oplus\ Pcin_{dc} \\ Pcout_l\ = Pcin_1 \oplus .. \oplus\ Pcin_{l-1} \oplus Pcin_{l+1} \oplus .. \oplus Pcin_{dc} \\ Pcout_{dc} = Pcin_1 \oplus \ldots\ldots\ldots\ldots \oplus\ Pcin_{dc-1} \end{array}\right\}$$

et envoie les résultats des sorties aux nœuds de variable correspondants.

Simultanément, les nœuds de contrôle envoient des états de sortie en utilisant

$$Cout = Pcin_1 \oplus . \ . \ . \oplus\ Pcin_{dc}$$

vers le vérificateur de syndrome.

5. Décision:

Si $\hat{x} \times H^T = 0$

ou

si le maximum de DC est atteint,

terminez le processus de décodage. Sinon, passez à l'étape 3.

## 3.4. L'Approche EM

(Edge Memories - EMs en anglais) on garde les mêmes nœuds de parité , mais les bascules Jk sont remplacées par des mémoires séries [36]-[37]. Chaque EM est un registre à décalage de M bits et est introduit pour chaque branche des nœuds de variable. L'architecture de la technique de l'approche EM est exposée sur la figure 3.8.

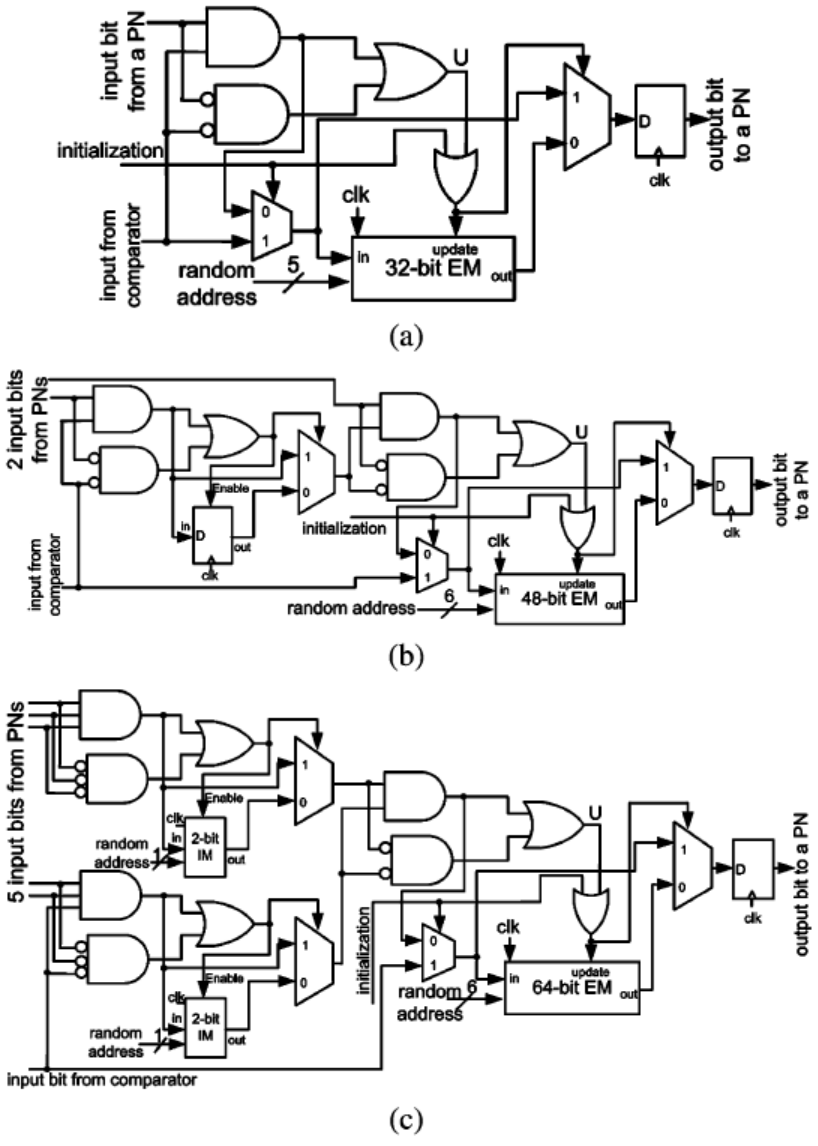


**Figure 3.8.** Architectures des Nœuds de variable à 2, 3 et 6 degrés de la technique EM.

## 3.5. L'Approche MTFM

Pour l'approche MTFM on garde toujours les mêmes nœuds de parités, mais un groupe de bascules JK (ou mémoires EM) est remplacé par 1 une seule mémoire [38]-[41].

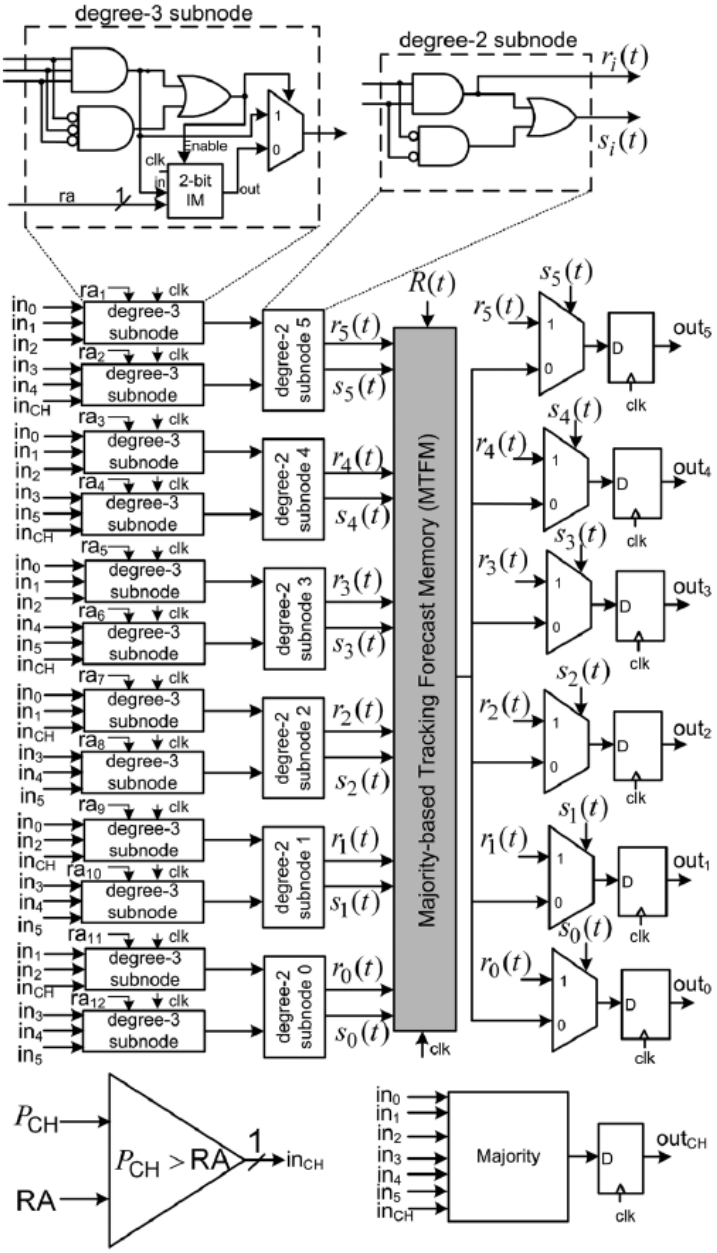


**Figure 3.9.** Architectures des Nœuds de variable à 6 degrés de l'approche MTFM.

## 3.6. L'Approche DS

En 2011, Naderi et al proposent une nouvelle approche. Cette méthode est basée sur un décodage stochastique différé (Delayed Stochastic DS decoding). Les nœuds de parité proposées réalisent un traitement direct et différé, en fonction des états des nœuds de variable [42]. La figure montre le principe des nœuds de parité proposées.

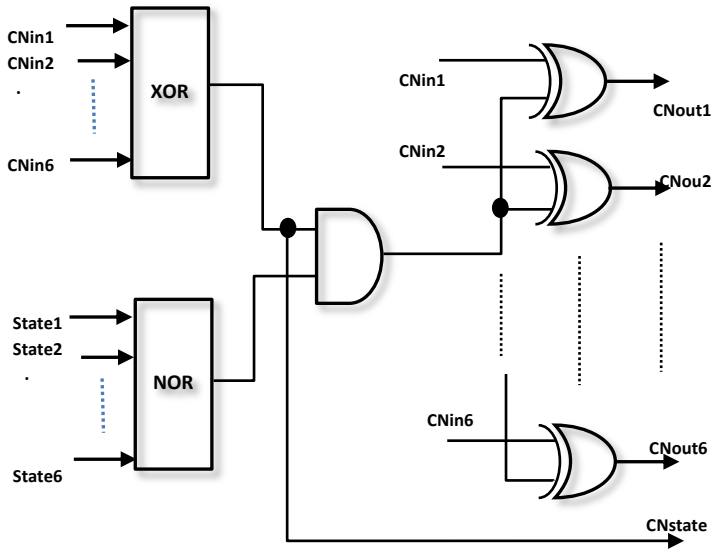


**Figure 3.10.** Nœud de parité à 6 degrés.

Dans cette méthode, les nœuds de variable produisent à la sortie un état associé à la valeur de sortie. La figure 3.10. montre le principe des nœuds proposés. Les mémoires EM, TFM et MTFM sont remplacées par des mémoires à deux (02) bits dans les nœuds à 3 degrés et par un bit aléatoire dans les nœuds à deux degrés. Un nœud à 6 degrés est composé de deux nœuds à 3 degrés et un nœud à 2 degrés.

Le principe du comparateur et du générateur des variables aléatoires est toujours gardé. Un détecteur d'état majeur est ajouté dans les nœuds de variables.

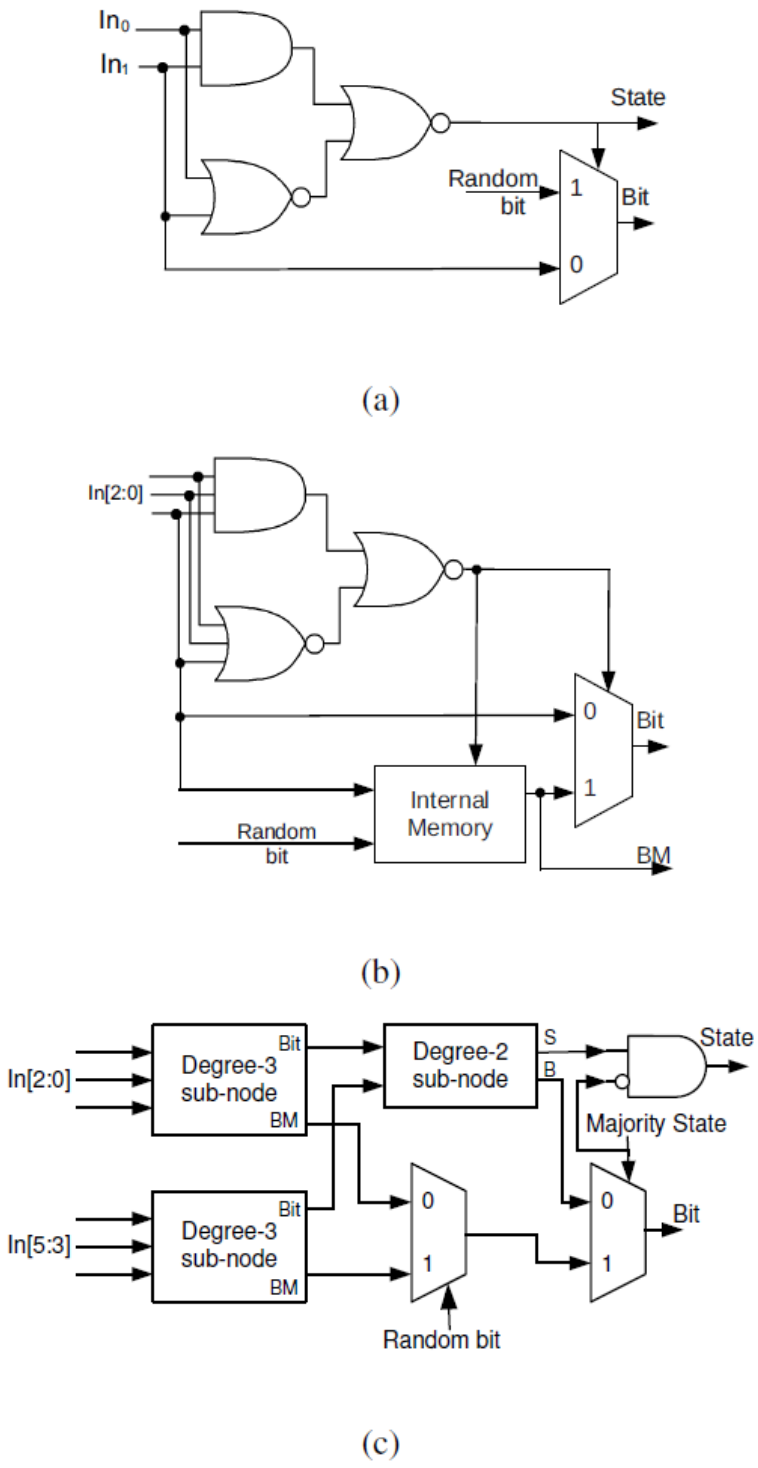


**Figure 3.11.** Architectures des Nœuds de variable à 2, 3 et 6 degrés de la technique du décodage stochastique différé (Delayed Stochastic DS decoding).

## 3.7. Conception d'une nouvelle architecture LDPC stochastique

Il a été prouvé que l'initialisation du premier bit de sortie des VNs, basée sur la probabilité reçut du canal, améliore la convergence du décodeur LDPC stochastique [44]. En outre, il a été montré que le processus de décodage stochastique LDPC, où les VN utilisent les derniers bits de sortie comme bits de code, offre des performances de BER comparables à la version avec compteur-décompteur à saturation [45]-[46]. Dans cette section nous proposons une nouvelle structure pour les nœuds de variable. Le nœud de variable proposé exploite les deux caractéristiques référencées, en plus d'adopter une configuration basée sur la mémoire interne, analogue aux versions DS et EM.
La probabilité de sortie des VNs sera calculée comme suit

$$Pvout(t) = \begin{cases} Pst & if\ t = 1 \\ Pa\ or\ Pb & if\ t \neq 1\ \&\ Pa = Pb \\ Bit\ from\ FPGA\ LUT\ RAM & otherwis \end{cases} \quad (3.10)$$

$$\text{where} \qquad Pst = \begin{cases} 1 & if\ Pch \geq 0.5 \\ 0 & otherwise \end{cases}$$

L'initialisation des VNs proposées est achevée à (t = 1). Dans l'état de désaccord (t ≠ 1), la nouvelle architecture génère un nouveau bit en lisant de manière aléatoire la mémoire interne de VN (IM). La longueur de la séquence de Bernouli peut être augmentée jusqu'à la longueur de la RAM-LUT du FPGA. Sur la base de (3.10), l'implémentation matérielle de la nouvelle structure de décodage n'appelle pas une utilisation supplémentaire des ressources logiques du FPGA. De plus, la technique proposée calcule la probabilité reçue sans cycle de décodage supplémentaire.

Le bit Pch (k-1) est transmis à la bascule de sortie de la VN, au cours du premier cycle. Après le premier cycle et jusqu'au dernier, le multiplexeur envoie le

bit de sortie du processeur de nœud variable à la bascule de sortie de la VN. De cette manière et de manière comparable au processus du CSS, la plus grande partie des sorties des nœuds de variable commencent par un bit correct et évite l'initialisation stochastique conventionnelle. La figure 3.12. représente l'organisation principale du nouveau nœud de variable.

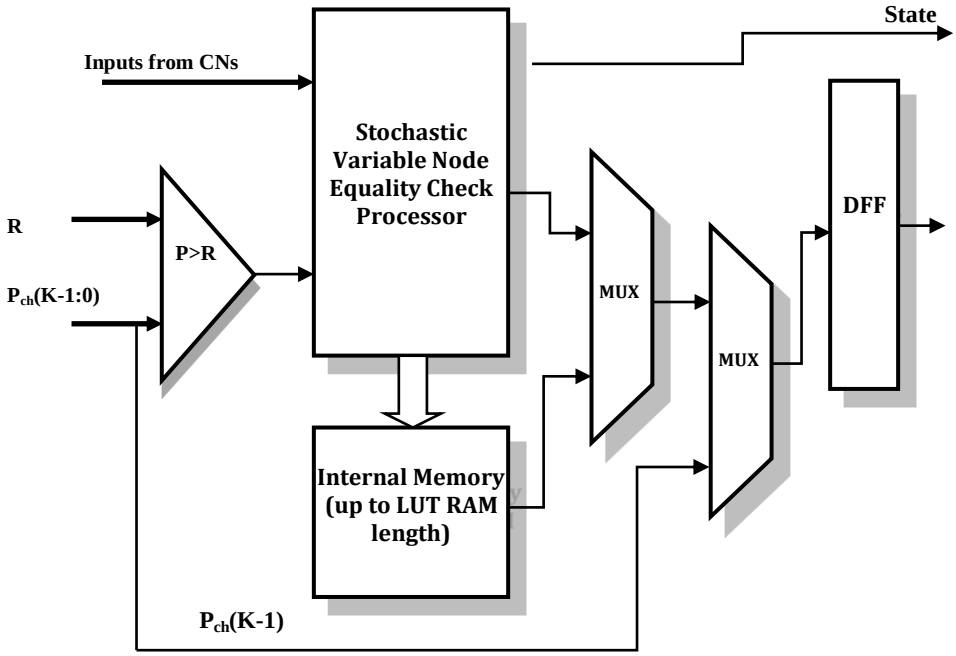


**Figure 3.12.** La structure du VN stochastique proposé.

Le CN proposé a deux types d'entrées et deux de sorties. Les entrées sont les signaux $CNin_i$ et les signaux $State_i$, dans lesquels $i \in \{1, 2, ....., dc\}$. Les entrées sont accouplées aux signaux de sortie des VNs. Les premiers signaux de sortie sont les $CNout_i$, qui sont envoyés aux entrées des VNs. La deuxième sortie est l'état de sortie de contrôle de parité CN. Les sorties CNstate sont connectées à l'unité de vérification de syndrome. L'état de sortie de contrôle de parité CNstate, du CN de degré dc, est calculé avec la même méthode en utilisant (3.7) et peut s'écrire comme suit.

$$CNstate = \sum_{i=1}^{dc} \oplus CNin_i \tag{3.11}$$

où $\sum \oplus$ est l'opération XOR bit à bit et dc est le degré du nœud de parité.

Chaque signal de sortie $CNout_i$ des CNs utilise les signaux $CNin_i$ et les signaux $State_i$, pour générer les signaux $CNout_i$ selon l'expression (8). Le résultat CNstate donné par (3.11) peut- être exploité.

$$CNout_i = \left( (CNstate) \wedge \overline{\left( \bigvee_{i=1}^{dc} state_i \right)} \right) \oplus CNin_i \tag{3.12}$$

où $\overline{\vee}$ est l'opération NOR au niveau du bit et $\wedge$ est l'opération AND au niveau du bit.

L'algorithme de décodage LDPC stochastique proposé peut être décrit comme suit

**Algorithme 8** Nouveau décodage LDPC stochastique

Entrée: $y = (y_1, \dots, y_N)$
Sortie: $\hat{x} = (\hat{x}_1, \dots, \hat{x}_N,)$

**Initialisation:**

1. Pour tout $n = 1, \dots, N$ Charger les probabilités Pch correspondantes à $L_n$ simultanément avec l'initialisation des sorties des nœuds de variable (un DC) Transformer Pch en séquence de Bernouli ai (chaque DC).

**Itérations:**

2. Du nœud de variable vers nœud de parité:

A chaque cycle de décodage, le nœud de variable calcule ses bits d'entrée en utilisant

$$Pvout(t) = \begin{cases} Pst & if\ t = 1 \\ \dfrac{Pvin_1 . Pvin_2}{Pvin_1 . Pvin_2 + (1 - Pvin_1).(1 - Pvin_2)} & if\ t \neq 1\ \&\ Pvin_1 = Pvin_2 \\ Bit\ from\ FPGA\ LUT\ RAM & otherwis \end{cases}$$

$$\text{where} \qquad Pst = \begin{cases} 1 & if\ Pch \geq 0.5 \\ 0 & otherwise \end{cases}$$

et envoie ses bits de sortie aux nœuds de parité correspondants.

3. Du nœud de parité vers nœud de variable:

A chaque cycle de décodage, le nœud de contrôle calcule ses bits d'entrée en utilisant

$$CNout_i = \left( (CNstate) \wedge \overline{\left( \bigvee_{i=1}^{dc} state_i \right)} \right) \oplus CNin_i$$

et envoie les résultats des sorties aux nœuds de variable correspondants. Simultanément, les nœuds de contrôle envoient des états de sortie en utilisant

$$CNstate = \sum_{\mathrm{i=1}}^{\mathrm{dc}} \oplus CNin_i$$

vers le vérificateur de syndrome.

4. Décision:

Si $\hat{x} \times H^T = 0$

ou

si le maximum de DC est atteint,

terminez le processus de décodage. Sinon, passez à l'étape 2.

## 3.8. Conclusion

Dans ce chapitre, nous avons introduit les concepts de base du calcul stochastique, la représentation de l'information stochastique et les opérations arithmétiques. De plus et dans la deuxième partie une nouvelle étude du décodage LDPC stochastique sur le plan architectural et sur le plan logarithmique est présentée. Les approches EM, TFM, MTFM et DS sont rappelées.

La conception d'une performante approche de décodage LDPC stochastique, complètement parallèle et implémentable sur FPGA a été illustrée avec les détails de cette architecture, ainsi que les calculs et les étapes de l'algorithme associé. De ce fait, le suivant chapitre sera consacré à l'implémentation FPGA pour validation et comparaison.

# Chapitre 4
# RESULTATS ET PERFORMANCES DU NOUVEAU DECODAGE LDPC STOCHASTIQUE SUR FPGA

## 4.1. Introduction

Il a été clairement prouvé qu'un débit plus élevé est atteint par des solutions de décodage entièrement parallèles, cependant elles augmentent les utilisations logiques du FPGA et la complexité matérielle. Plusieurs implémentations de décodage LDPC ont été explorées pour atteindre des résultats à haut débit. De nombreuses architectures de décodage LDPC stochastique et à complexité réduite sont élaborées, pour contourner ce désagrément. Les récentes approches du décodage LDPC stochastique ont confirmé leurs adaptabilités pour le décodage parallèle. De plus, pour une réduction supplémentaire de la surface de silicium, différentes architectures et stratégies de décodage LDPC, utilisant les méthodes stochastiques, sont proposées.

Néanmoins, une architecture optimisée pour un décodeur LDPC stochastique basé sur ASIC ne peut pas générer systématiquement une utilisation efficace de logique FPGA. Il est évident que la mise en œuvre ASIC du compteur à six bits nécessite moins de surface de silicium par rapport à la mémoire de 64 bits. Cependant, un résultat contraire est obtenu par la mise en œuvre sur un FPGA. Une implémentation d'une mémoire de 64 bits sur un FPGA de Xilinx peut être routée en utilisant une seule LUT, contrairement à la mise en œuvre du compteur à six bits.

Dans ce chapitre, nous présentons les résultats d'implémentation et les comparaisons entre les architectures de décodage LDPC stochastique basé sur un FPGA. De plus, nous présentons la validation de notre technique qui permet une réduction supplémentaire de l'utilisation de la logique FPGA et une amélioration de la convergence, même pour les codes courts. L'architecture mise en œuvre de notre décodeur LDPC est entièrement parallèle. Des descriptions VHDL sont données en annexes.

## 4.2. Implémentation sur FPGA

Il a été démontré que l'élargissement de la taille des mémoires internes des VNs augmente les convergences de décodage stochastique LDPC [36]-[37]. Cependant et surtout, l'ajout de capacité de mémoire supplémentaire implique une complexité matérielle additionnelle et des ressources supplémentaires. L'organisation et la mise en œuvre sur FPGA nécessitent des considérations spéciales. En plus des ressources logiques des LUTs, la mémoire peut être mappée à l'aide des blocs RAM ou à l'aide des RAM distribuées (LUT RAM).

La cible principale de la structure proposée est d'améliorer les performances de décodage LDPC sur FPGA, sans ressources FPGA supplémentaires. Pour valider l'amélioration de notre nouvelle conception, des codes LDPC moyens (1024, 512) et courts (200, 100) sont implémentés sur FPGA (Field Programmable Gate Array) de Xilinx, le Virtex-6, avec différentes méthodes. La figure 4.1. présente le synoptique des connexions pour l'association des sous-nœuds de variable à 3 degrés [47]-[49].

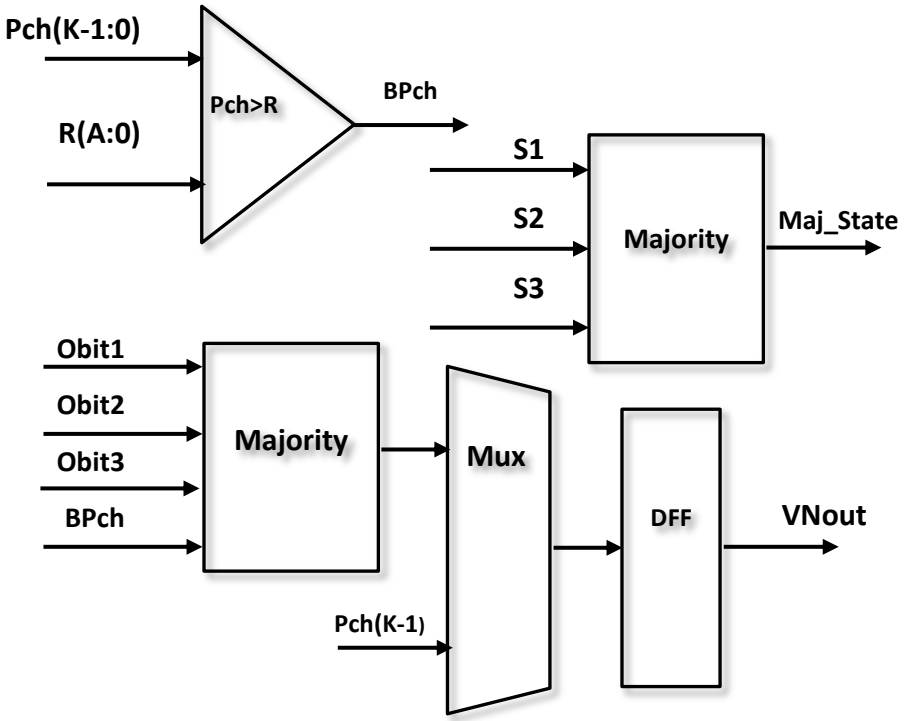


**Figure 4.1.** Synoptique des connexions pour l'association des sous-nœuds de variable à 3 degrés

La figure 4.2. présente le diagramme fonctionnel du nouveau sous-nœud de variable des codes LDPC proposés (1024, 512) et (200, 100). Le nœud de variable de degré 3 est composé de 3 sous-nœuds de degrés 3. L'état majoritaire et les signaux d'adresse aléatoire sont connectés à tous les sous-nœuds. L'une des entrées du sous-nœud à 3 degrés 3 est connectée au signal de probabilité par un comparateur. Les deux autres entrées du sous-nœud sont connectées à la sortie du nœud de parité. Les trois signaux S sont combinés pour produire le signal d'état majoritaire. L'implémentation sur FPGA de la mémoire interne de la VN proposée est réalisée en utilisant les RAM distribuées du FPGA.

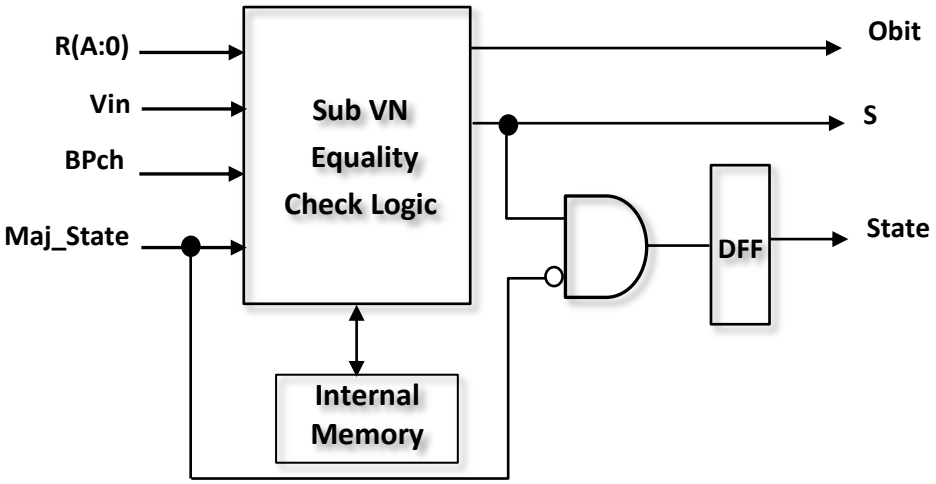


**Figure 4.2.** Structure du nouveau sous-nœud de variable.

La figure 4.3. montre le principe de l'implémentation FPGA d'une partie qui compose les nœuds de variable proposés de degré 3. Le nœud de variable proposé est composé de trois parties. Cette variante du nœud de variable utilisera trois (03) RAM distribuées du FPGA. Chaque RAM distribuées du FPGA est l'équivalent d'une seule LUT. Le principe du comparateur pour produire les séquences de Bernouli est toujours gardé. De même, nous gardons le principe du générateur des variables aléatoires. Le générateur est utilisé d'une part pour piloter le bus d'adresses des mémoires internes des nœuds de variable, et d'autre part pour produire les deuxièmes entrées des comparateurs.

Les CNs implémentés utilisent une structure similaire à celle des CNs adoptés par les décodeurs DS et CSS. La figure 4.4. donne la structure principale d'un CN de degré-6.

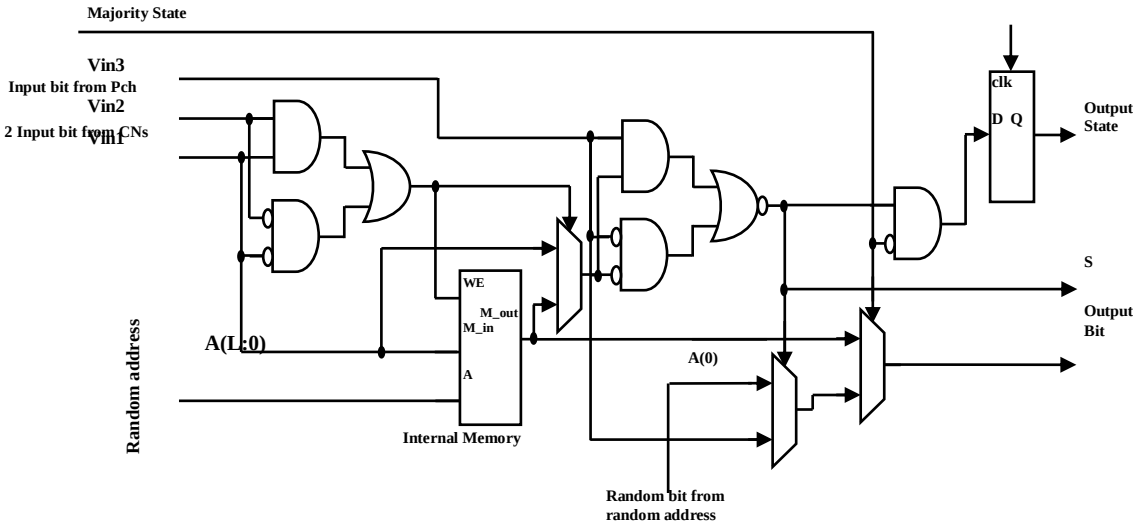


**Figure 4.3.** Implémentation de VN avec dv=3, Nouvelle approche.

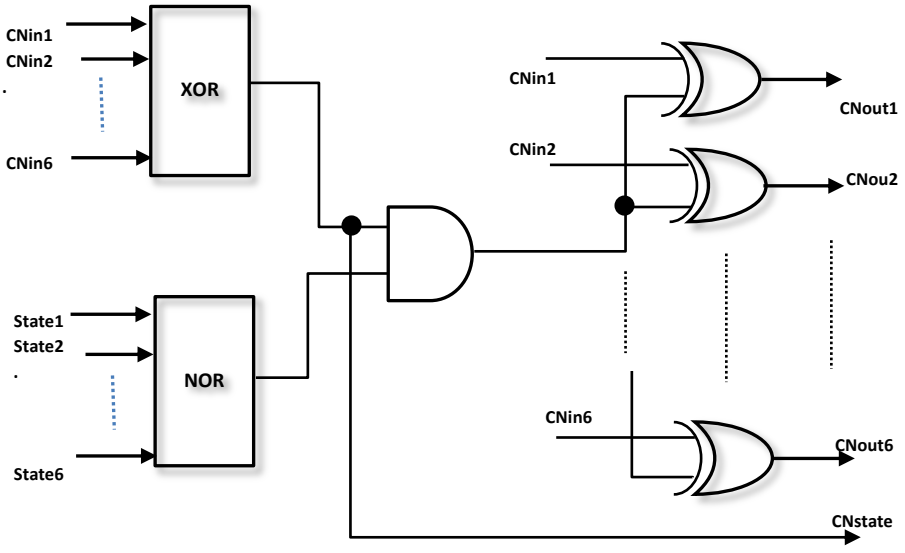


**Figure 4.4.** Nœud de parité à 6 degrés.

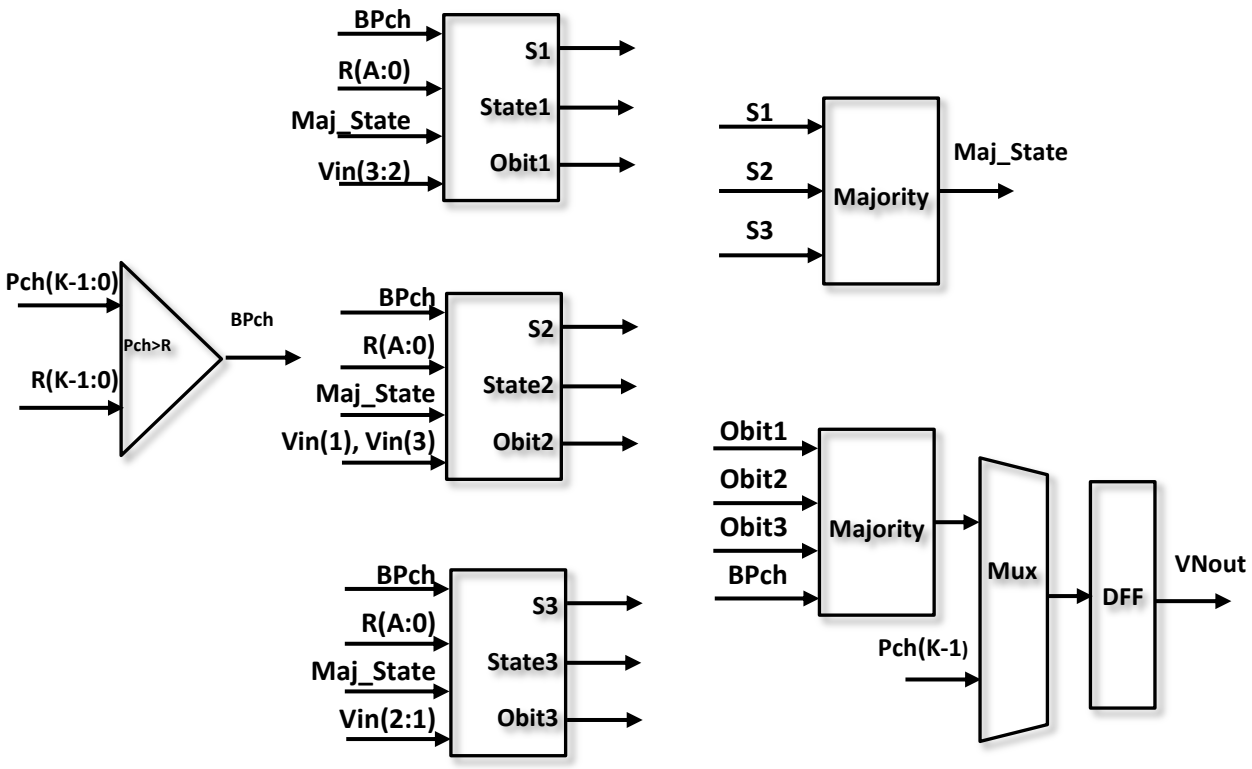


**Figure 4.5.** Structure du nouveau nœud de variable à 3 degrés

Le décodeur LDPC stochastique avec la version EM donne un résultat d'implémentation FPGA proche de la version DS. L'implémentation de 2 × 1 bits jusqu'à 64 × 1 bits n'utilise qu'un seul LUT sur le FPGA Xilinx Virtex-6. Par conséquent, la version de décodeur LDPC proposée peut être implémentée en utilisant une mémoire interne d'une VN allant jusqu'à 64 bits sans réclamer des ressources FPGA supplémentaires.

Comme nous pouvons le voir, l'implémentation sur FPGA du décodeur basé sur les compteurs avec one-step initialisation nécessite des ressources supplémentaires, comparées aux versions EM et DS. Cet inconvénient est dû à l'utilisation de compteurs d'initialisation à la place des mémoires internes des VNs

utilisées dans DS. La figure 4.5. présente la structure du nouveau nœud de variable à 3 degrés. La réduction supplémentaire de l'utilisation de la logique FPGA observée pour le nouveau décodeur proposé est principalement obtenue à la suite de s'en passer des compteurs à saturation de sortie VN.

La figure 4.6. présente le schéma de principe du décodeur stochastique LDPC proposé (1024, 512). Les unités principales sont l'unité des nœuds de variable, l'unité des nœuds de contrôle de parité et l'unité du contrôle du syndrome. L'unité des VNs et l'unité des CNs échangent les informations stochastiques jusqu'à atteindre un code correct ou le nombre maximum d'itérations. Le code correct est détecté par le contrôleur de syndrome, et le maximum d'itérations est détecté par le compteur d'itérations.

Les sorties du générateur des nombres aléatoires sont utilisées avec les comparateurs des VNs pour générer les séquences de Bernouli. De plus, ils sont directement utilisés pour piloter les bus d'adresses de la RAM distribuée FPGA, utilisée comme mémoire interne VN.

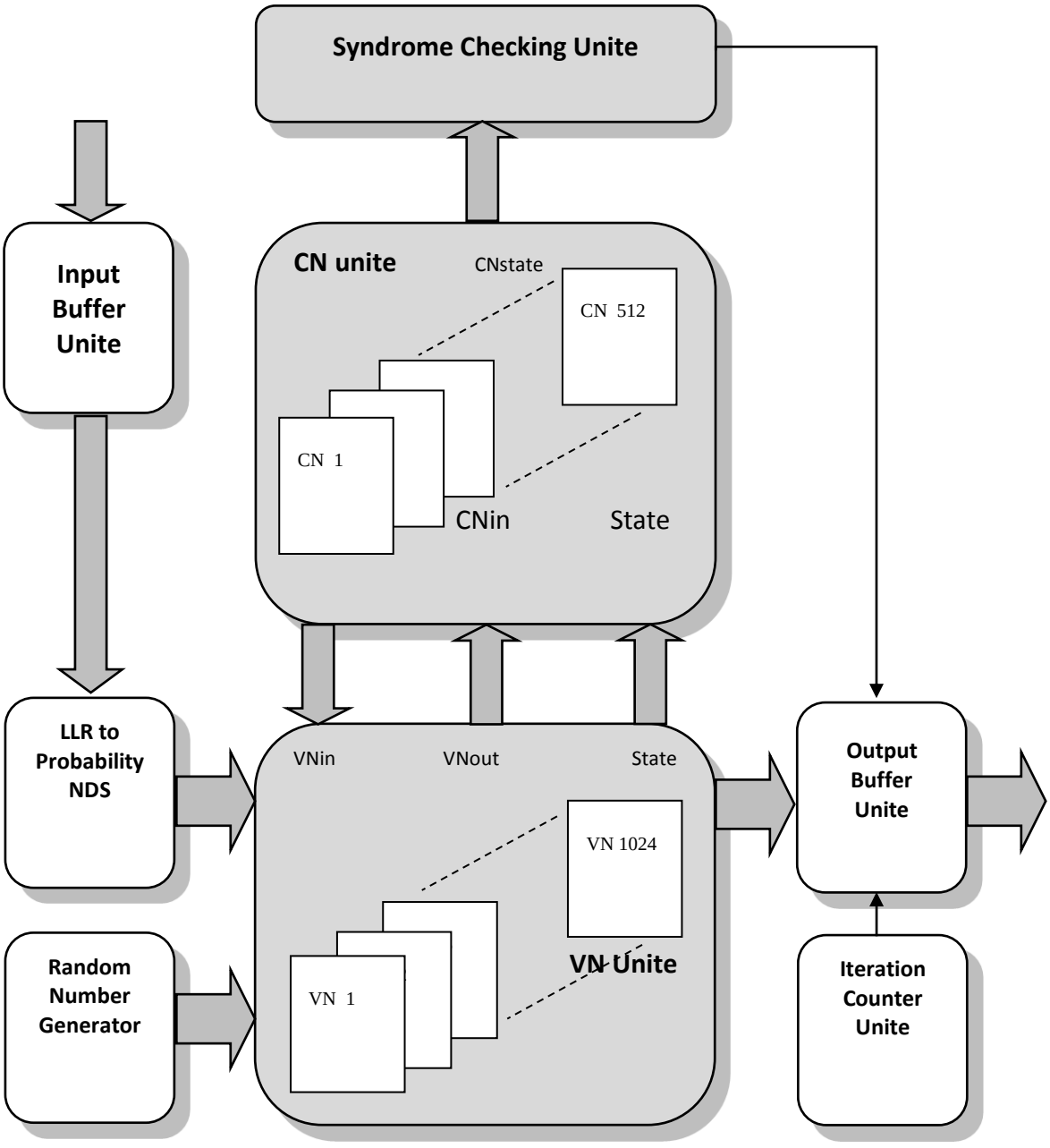


**Figure 4.6.** Synoptique du décodeur LDPC (1024, 512) proposé

## 4.3. Comparaison des taux d'utilisation logique sur FPGA

La figure 4.7. présente le rapport d'utilisation des ressources FPGA. Le décodeur proposé atteint une réduction moyenne d'environ 50% par rapport à la version du compteur à initialisation one-step [50] et une réduction moyenne d'environ 35% par rapport aux décodeurs DS et EM.

En plus de la réduction de l'utilisation de la logique, l'architecture proposée offre des performances supplémentaires même pour les codes courts.

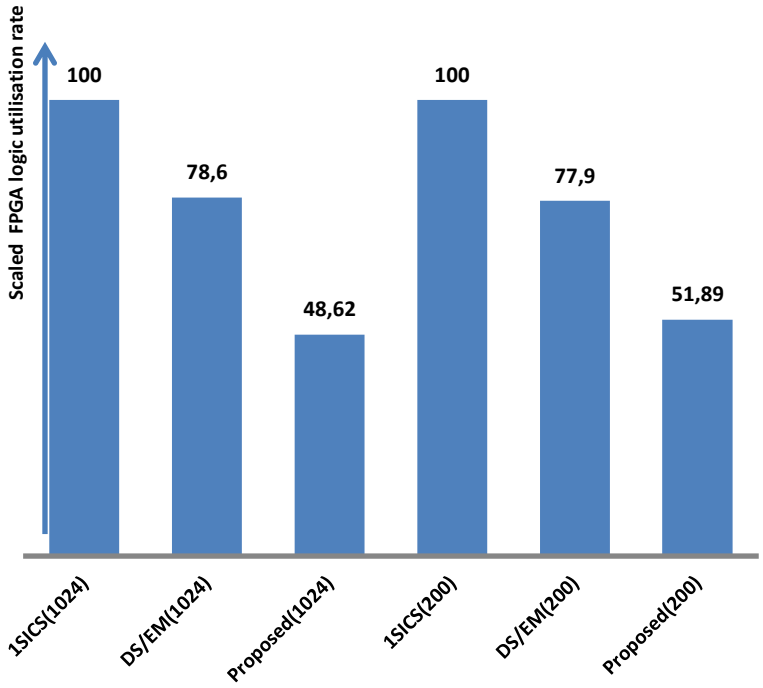


**Figure 4.7.** Rapport d'utilisation des ressources FPGA

Les résultats de la mise en œuvre FPGA des codes LDPC réguliers (200, 100) et (1024, 512) avec des VN basés sur l'approche DS, sur l'approche des compteurs à initialisation one-step et sur notre approche proposée, sont présentés dans le Tableau I.

**Tableau I.** Utilisation des ressources de l'implémentation des décodeurs LDPC stochastiques sur l'FPGA de Xilinx Virtex-6 VLX240T

| ***LDPC Architecture*** | ***Code*** | ***Implementation*** | ***Decoder Logic utilization*** *(LUT)* | ***VNs Logic utilization*** *(LUT)* | ***VN counter / Memory length*** *(bits)* | ***VNs rate utilization*** *(%)* | ***VNs Reduction rate compared to DS*** *(%)* | ***Reduction rate compared to DS*** *(%)* |
|---|---|---|---|---|---|---|---|---|
| *Initialized Counter-based VN [50]* | *(1024, 512)* | *Stochastic fully parallel* | 60 947 | 51 200 | 06 | 84,01 | - 35,13 | - 27,95 |
| | *(200, 100)* | | 11 767 | 10 000 | 06 | 84,98 | - 35,13 | - 28.36 |
| *DS [42]* | *(1024, 512)* | *Stochastic fully parallel* | 47 635 | 37 888 | 02 | 79,54 | 0 | 0 |
| | *(200, 100)* | | 9 167 | 7 400 | 02 | 80,72 | 0 | 0 |
| *Proposed* | *(1024, 512)* | *Stochastic fully parallel* | 29 633 | 21 504 | 16 | 72,57 | + 43,24 | + 37,79 |
| | *(200, 100)* | | 6 106 | 4 200 | 16 | 68,78 | + 43,24 | + 33,39 |

## 4.4. Performances et convergence du nouveau décodage sur FPGA

La figure 4.8. présente les performances BER du décodage LDPC (200 100) sur FPGA avec DS, CSS, EM et l'approche proposée. Le nombre d'itérations de décodage maximal est fixé à 1000 DC. Le BER de la nouvelle approche illustre une amélioration très significative.

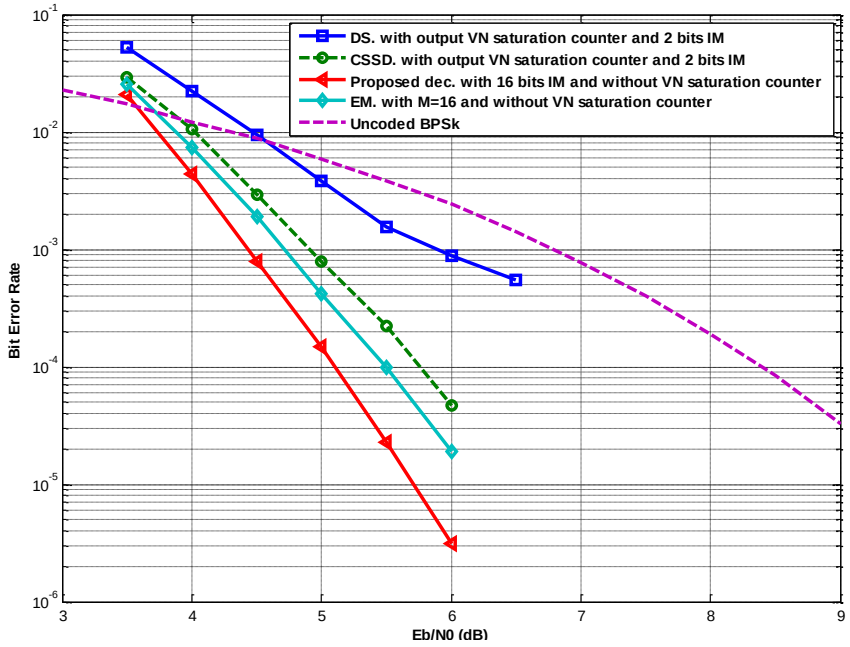


**Figure 4.8.** Comparaison des performances BER du code (200,100)

Le décodeur proposé avec une mémoire interne de 16 bits surpasse le décodeur EM de 0,5 dB pour un BER de $10^{-5}$. Les valeurs moyennes du DCs requis pour le DS et l'architecture proposée, avec un délai de décodage maximum de 1000 et 400, sont présentés sur la figure 4.9. Des réductions de 85,3% et 81,1%, dans l'ADC par rapport au décodeur DS à un SNR de 6 dB, sont observées pour 1000 et 400 Max-DC respectivement.

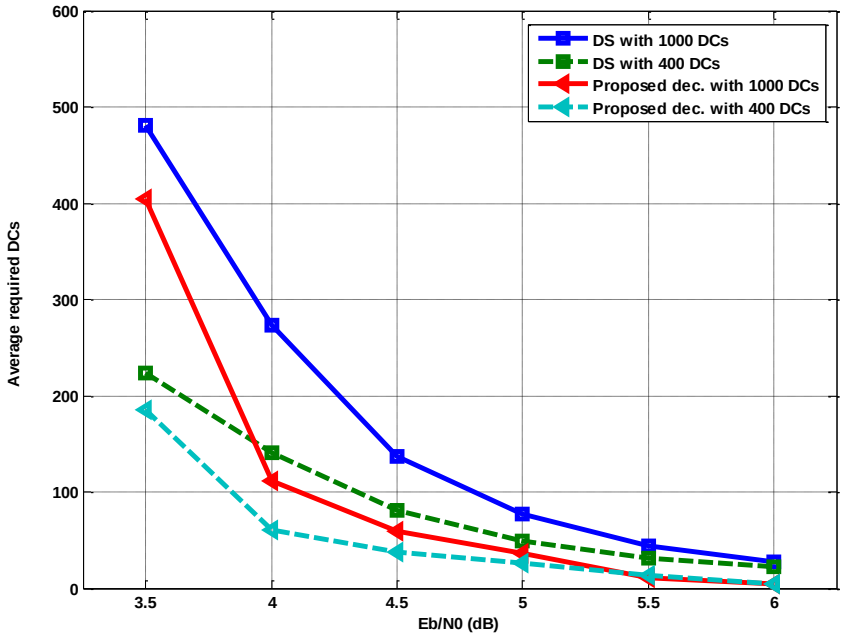


**Figure 4.9.** Cycles des décodages moyens

La figure 4.10. et la figure 4.11. montrent le résultat et la comparaison de l'implémentation des utilisations logiques FPGA de l'unité des nœuds de variable et du décodeur respectivement. Le décodeur proposé atteint une réduction moyenne d'environ 50% par rapport à la variante du compteur à initialisation one-step et une réduction moyenne d'environ 35% par rapport aux décodeurs DS et EM. En plus de la réduction de l'utilisation de la logique, la nouvelle architecture offre des performances supplémentaires même pour les codes courts. Le décodeur EM fournit un résultat d'implémentation FPGA proche de la variété DS.

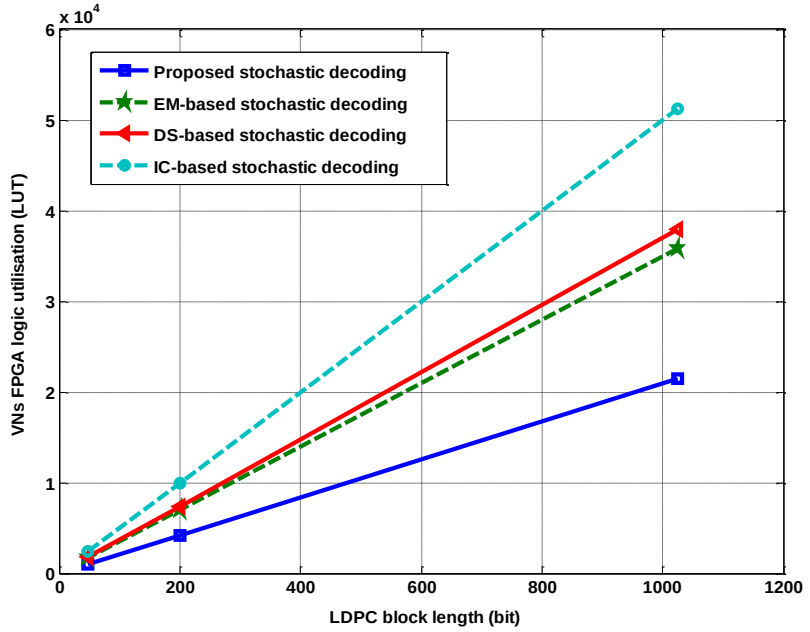


**Figure 4.10.** Résultat et comparaison de la logique d'utilisation FPGA pour l'implémentation des nœuds de variable

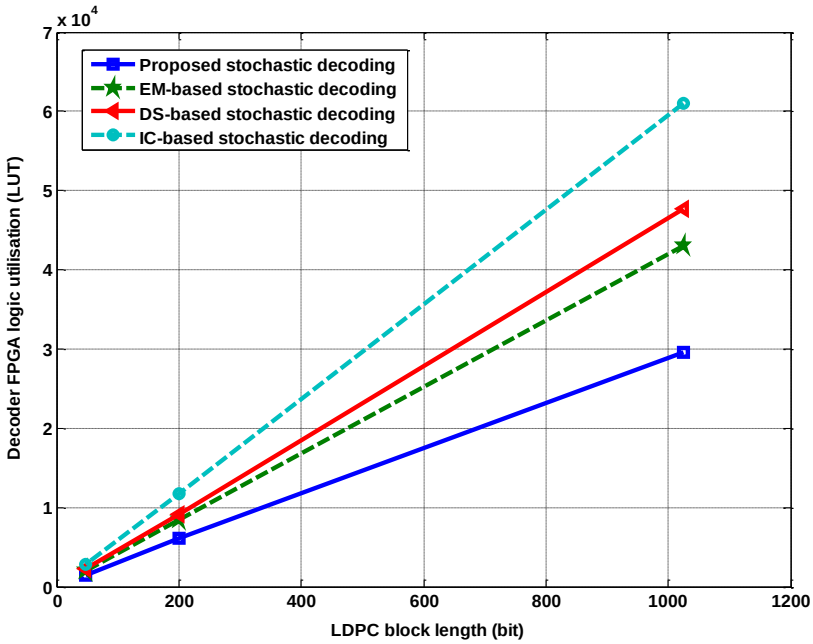


**Figure 4.11.** Résultat et comparaison de la logique d'utilisation FPGA pour l'implémentation des décodeurs stochastiques

## 4.5. Conclusion

Dans ce chapitre, nous avons étudié la complexité et les performances des décodeurs stochastiques LDPC implémentés sur FPGA. De plus, la nouvelle approche du décodage LDPC stochastique complètement parallèle a été présentée, qui peut surclasser tous les versions récentes. L'amélioration a été réalisée en introduisant un nœud de variable stochastique efficace.

Une réduction de la latence et de la complexité du décodage, en plus de l'amélioration du BER, est obtenue même pour les codes courts. Les résultats d'implémentation sur un FPGA de Xilinx, le Virtex-6 VLX 240T valident l'amélioration apportée par notre nouvelle méthode. Une réduction moyenne de 35% de l'utilisation logique et une réduction moyenne de 85% des cycles de décodage, par rapport à la méthode DS, sont réalisés. Les performances BER du nouveau décodeur (200, 100) dépassent les versions DS, CSS et EM. Un écart de gain de 0,5 dB, comparé à EM, est observé pour un BER de 10-5. Des descriptions VHDL sont données en annexes.

# CONCLUSION

Nous nous sommes intéressés dans ce travail à la conception de nouvelles architectures et algorithmes pour réduire la complexité matérielle tout en conservant les performances du décodage. Après une introduction du décodage LDPC, nous avons présenté les principes et les architectures des décodeurs LDPC à complexité réduite et leurs algorithmes associés. Le premier décodage LDPC non stochastique à complexité réduite est le décodage avec le "MIN-SUM algorithm" (MSA). La deuxième variante est le décodage avec le "Normalized MIN-SUM algorithm" (NMS). La troisième variante présentée est la variante du décodage avec l' "Offset MIN-SUM algorithm" (OMS).

Une imposante amélioration de l'algorithme de décodage OMS a été présentée. L'algorithme proposé est conçu pour réduire la complexité matérielle tout en conservant les performances du décodage. Le nouvel algorithme a été appliqué en utilisant deux différents codes LDPC réguliers (200, 100) et (1024, 512). Les résultats des simulations montrent que les performances du nouvel algorithme est proche du SPA à virgule flottante pour un faible rapport signal sur bruit (SNR) et peut surpasser le SPA à virgule flottante pour un SNR élevé, avec une complexité inférieure à la complexité du OMS classique.

Nous avons introduit les concepts de base du calcul stochastique, la représentation de l'information stochastique et les opérations arithmétiques. Une nouvelle étude du décodage LDPC stochastique sur le plan architectural et sur le plan logarithmique est présentée avec une investigation de complexité et de performances des décodeurs stochastiques LDPC implémentables sur FPGA.

La conception d'une performante approche de décodage LDPC stochastique, complètement parallèle et implémentable sur FPGA a été illustrée avec les détails de cette architecture, ainsi que les calculs et les étapes de l'algorithme associé.
Cette nouvelle approche du décodage LDPC stochastique, avec sa capacité du parallélisme complet sur FPGA, peut surclasser toutes les versions stochastiques récentes. L'amélioration a été réalisée en introduisant un nœud de variable stochastique très efficace. Une réduction de la latence et de la complexité du décodage, en plus de l'amélioration du BER, est obtenue même pour les codes courts.

Les résultats d'implémentation sur un FPGA de Xilinx, le Virtex-6 VLX 240T valident l'amélioration apportée par notre nouvelle méthode. Une réduction moyenne de 35% de l'utilisation logique sur FPGA, par rapport à la méthode DS, est réalisée pour les codes (200, 100) et (1024, 512). Également, une réduction de 85% pour les cycles de décodage, par rapport à la méthode DS, est enregistrée. Au même titre, les performances du BER du nouveau décodeur (200, 100) dépassent les versions DS, CSS et EM. Un écart de gain de 0,5 dB, comparé à la version EM, est observé pour un BER de $10^{-5}$.

Les perspectives à l'issue de cette thèse sont nombreuses, nous pouvons en citer à titre d'exemple le développement et l'implantation sur FPGA des codes LDPC non binaires.

# BIBLIOGRAPHIE


1. C. E. SHANNON, "A Mathematical Theory of Communication", The Bell System Technical Journal, Vol. 27, pp. 379–423, 623–656, July, October, 1948.
2. Claude E. Shannon and Warren Weaver, " The mathematical theory of communication", The University of Illinois Press, USA, 1964.
3. R. G. Gallager, “Low density parity check codes,” IRE Trans. Inf. Theory, vol. 8, no. , pp. 21–28, Jan. 1962.
4. Robert G. Gallager, “Low-Density Parity-Check Codes", Cambridge Mass, USA, July, 1963.
5. D. J. C. MacKay and R. M. Neal, “Near Shannon limit performance of low density parity check codes,” Electron. Lett., vol. 32, no. 18, pp. 1645–1646, 1996.
6. D. J. C. Mackay, "Good error correcting codes based on very sparse matrices", *IEEE Trans. on Inform. Theory*, vol. 45, pp. 399-431, Mar. 1999.
7. The Digital Video Broadcasting Standard [Online]. Available: www.dvb.org
8. The IEEE 802.3an 10GBASE-T Task Force [Online]. Available: www.ieee802.org/3/an
9. IEEE Working Group for 40 Gb/s and 100 Gb/s Operation Std. [Online]. Available: http://www.ieee802.org/3/
10. The IEEE 802.16 Working Group [Online]. Available: http://www. ieee802.org/16/
11. The IEEE 802.11n Working Group Std. [Online]. Available: http://www.ieee802.org/11/
12. G.975.1: Forward Error Correction for High Bit Rate DWDM Submarine Systems International Telecommunication Union Std., 2004 [Online]. Available: http://www.itu.int/rec/T-REC-G.975.1-200402-I/en
13. Rolf. Johannesson , Kamil .Sh. Zigangirov, FUNDAMENTALS OF CONVOLUTIONAL CODING , IEEE PRESS, 1999
14. WILLIAM. E. RYAN ,SHU. LIN, Channel Codes Classical and Modern, Cambridge University Press 2009.

15. Upamanyu Madhow, Fundamentals of Digital Communication, Cambridge University Press, 2008

16. F. Kschischang, B. Frey, and H. Loeliger, “Factor graphs and the sumproduct algorithm,” IEEE Trans. Inf. Theory, vol. 47, pp. 498–519, Feb. 2001.

17. Shu Lin. Daniel J. Costello, "Error Control Coding", second Edition", Prentice Hall, April. 2004.

18. Claude Berrou, "Codes et turbocodes". Springer-Verlag, Paris, 2007

19. Sarah J. Johnson, "Iterative Error Correction: Turbo, Low-Density Parity-Check and Repeat–Accumulate Codes", Cambridge University Press, USA, 2010.

20. I. Djurdjevic, J. Xu, K. Ghaffar, and S. Lin, "A class of low-density parity-check codes constructed based on Reed-Solomon codes with two information symbols," IEEE Commun. Lett., vol. 7, no. 7, pp. 317-319, July 2003.

21. M. P. C. Fossorier, M. Mihaljevic, H. Imai, "Reduced complexity iterative decoding of low density parity check codes based on belief propagation", IEEE Trans. Commun., vol. 47, no. 5, pp. 673-680, May 1999.

22. Jinghu Chen, A. Dholakia, E. Eleftheriou, M.P.C. Fossorier and Xiao-Yu Hu, "Reduced-Complexity Decoding of LDPC Codes", IEEE Trans. Commun., Vol. 53, no. 8, pp. 1288 - 1299, 2005

23. Ming Jiang, Chunming Zhao, Li Zhang, Enyang Xu, "Adaptive offset min-sum algorithm for low-density parity check codes", *IEEE Communication Letters*, vol. 10, no. 6, pp. 483-485, June 2006.

24. X. Wu, Y. Song, M. Jiang, C. Zhao, "Adaptive-normalized/offset min-sum algorithm", *IEEE Commun. Lett.*, vol. 14, no. 7, pp. 667-669, Jul. 2010.

25. Z. Zhang, V. Anantharam, M. J. Wainwright, and B. Nikolic, “A 47 Gb/s LDPC decoder with improved low error rate performance,” in Proc. Symp. VLSI Circuits, Jun. 2009, pp. 286–287.

26. Z. Zhang, V. Anantharam, M. J. Wainwright, and B. Nikolic, “An Efficient 10GBASE-T Ethernet LDPC Decoder Design With Low Error Floors” IEEE Trans. Solid- State Circuits, vol. 45, no. 4, pp. 843–855, Apr. 2010.

27. Oh D, Parhi K K, "Min-sum decoder architectures with reduced word length for LDPC codes", Circuits and Systems I: Regular Papers, IEEE Trans. on, 2010, V57(1), pp. 105-115.

28. Z. Zhang, L. Dolecek, B. Nikolic, V. Anantharam, M. Wainwright, "Design of LDPC decoders for improved low error rate performance: quantization and algorithm choices", *IEEE Trans. Commun.*, vol. 57, no. 11, pp. 3258-3276, Nov. 2009.

29. Tinoosh Mohsenin "Algorithms and Architectures for Efficient Low Density Parity Check (LDPC) Decoder Hardware," Ph.D Thesis, VLSI Computation Laboratory, ECE Department, University of California, Davis, USA, 2010.

30. C.-L. Wey, M.-D. Shieh, S.-Y. Lin, "Algorithms of finding the first two minimum values and their hardware implementation", *IEEE Trans. Circuits Syst. I Reg. Papers*, vol. 55, no. 11, pp. 3430-3437, Dec. 2008.

31. J. Jung, Y. Lee, I.-C. Park, "Area-efficient method to approximate two minima for LDPC decoders", *Electron. Lett.*, vol. 50, no. 23, pp. 1701-1702, Nov. 2014.

32. Y. Lee, B. Kim, J. Jung, I. C. Park, "Low-complexity tree architecture for finding the first two minima", *IEEE Trans. Circuits Syst. II Exp. Briefs*, vol. 62, no. 1, pp. 61-64, Jan. 2015.

33. I. Tsatsaragkos, V. Paliouras, "Approximate algorithms for identifying minima on min-sum LDPC decoders and their hardware implementation", *IEEE Trans. Circuits Syst. II Exp. Briefs*, vol. 62, no. 8, pp. 766-770, Aug. 2015.

34. R. G. Gallager, Stochastic Processes Theory for Applications, Cambridge University Press, 2013

35. V. Gaudet and A. Rapley, “Iterative decoding using stochastic computation,” Electron. Lett., vol. 39, no. 3, pp. 299–301, Feb. 2003.

36. S. S. Tehrani,W. Gross, and S.Mannor, “Stochastic decoding of LDPC codes,” IEEE Commun. Lett., vol. 10, no. 10, pp. 716–718, Oct. 2006.

37. S. S. Tehrani, S. Mannor, and W. Gross, “Fully parallel stochastic LDPC decoders,” IEEE Trans. Signal Process., vol. 56, no. 11, pp. 5692–5703, Nov. 2008.

38. S. Tehrani, A. Naderi, G.-A. Kamendje, S. Mannor, and W. Gross, “Tracking forecast memories in stochastic decoders,” in IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP), Apr. 2009, pp. 561–564.

39. S. S. Tehrani, A. Naderi, G. A. Kamendje, S. Mannor, and W. Gross, “Tracking forecast memories for stochastic decoding,” J. Signal Process. Syst. pp. 1–11, 2010 [Online]. Available: http://dx.doi.org/10.1007/s11265-009-0441-5

40. S. S. Tehrani, A. Naderi, G.-A. Kamendje, S. Hemati, S. Mannor, and W. Gross, “Majority-based tracking forecast memories for stochastic LDPC decoding,” IEEE Trans. Signal Process., vol. 58, no. 9, pp. 4883–4896, Sep. 2010.

41. Saeed Sharifi Tehrani, "Stochastic Decoding of Low-Density Parity-Check Codes", Doctor of Philosophy, McGill University Montreal, Department of Electrical and Computer Engineering, Quebec, Canada, January 2011.

42. Ali Naderi, Shie Mannor, Mohamad Sawan and Warren J. Gross, “Delayed Stochastic Decoding of LDPC Codes” IEEE Trans. Signal Process., vol. 59, no. 11, pp. 5617–5626, Nov. 2011.

43. C. Winstead, V. Gaudet, A. Rapley, and C. Schlegel, “Stochastic iterative decoders,” in Proc. Int. Symp. Inf. Theory (ISIT), Sep. 2005, pp. 1116–1120.

44. Mountassar Maamoun, Rafik Bradai, Ali Naderi, Rachid Beguenane, Mohamad Sawan, "Controlled Start-Up Stochastic Decoding of LDPC Codes", *IEEE Int. NEWCAS Conference*, June. 2013.

45. Kuo-Lun Huang, Vincent Gaudet and Masoud Salehi, "Output Decisions for Stochastic LDPC Decoders" in 48th Annual Conference on Information Sciences and Systems, (CISS), *Princeton, USA,* March. 2014.

46. Kuo-Lun Huang, "Efficient Algorithms for Stochastic Decoding of LDPC Codes" Doctor of Philosophy, Northeastern University, Boston, USA, May. 2016.

47. Ghania Zerari and Abderrezak Guessoum, “Reduced-Latency and Area-Efficient Architecture for FPGA-Based Stochastic LDPC Decoders” International Journal of Advanced Computer Science and Applications (ijacsa), 8(7), 2017. http://dx.doi.org/10.14569/IJACSA.2017.080707

48. Ghania Zerari, Abderrezak Guessoum, and Rachid Beguenane, "A New Design for Reducing Logic Utilizations in FPGA-Based Stochastic LDPC Decoders," International Journal of Electrical and Electronic Engineering & Telecommunications, Vol. 6, No. 4, pp. 1-5, October 2017. DOI: 10.18178/ijeetc.6.4.1-5

49. Ghania Zerari and Abderrezak Guessoum, "Improved Controlled Start-up Stochastic LDPC Decoder for Efficient FPGA-based Implementation", in 2015 4th International Conference on Electrical Engineering (ICEE), Boumerdes, Algeria, Dec.2015

50. Di Wu, Yun Chen, Qichen Zhang, Yeong-luh Ueng and Xiaoyang Zeng "Strategies for Reducing Decoding Cycles in Stochastic LDPC Decoders" IEEE Trans. Circuits and Systems II, vol. 63, no.91, pp. 873 - 877, Sept. 2016.

# ANNEXES

# A1. VHDL du nouveau sous-nœud de variable avec $d_v$=3

```
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL;
library UNISIM;
use UNISIM.Vcomponents.ALL;

entity DS_VN1_3_005_MUSER_DS_VN_0052 is
  port ( A       : in    std_logic_vector (3 downto 0);
         CLK     : in    std_logic;
         CLKE    : in    std_logic;
         MajorStt : in    std_logic;
         Vin1    : in    std_logic;
         Vin2    : in    std_logic;
         Vin3    : in    std_logic;
         OBit    : out   std_logic;
         S       : out   std_logic;
         State   : out   std_logic);
end DS_VN1_3_005_MUSER_DS_VN_0052;


architecture BEHAVIORAL of DS_VN1_3_005_MUSER_DS_VN_0052 is
  attribute BOX_TYPE   : string ;
  attribute HU_SET     : string ;
  signal XLXN_1   : std_logic;
  signal XLXN_2   : std_logic;
  signal XLXN_6   : std_logic;
  signal XLXN_12  : std_logic;
  signal XLXN_13  : std_logic;
  signal XLXN_14  : std_logic;
  signal XLXN_47  : std_logic;
  signal XLXN_49  : std_logic;
  signal XLXN_54  : std_logic;
  signal S_DUMMY  : std_logic;

  component AND2
    port ( I0 : in    std_logic;
           I1 : in    std_logic;
           O  : out   std_logic);
  end component;

  attribute BOX_TYPE of AND2 : component is "BLACK_BOX";

 component NOR2
    port ( I0 : in    std_logic;
           I1 : in    std_logic;
```

```
      O  : out   std_logic);
end component;

attribute BOX_TYPE of NOR2 : component is "BLACK_BOX";

component M2_1_MXILINX_DS_VN_0052
  port ( D0 : in    std_logic;
      D1 : in    std_logic;
      S0 : in    std_logic;
      O  : out   std_logic);
end component;

component AND2B1
  port ( I0 : in    std_logic;
      I1 : in    std_logic;
      O  : out   std_logic);
end component;

attribute BOX_TYPE of AND2B1 : component is "BLACK_BOX";

component FDE
  generic( INIT : bit :=  '0');
  port ( C  : in    std_logic;
      CE : in    std_logic;
      D  : in    std_logic;
      Q  : out   std_logic);
end component;

attribute BOX_TYPE of FDE : component is "BLACK_BOX";

component OR2
  port ( I0 : in    std_logic;
      I1 : in    std_logic;
      O  : out   std_logic);
end component;

attribute BOX_TYPE of OR2 : component is "BLACK_BOX";

component IM_16bit_Sch_MUSER_DS_VN_0052
  port ( WE   : in    std_logic;
      M_in  : in    std_logic;
      CLK   : in    std_logic;
      A     : in    std_logic_vector (3 downto 0);
      M_out : out   std_logic);
end component;
attribute HU_SET of XLXI_14 : label is "XLXI_14_5";
attribute HU_SET of XLXI_20 : label is "XLXI_20_3";
```

```
  attribute HU_SET of XLXI_33 : label is "XLXI_33_4";

begin

  S <= S_DUMMY;
  XLXI_2 : AND2
    port map (I0=>Vin2,
              I1=>Vin1,
              O=>XLXN_1);

  XLXI_3 : NOR2
    port map (I0=>Vin2,
              I1=>Vin1,
              O=>XLXN_2);

  XLXI_14 : M2_1_MXILINX_DS_VN_0052
    port map (D0=>XLXN_54,
              D1=>Vin2,
              S0=>XLXN_6,
              O=>XLXN_14);

  XLXI_17 : AND2
    port map (I0=>Vin3,
              I1=>XLXN_14,
              O=>XLXN_12);

  XLXI_18 : NOR2
    port map (I0=>Vin3,
              I1=>XLXN_14,
              O=>XLXN_13);

  XLXI_19 : NOR2
    port map (I0=>XLXN_13,
              I1=>XLXN_12,
              O=>S_DUMMY);

  XLXI_20 : M2_1_MXILINX_DS_VN_0052
    port map (D0=>Vin3,
              D1=>A(0),
              S0=>S_DUMMY,
              O=>XLXN_47);

  XLXI_32 : AND2B1
    port map (I0=>MajorStt,
              I1=>S_DUMMY,
              O=>XLXN_49);

  XLXI_33 : M2_1_MXILINX_DS_VN_0052
```

```
    port map (D0=>XLXN_47,
              D1=>XLXN_54,
              S0=>MajorStt,
              O=>OBit);

  XLXI_37 : FDE
    port map (C=>CLK,
              CE=>CLKE,
              D=>XLXN_49,
              Q=>State);

  XLXI_38 : OR2
    port map (I0=>XLXN_2,
              I1=>XLXN_1,
              O=>XLXN_6);

  XLXI_39 : IM_16bit_Sch_MUSER_DS_VN_0052
    port map (A(3 downto 0)=>A(3 downto 0),
              CLK=>CLK,
              M_in=>Vin2,
              WE=>XLXN_6,
              M_out=>XLXN_54);

end BEHAVIORAL;
```

# A2. VHDL de la mémoire interne à 16 bits de la VN proposée

```
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL;
library UNISIM;
use UNISIM.Vcomponents.ALL;

entity IM_16bit_Sch_MUSER_DS_VN_0052 is
  port ( A    : in    std_logic_vector (3 downto 0);
         CLK  : in    std_logic;
         M_in : in    std_logic;
         WE   : in    std_logic;
         M_out : out  std_logic);
end IM_16bit_Sch_MUSER_DS_VN_0052;

architecture BEHAVIORAL of IM_16bit_Sch_MUSER_DS_VN_0052 is
  attribute INIT      : string ;
  attribute BOX_TYPE  : string ;
  component RAM16X1S
    -- synopsys translate_off
    generic( INIT : bit_vector :=  x"0000");
    -- synopsys translate_on
    port ( A0   : in    std_logic;
           A1   : in    std_logic;
           A2   : in    std_logic;
           A3   : in    std_logic;
           D    : in    std_logic;
           WCLK : in    std_logic;
           WE   : in    std_logic;
           O    : out   std_logic);
  end component;
  attribute INIT of RAM16X1S : component is "0000";
  attribute BOX_TYPE of RAM16X1S : component is "BLACK_BOX";

begin
  XLXI_3 : RAM16X1S
    port map (A0=>A(0),
              A1=>A(1),
              A2=>A(2),
              A3=>A(3),
              D=>M_in,
              WCLK=>CLK,
              WE=>WE,
              O=>M_out);
 end BEHAVIORAL;
```

# A3. VHDL du compteur-décompteur à saturation

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Counter_St6b is
   Port ( C : in  STD_LOGIC;
          CE : in  STD_LOGIC;
          UD : in  STD_LOGIC;
          RST : in  STD_LOGIC;
          BN_1 : out  STD_LOGIC;
          TCount: out STD_LOGIC);
end Counter_St6b;

architecture Behavioral of Counter_St6b is
          signal count: STD_LOGIC_VECTOR (5 downto 0):="000000";
          signal TC: std_logic:='0';
begin
process (C)
begin
  if C='1' and C'event then
    if (RST='0') then
          if (CE='1') then
                              if ((signed(count) /= -31) and (signed(count) /= 31)) then
                                        TC <= '0';
                                        if (UD='1') then
        count <= count + 1;
                             else
       count <=  count - 1;
                                        end if;
                                        else
                                                  count <= count;
                                        TC <= '1';
                                        end if;
                              end if;
                              else
      count <= "000000";
              TC <= '0';
                    end if;
  end if;
end process;
          BN_1 <= NOT(count(5));
          TCount <= TC;
end Behavioral;
```

# A4. VHDL du nœud de contrôle adopté avec $d_c=6$

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity DS_CN6 is
  port ( CNin1  : in    std_logic;
         CNin2  : in    std_logic;
         CNin3  : in    std_logic;
         CNin4  : in    std_logic;
         CNin5  : in    std_logic;
         CNin6  : in    std_logic;
         State1 : in    std_logic;
         State2 : in    std_logic;
         State3 : in    std_logic;
         State4 : in    std_logic;
         State5 : in    std_logic;
         State6 : in    std_logic;
         CNout  : out   std_logic;
         CNout1 : out   std_logic;
         CNout2 : out   std_logic;
         CNout3 : out   std_logic;
         CNout4 : out   std_logic;
         CNout5 : out   std_logic;
         CNout6 : out   std_logic);
end DS_CN6;

architecture Behavioral of DS_CN6 is
            signal CNall : STD_LOGIC;
            signal CS : STD_LOGIC;
            signal CNint : STD_LOGIC;
begin
            CNall <= CNin1 xor CNin2 xor CNin3 xor CNin4 xor CNin5 xor CNin6 ;
            CS <= '1' when not(State1 or State2 or State3 or State4 or State5 or State6) = '1'
else '0';
CNint <= (CNall and CS);
CNout <= CNall;
CNout1 <= (CNint xor CNin1);
CNout2 <= (CNint xor CNin2);
CNout3 <= (CNint xor CNin3);
CNout4 <= (CNint xor CNin4);
CNout5 <= (CNint xor CNin5);
CNout6 <= (CNint xor CNin6);

end Behavioral;
```

# A5. VHDL du générateur aléatoire à distribution uniforme.

```
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL;
library UNISIM;
use UNISIM.Vcomponents.ALL;

entity RE_16bitXOR is
  port ( CE      : in    std_logic;
         CLK     : in    std_logic;
         RST     : in    std_logic;
         EMout_16 : out   std_logic_vector (15 downto 0));
end RE_16bitXOR;

architecture BEHAVIORAL of RE_16bitXOR is
  attribute BOX_TYPE   : string ;
  signal XLXN_17       : std_logic;
  signal XLXN_18       : std_logic;
  signal XLXN_19       : std_logic;
  signal EMout_16_DUMMY : std_logic_vector (15 downto 0);

 component XOR2
    port ( I0 : in    std_logic;
           I1 : in    std_logic;
           O  : out   std_logic);
  end component;
  attribute BOX_TYPE of XOR2 : component is "BLACK_BOX";

  component FDE
    generic( INIT : bit :=  '0');
    port ( C  : in    std_logic;
           CE : in    std_logic;
           D  : in    std_logic;
           Q  : out   std_logic);
  end component;
  attribute BOX_TYPE of FDE : component is "BLACK_BOX";

  component FDPE
    generic( INIT : bit :=  '1');
    port ( C   : in    std_logic;
           CE  : in    std_logic;
           D   : in    std_logic;
           PRE : in    std_logic;
           Q   : out   std_logic);
```

```
  end component;
  attribute BOX_TYPE of FDPE : component is "BLACK_BOX";

begin
  EMout_16(15 downto 0) <= EMout_16_DUMMY(15 downto 0);
  XLXI_19 : XOR2
    port map (I0=>EMout_16_DUMMY(14),
          I1=>EMout_16_DUMMY(15),
          O=>XLXN_17);

  XLXI_20 : XOR2
    port map (I0=>EMout_16_DUMMY(12),
          I1=>XLXN_17,
          O=>XLXN_18);

  XLXI_21 : XOR2
    port map (I0=>EMout_16_DUMMY(3),
          I1=>XLXN_18,
          O=>XLXN_19);

  XLXI_26 : FDE
    port map (C=>CLK,
          CE=>CE,
          D=>EMout_16_DUMMY(9),
          Q=>EMout_16_DUMMY(10));

  XLXI_27 : FDE
    port map (C=>CLK,
          CE=>CE,
          D=>EMout_16_DUMMY(10),
          Q=>EMout_16_DUMMY(11));

  XLXI_28 : FDE
    port map (C=>CLK,
          CE=>CE,
          D=>EMout_16_DUMMY(11),
          Q=>EMout_16_DUMMY(12));

  XLXI_29 : FDE
    port map (C=>CLK,
          CE=>CE,
          D=>EMout_16_DUMMY(12),
          Q=>EMout_16_DUMMY(13));

  XLXI_30 : FDE
    port map (C=>CLK,
          CE=>CE,
          D=>EMout_16_DUMMY(13),
```

```
        Q=>EMout_16_DUMMY(14));

XLXI_31 : FDE
  port map (C=>CLK,
        CE=>CE,
        D=>EMout_16_DUMMY(14),
        Q=>EMout_16_DUMMY(15));

XLXI_32 : FDE
  port map (C=>CLK,
        CE=>CE,
        D=>EMout_16_DUMMY(8),
        Q=>EMout_16_DUMMY(9));

XLXI_33 : FDE
  port map (C=>CLK,
        CE=>CE,
        D=>EMout_16_DUMMY(7),
        Q=>EMout_16_DUMMY(8));

XLXI_34 : FDE
  port map (C=>CLK,
        CE=>CE,
        D=>EMout_16_DUMMY(6),
        Q=>EMout_16_DUMMY(7));

XLXI_35 : FDE
  port map (C=>CLK,
        CE=>CE,
        D=>EMout_16_DUMMY(5),
        Q=>EMout_16_DUMMY(6));

XLXI_36 : FDE
  port map (C=>CLK,
        CE=>CE,
        D=>EMout_16_DUMMY(4),
        Q=>EMout_16_DUMMY(5));

XLXI_37 : FDE
  port map (C=>CLK,
        CE=>CE,
        D=>EMout_16_DUMMY(3),
        Q=>EMout_16_DUMMY(4));

XLXI_38 : FDE
  port map (C=>CLK,
        CE=>CE,
        D=>EMout_16_DUMMY(2),
```

```
            Q=>EMout_16_DUMMY(3));

    XLXI_39 : FDE
      port map (C=>CLK,
            CE=>CE,
            D=>EMout_16_DUMMY(1),
            Q=>EMout_16_DUMMY(2));

    XLXI_40 : FDE
      port map (C=>CLK,
            CE=>CE,
            D=>EMout_16_DUMMY(0),
            Q=>EMout_16_DUMMY(1));

    XLXI_41 : FDPE
      port map (C=>CLK,
            CE=>CE,
            D=>XLXN_19,
            PRE=>RST,
            Q=>EMout_16_DUMMY(0));

end BEHAVIORAL;
```

# A6. VHDL du compteur d'itération.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Counter_Itir1 is
   Port ( CLK : in  STD_LOGIC;
          CE : in  STD_LOGIC;
          Cout: out  STD_LOGIC;
          RST : in  STD_LOGIC);

end Counter_Itir1;

architecture Behavioral of Counter_Itir1 is
          signal count: STD_LOGIC_VECTOR (13 downto 0):="00000000000000";
          signal TC: STD_LOGIC:='0';

begin

process (CLK)
begin
  if CLK='1' and CLK'event then
    if (RST='0') then
                              if (CE='1') then
      if (count /= 4000) then -- valeur max d'itirations
        count <= count + 1;
                                        TC <= '0';
                              else
                                        count <= count;
                                        TC <= '1';
                                        end if;
                              end if;
                              else
      count <= "00000000000000";
                              TC <= '0';
                              end if;
  end if;
end process;

Cout <= TC;

end Behavioral;
```