Le principe de calcul stochastique appliqué au décodage des turbocodes : conception, implémentation et prototypage sur circuit FPGA

Quang Trung Dong

N° d'ordre : 2011telb0208

Sous le sceau de l'Université européenne de Bretagne

Télécom Bretagne

En habilitation conjointe avec l'Université de Bretagne-Sud

Ecole Doctorale – SICMA

Le principe du calcul stochastique appliqué au décodage des turbocodes : conception, implémentation et prototypage sur circuit FPGA

Thèse de Doctorat

Mention : Sciences et Technologies de l'Information et de la Communication

Présentée par **Quang Trung DONG**

Département: Electronique

Laboratoire : Lab-STICC Pôle: Brest

Directeur de thèse : Christophe JEGO

Soutenue le 20 Décembre 2011

Jury :

M. Jean-Philippe DIGUET	Directeur de Recherche CNRS, Lab-STICC/UBS Lorient (Président)
M. Emmanuel CASSEAU	Professeur des Universités, ENSSSAT Lannion (Rapporteur)
M. Dominique DALLET	Professeur des Universités, IPB Bordeaux (Rapporteur)
M. Christophe JEGO	Professeur des Universités, IPB Bordeaux (Directeur de thèse)
M. Matthieu ARZEL	Maître de conférences, Telecom Bretagne Brest (Encadrant)

Le principe du calcul stochastique appliqué au décodage des turbocodes : conception, implémentation et prototypage sur circuit FPGA

Quang Trung DONG

Laboratoire en Sciences et Techniques de l'Information, de la Communication

et de la Connaissance (Lab-STICC)

Télécom Bretagne

Brest, Décembre 2011

Remerciements

Avant tout, je tiens à exprimer ma profonde reconnaissance et mon amitié à Mr. Christophe Jégo, professeur des universités à l'IPB Enseirb - Matmeca Bordeaux et mon directeur de thèse, qui m'a permis de développer mon goût pour la recherche et qui m'a guidé durant ces trois années. Par sa disponibilité de tous les instants, son investissement personnel, ses critiques constructives ainsi que par son enthousiasme et sa persévérance, il m'a permis de faire ressortir le meilleur de moi-même tout au long de ces années.

Je remercie Mr. Michel Jézéquel, directeur du laboratoire d'Electronique de Telecom Bretagne et mon ancien directeur de thèse pour m'y avoir acceuilli et pour la qualité de son encadrement. Sa disponibilité, ses remarques judicieuses sont autant de raisons pour lesquelles son encadrement fût extrêmement profitable.

Je tiens à exprimer tout ma gratitude à Mr. Matthieu Arzel, Maître de conférences à Télécom Bretagne. Je tiens à le remercier pour m'avoir encadré, pour son suivi, ses conseils utiles, son aide technique et sa gentillesse. Mes remerciements particulièrement pour son soutien sans faille au cours de ces trois années de thèse et pour la qualité de son expertise qui a éclairé nombre de nos discussions.

Je tiens à remercier Mr Dominique Dallet, professeur des universités à l'IPB Enseirb - Matmeca Bordeaux, et Mr. Emmanuel Casseau, professeur des universités ENSSAT Lannion d'avoir acceptés d'être rapporteurs de ces travaux de thèse. Leurs commentaires et leurs questions ont permis de clarifier le manuscrit. Je remercie également Mr. Jean-Philippe Diguet, directeur de recherche, université de Bretagne-Sud et qui m'a fait l'honneur de présider le jury de cette thèse.

Je tiens aussi à mentionner le plaisir que j'ai eu à travailler au sein du laboratoire Lab-STICC. Je remercie tous les membres (passé et présent) du Département Eletronique de Telecom Bretagne et du laboratoire Lab-STICC de l'université de Bretagne-Sud, tout spécialement mon meilleur ami Camilo Arturo Londoño Fuentes. Par ailleurs, mes remerciements vont également au personnel administratif et les services informatiques pour les aides que j'ai reçues.

Je remercie chaleureusement mes amies Ngoc Dieu NGO et Thi Thu Huyen NGUYEN d'avoir partagé avec moi les moments de joie, de tristesse, et de bonheur durant ces trois années de thèse.

Pour leurs encouragements et leur assistance aussi bien matérielle que morale qui m'ont permis de mener ce travail de thèse dans de bonnes conditions, je remercie chaudement tous les membres de ma famille. Ils ont été présents pour écarter les doutes, soigner les blessures et partager les joies. Cette thèse est donc un peu la leur.

Auteur : Quang Trung DONG

Titre : Le principe du calcul stochastique appliqué au décodage des turbocodes : conception, implémentation et prototypage sur circuit FPGA

Directeurs de thèse : Christophe JEGO et Matthieu ARZEL

RÉSUMÉ :

Depuis leur publication en 1993 et leur exploitation dans le domaine des communications numériques, les turbocodes ont été adoptés dans de nombreux standards de télécommunications (UMTS, CDMA2000, LTE). Avec le développement des services sans-fil, le besoin de turbo-décodeurs à débit jusqu'au Gbits/s devient incontournable. Or, les techniques conventionnelles d'exploitation du parallélisme et de réduction des chemins critiques atteignent leur limite. Une approche alternative a été explorée dans cette thèse : le décodage de codes correcteurs d'erreurs à partir d'une représentation stochastique de l'information.

Le calcul stochastique fut proposé dans les années 1960 comme une méthode traitant des opérations arithmétiques complexes pour un faible coût matériel. Pour ce faire, les probabilités sont converties en des flux de bits aléatoires dans lesquels l'information est représentée par des statistiques de bits. Des opérations arithmétiques complexes sur les probabilités sont transformées en des opérations sur les bits utilisant des portes logiques. Ainsi, l'application du calcul stochastique au décodage itératif de codes correcteurs d'erreurs favorise des structures matérielles simples pour les noeuds de calcul.

L'objectif principal de cette thèse fut d'étendre l'approche stochastique au décodage de turbocodes. Dans un premier temps, nous avons proposé une première architecture pour un turbo-décodeur stochastique.

Le principal défi fut ensuite d'augmenter le débit. Pour ce faire, nous avons considéré deux techniques : le passage dans le domaine exponentiel et l'exploitation du parallélisme. La première technique consiste à remplacer l'addition stochastique par des opérations plus simples dans le domaine exponentiel. Cette technique a permis d'une part de réduire la complexité calculatoire, et d'autre part, d'améliorer le débit de décodage. La deuxième technique est de représenter une probabilité par plusieurs flux stochastiques en parallèle. Cette méthode permet en outre de proposer une nouvelle approche pour compenser le problème de corrélation. L'exploitation de ces deux techniques a permis d'aboutir à un décodage stochastique pour les codes convolutifs et les turbocodes ayant des performances similaires à celles de décodeurs classiques.

Enfin, les architectures proposées ont été intégrées dans un circuit configurable FPGA. Le prototype de turbo-décodeur stochastique n'a pas pu nous fournir un débit de l'ordre du Gbits/s comme souhaité. Cependant, il a permis de démontrer la faisabilité matérielle d'un turbo-décodeur stochastique en assurant de bonnes performances de décodage. En outre, ce démonstrateur a fait apparaître de nombreuses perspectives pour cette solution d'intégration alternative.

Mots clés : turbocodes, algorithme MAP, décodage stochastique, prototypage FPGA.

Author : Quang Trung DONG

Title : Principles of stochastic computation applied to turbo decoding : design, implementation and prototyping on FPGA circuits

Supervisors : Christophe JEGO and Matthieu ARZEL

ABSTRACT :

The publication of turbo codes in 1993 proved that there were error-correcting codes with performance close to the theoretical limit and which can be used in industrial products. Quickly, turbo codes have been adopted in several standards such as DVB-RCS, UMTS, CDMA2000 and 3GPP-LTE. The increasing demand for high throughput applications in broadband applications until 1 Gb/s and beyond is strongly calling for high-speed turbo decoder implementations, thus leading to new challenges. An alternative approach was explored in this thesis : the decoding of error-correcting codes based on a stochastic representation of information.

Principles of stochastic computation were first presented in the 1960's as a method to carry out complex operations with a low hardware complexity. The main feature of this method is that the probabilities are converted into streams of random bits using Bernoulli sequences, in which the information is given by the statistics of the bits. As a result, complex arithmetic operations on probabilities are transformed into operations on bits using elementary logic gates. The application of stochastic calculation in iterative decoding of error-correcting codes leads to very simple physical structures of computation nodes.

The objective of this thesis is to apply the stochastic decoding approach to the turbo codes. Our first contribution shows that a stochastic turbo decoder architecture can be obtained with no significant loss of performance. However, a major challenge in the implementation of stochastic turbo decoders is to improve the decoding throughput.

In order to solve this challenge, we have proposed two efficient solutions : the transformation of stochastic additions into the exponential domain and the exploration of new parallelism schemes. The first technique consists in replacing the stochastic additions by simple operations after exponential transformations. This technique allows to reduce the computational complexity, and to improve the decoding throughput. The second technique is to represent a probability by several parallel stochastic streams. This method also allows us to compensate the correlation problem. These two techniques have resulted in stochastic convolutional decoders and turbo decoders with performances similar to the ones of conventional decoders.

Finally, the proposed architectures for stochastic convolutional decoders and stochastic turbo decoders were implemented on programmable devices (FPGAs). The stochastic turbo decoder prototype demonstrates the feasibility of a stochastic turbo decoder in terms of performance and complexity. In addition, it enables many prospects for this alternative integration solution.

Keywords : turbo codes, MAP algorithm, stochastic decoding, FPGA prototyping.

Glossaire

3GPP	3rd Generation Partnership Project
ABBG	Additif à Bruit Blanc Gaussien
ARP	Almost Regular Permutation
ASIC	Application Specific Integrated Circuit
AWGN	Additive White Gaussian Noise
BCJR	Bahl, Cocke, Jelinek and Raviv
BPSK	Binary Phase Shift Keying
BRAM	Bloc Random Access Memory
CCRS	Codes Convolutifs Récursifs Systématiques
CDMA	Code Division Multiple Acess
DVB-RCT	Digital Video Broadcasting - Return Channel Terrestrial
DVB-RCS	Digital Video Broadcasting - Return Channel via Satellite
DRP	Dithered Relatively Prime
EM	Edge Memory
FPGA	Field Programmable Gate Array
HSDPA	High Speed Downlink Packet Access
LDPC	Low Density Parity Check
LFSR	Linear Feedback Shift Registers
LRV	Logarithme du Rapport de Vraisemblance
LTE	Long Term Evolution
LUT	Look Up Table
MAP	Maximum A Posteriori
MTFM	Majority-based Tracking Forecast Memories
NDS	Noise-Dependant Scaling
RHS	Relaxed Half Stochastic
SNR	Signal to Noise Ratio
SISO	Soft Input Soft Output
TEB	Taux d’ Erreur Binaire
TEP	Taux d’ Erreur par Paquet
TFM	Tracking Forecast Memories
UMTS	Universal Mobile Telecommunications System
USB	Universal Serial Bus
VHDL	Very high speed integrated circuit Hardwave Description Language
WCDMA	Wideband Code Division Multiple Access
WiMAX	Worldwide Interoperability for Microwave Acess

Notations

Dans le manuscrit, les notations des matrices et des vecteurs sont désignées en gras, par exemple, $\mathbf{x}$ est le vecteur des symboles, tandis que x est un symbole ou une valeur scalaire en fonction du contexte.

Codage et Décodage

$\mathbf{c}$	Séquence d'information codée.
$\hat{\mathbf{c}}$	Séquence d'information décodée.
$\mathbf{d}$	Séquence d'information numérique compressée.
$\hat{\mathbf{d}}$	Séquence d'estimation de $\mathbf{d}$.
$\mathbf{x}$	Séquence numérique d'information émise.
$\hat{\mathbf{x}}$	Séquence d'estimation de $\mathbf{x}$.
$\mathbf{w}$	Séquence d'information modulée.
$\mathbf{w}'$	Séquence d'information démodulée.
$\mathbf{u}$	Séquence reçue correspondant à la séquence émise $\mathbf{d}$.
$\mathbf{v}_1$	Séquence reçue correspondant à la séquence de la redondance émise $\mathbf{y}_1$.
$\mathbf{v}_2$	Séquence reçue correspondant à la séquence de la redondance émise $\mathbf{y}_2$.
$\mathbf{y}_1$	Séquence de la redondance sortie du premier codeur.
$\mathbf{y}_2$	Séquence de la redondance sortie du deuxième codeur.
d^i	Bit d'information à coder à l'instant i.
y_1^i	Bit de redondance à l'instant i du premier codeur.
y_1^i	Bit de redondance à l'instant i du deuxième codeur.
ν	Longueur du registre à décalage du code convolutif.
$\alpha^i(s)$	Métrique récurrente *aller* à l'instant i de l'état s.
α^i	Vecteur de la métrique récurrente *aller* à l'instant i.
$\beta^i(s)$	Métrique d'état récurrente *retour* à l'instant i de l'état s.
β^i	Vecteur de la métrique récurrente *retour* à l'instant i.
$\gamma^i(s,s')$	Métrique de branche de l'état s à l'instant i à l'état s' à l'instant $i+1$.
γ^i	Vecteur de la métrique de branche à l'instant i.
$\gamma_e^i(s,s')$	Métrique de branche extrinsèque de l'état s à l'instant i à l'état s' à l'instant $i+1$.
γ_e^i	Vecteur de la métrique de branche extrinsèque à l'instant i.

$\Pr(x)$	Probabilité d'un symbole x.
$\Pr_e(x)$	Probabilité extrinsèque d'un symbole x.
$\Pr_a(x)$	Probabilité *a priori* d'un symbole x.
σ	Déviation du bruit.
s	Etat du code convolutif.
s^i	Etat du code convolutif à l'instant i.
s^c	Etat circulaire.
m	Nombre de bits du symbole à coder à chaque instant.
k	Longueur de la séquence d'information $\mathbf{d}$.
n_1	Longueur de la séquence $\mathbf{y}_1$.
n_2	Longueur de la séquence $\mathbf{y}_2$.
R	Rendement du code.
n	Longueur totale de la séquence codée.
$L^i(x)$	Vraisemblance *a posteriori* du symbole x à l'instant i.
$L_a^i(x)$	Vraisemblance *a priori* du symbole x à l'instant i.
$L_e^i(x)$	Information extrinsèque du symbole x à l'instant i.
$\prod$	Entrelaceur / fonction d'entrelacement.
S	Facteur de dispersion de l'entrelaceur.
EM	Edge-Memory.
NDS	Coefficient d'échelle dépendant du bruit.
$\mathbf{A}$	Matrice d'état du codeur.
$\mathbf{B}$	Matrice d'entrée du codeur.
$\mathbf{I}$	Matrice identité.
$(.)^t$	Transposition de la matrice ou du vecteur équivalent.
u_e	Ordre du module exponentiel.
u_l	Ordre du module logarithmique.
ρ	Nombre de flux stochastiques de décodage en parallèle.

Mathématiques

$\mathbb{R}$	Champ des numéros réels.
$\exp(x)$	Opération exponentielle de x avec $x \in \mathbb{R}$.
$\log(x)$	Opération logarithmique de x avec $x \in \mathbb{R}$.
$\langle \mathbf{a}, \mathbf{b} \rangle$	Multiplication vectorielle entre deux vecteurs $\mathbf{a}$ et $\mathbf{b}$.
$\|\mathbf{a} - \mathbf{b}\|$	Distance entre deux vecteurs $\mathbf{a}$ et $\mathbf{b}$.

Table des matières

Introduction

Le développement de nouveaux systèmes de télécommunication a pour objectif principal la transmission d'information numérique à des débits toujours plus élevés et pour une qualité de service de plus en plus exigeante. Les études dans le domaine de la transmission d'information ont débuté avec les travaux de Shannon en 1948. Shannon a démontré qu'une transmission d'information fiable était possible lorsque le taux de données d'information est plus petite que la capacité du canal de transmission. Cette capacité de transmission est obtenue par l'utilisation de codes correcteurs d'erreurs qui permettent de corriger des erreurs introduites au cours de la transmission sur le canal. Pendant 40 ans, la communauté scientifique a ensuite tenté de construire des codes correcteurs d'erreurs pour atteindre la limite théorique de Shannon. Malheureusement, aucun des codes proposés ne s'est approché de cette limite d'une valeur inférieure à 2 dB. Le fait d'atteindre cette "capacité théorique du canal" constituait un véritable défi pour les chercheurs. La communauté scientifique a dû attendre l'année 1993 pour que Claude Berrou et Alain Glavieux proposent les turbocodes qui s'approchent de cette limite [1]. Les turbocodes permettent d'atteindre à quelques dixièmes de dB la limite de Shannon, pour un TEB - Taux d'Erreur Binaire - de 10^{-5}, avec une complexité calculatoire relativement faible.

Le principe de cette famille de code est basé sur un concept simple qui consiste en une décomposition d'un problème complexe en plusieurs sous-problèmes qui sont faciles à résoudre. Les turbocodes sont construits par une association de plusieurs - conventionnellement deux - décodeurs composants simples connectés par une fonction de permutation temporelle, appelée *entrelaceur*. Ces décodeurs composants collaborent par un échange itératif des messages probabilistes associés aux symboles reçus afin de corriger des erreurs de transmission. Le principe du décodage itératif a permis de franchir une étape importante et d'aboutir à l'implémentation d'un décodeur adapté à un tel code. Par ailleurs, le principe de décodage itératif a favorisé la redécouverte des codes LDPC (*Low Density Parity Check*) [2] inventés par Gallager en 1962 [3]. Les excellentes performances des turbocodes expliquent leur intégration dans les normes de communications spatiales (CCSDS, DVB-RCS, DVB-RCT) et de communication sans fil (UMTS, LTE, WiMAX). Les premières applications exploitant les turbocodes produisaient un faible débit, seulement quelques Mbits/s. Cependant, avec le développement des services sans fil et les besoins de transmission à très haut débit, l'exigence des implémentations de turbo-décodeurs à haut débit jusqu'au Gbits/s devient indispensable. La conception de telles architectures matérielles pour turbo-décodeurs à haut débit est donc un défi ouvert.

Parmi les architectures de décodage à haut débit les plus efficaces, la plus originale est celle proposée par l'équipe de Warren J. Gross (Université de McGill, Canada) ce qui atteint 61,3 Gbits/s sur 6,38 mm^2 de technologie ASIC (*Application-Specific Integrated Circuit*) 90 nm [4]. Les codes employés sont des codes LDPC et non pas des turbocodes. L'architecture de ce décodeur a une spécificité : le traitement n'est pas réalisé de manière classique en virgule fixe, mais dans le domaine logique stochastique. Le principe du calcul stochastique a été présenté dans les années 1960 par Gaines [5]. Il a récemment été appliqué comme une technique de traitement probabiliste d'algorithme de décodage de codes correcteurs d'erreurs. Dans ce domaine, les messages

probabilistes reçus à partir du canal sont transformés en des séquences de Bernoulli. Un fort avantage du calcul stochastique est qu'il permet de traiter les opérations arithmétiques complexes sur les probabilités avec un coût matériel faible. Cette approche a d'abord été utilisée pour décoder des codes en graphe de petites tailles : les codes Hamming, les codes BCH et les codes LDPC. Dans un premier temps, l'approche de décodage stochastique avait des performances médiocres et atteignait des débits modestes lorsqu'elle était envisagée pour décoder des codes de taille réaliste. En effet, les décodeurs correspondant nécessitaient une séquence de Bernoulli très longue pour converger.

Durant la décennie suivante, des décodeurs stochastiques permettant de décoder des codes LDPC de taille suffisamment grande ont été proposés. Les résultats obtenus ont prouvé que l'approche stochastique pouvait être une nouvelle perspective de recherche permettant de construire un décodeur capable de fournir un haut débit avec une complexité raisonnable pour le décodage itératif de codes correcteurs d'erreurs puissants. Cependant, à notre connaissance, jusqu'à présent, aucune extension de l'approche du décodage stochastique pour construire une architecture de turbo-décodeur à haut débit n'a été proposée.

Problématique et Objectifs

Dans le contexte de cette thèse, nos travaux ont pour objectif d'appliquer l'approche de décodage stochastique au décodage de turbocodes. Nous nous sommes focalisés sur des turbocodes convolutifs et leur débit de décodage stochastique à travers l'exploration de solutions algorithmiques et architecturales. Rappelons que ces travaux constituent les premiers pas dans le domaine du décodage stochastique de turbocodes. A première vue, il semble que les turbocodes et les codes LDPC possèdent des similarités ; c'est pourquoi, les techniques d'optimisation utilisées avec succès pour décoder des codes LDPC ont été examinées pour éteindre les bonnes performances de décodage stochastique aux turbocodes.

Ainsi, l'étude a consisté à proposer des solutions algorithmiques et architecturales afin de concevoir une architecture matérielle de turbo-décodeur stochastique possédant un haut débit. L'architecture proposée doit, quant à elle, conserver des performances de décodage similaires à celles d'une architecture de turbo-décodeur classique. La mise en œuvre, quant à elle, a consisté à intégrer un turbo-décodeur stochastique sur une plate-forme matérielle à base de circuit FPGA. Pour ce faire, les objectifs ont donc été de :

- démontrer la faisabilité et la validité des études algorithmiques et architecturales,
- estimer la complexité matérielle des solutions architecturales proposées,
- déterminer le débit de décodage dans un contexte réaliste.

Contributions

Pour répondre à ces objectifs, des contributions algorithmiques et des contributions architecturales ont été faites durant la thèse. Elles peuvent être récapitulées comme suit :

Contributions algorithmiques :

Elles portent sur l'étude pour le calcul stochastique dans les architectures de turbo-décodeur, à savoir :

- La conception de l'algorithme de décodage stochastique pour des turbocodes.
- L'analyse des caractéristiques des opérations stochastiques au niveau de la convergence de turbo-décodeurs stochastiques.
- La construction de l'architecture d'un turbo-décodeur stochastique à partir des éléments basiques du calcul stochastique.
- L'optimisation de la performance de turbo-décodeur stochastique en utilisant des techniques proposées pour décoder des codes LDPC.

Contributions architecturales :

Elles portent sur la conception d'une architecture stochastique à haut débit dédiée aux turbo-codes, à savoir :

- L'analyse de la contribution des opérations stochastiques à la convergence d'un décodeur convolutif stochastique.
- La proposition d'une solution alternative pour l'opération qui ralentit la convergence du décodeur convolutif stochastique. Cette solution nous permet la proposition d'un premier type de décodage convolutif stochastique.
- L'analyse de l'impact du décodage parallèle en termes de débit atteint sur des implémentations récentes. Cette analyse permet la proposition d'une deuxième solution de décodage stochastique parallèle pour les turbocodes.
- La combinaison de ces deux approches, avec pour but de construire une architecture de turbo-décodeur stochastique ayant un haut débit en assurant une bonne performance de décodage.
- L'intégration et le prototypage de deux décodeurs stochastique sur une plate-forme de base FPGA :
  1. Un décodeur de code convolutif.
  2. Un turbo-décodeur.

Plan du mémoire :

Ce mémoire est organisé en 4 chapitres :

Dans le chapitre 1, nous introduisons les notions de base nécessaires pour aborder les autres chapitres du manuscrit. Nous présentons d'une part les codes correcteurs d'erreurs, en particulier

les codes convolutifs et les turbocodes. Dans le contexte de notre travail, nous nous intéressons à des codes convolutifs systématiques récursifs circulaires (CRSC). L'algorithme de décodage itératif correspondant qui utilise des décodeurs SISO (*Soft-Input Soft-Output*) est ensuite décrit - l'algorithme MAP (*Maximum A Posteriori*). D'autre part, nous donnons le principe du décodage stochastique, le problème associé au décodage stochastique - problème de corrélation - ainsi que les solutions proposées dans la littérature. Une investigation complète du décodage stochastique des turbocodes est alors présentée au niveau l'algorithme. Enfin, nous situons la problématique associée aux opérations du calcul stochastique lorsqu'elles sont appliquées aux turbocodes.

Dans le chapitre 2, nous abordons les aspects algorithmiques du décodage stochastique des turbocodes. Un modèle générique d'un turbo-décodeur stochastique composé de deux décodeur stochastiques SISO est présenté. Ensuite, nous détaillons l'architecture des différents modules élémentaires constituant un décodeur stochastique SISO. Au sein de chaque module, nous proposons différentes approches pour tenter de supprimer le problème de corrélation d'un système de turbo-décodage stochastique. Des courbes de performance en termes de taux d'erreurs binaires - TEB ainsi qu'une estimation de la complexité calculatoire sont données, puis comparées avec celles du turbo-décodage SubMAP classique. Une évaluation du débit est également présentée, ce qui permet de situer notre solution architecturale en termes de débit.

Le chapitre 3 se focalise sur deux nouvelles approches ayant pour but de construire un turbo-décodeur stochastique atteignant un débit élevé. Le choix de ces deux solutions architecturales pour un turbo-décodeur stochastique est guidée par la volonté d'obtenir un haut débit de décodage tout en limitant les dégradations qui en découlent en termes de performance. La première se base sur les caractéristiques des opérations de calcul stochastique. Par une analyse de la contribution des opérations de base, en particulier l'opération de type addition, au niveau de la convergence du décodeur stochastique, nous proposons de remplacer cette opération par d'autres opérations plus simples. La seconde contribution traite une solution de décodage en parallèle avec mutualisations de ressources matérielles. Puis, nous associons ces deux approches et proposons une architecture globale dans le but d'accélérer le processus de décodage stochastique des turbocodes.

A partir des résultats obtenus précédemment, dans le chapitre 4, nous mettons en œuvre deux décodeurs stochastiques - décodeur stochastique SISO et turbo-décodeur stochastique - sur une plate-forme à base d'un circuit FPGA. L'objectif du prototypage est de valider et d'estimer l'efficacité des algorithmes et des architectures retenus pour ces deux décodeurs stochastiques. Pour atteindre cet objectif, nous avons implémenté une chaîne de communication numérique comprenant un émetteur, un émulateur de canal et un décodeur stochastique. Deux différents prototypes correspondant à deux types de décodeurs sont alors présentés. Des performances mesurées à partir de l'implémentation des deux prototypes sont données. Enfin, des évaluations de la complexité et du débit nous permettent d'analyser l'impact de nos approches.

CHAPITRE 1

Vers les turbo-décodeurs à haut débit

Sommaire

1.1 Introduction

Depuis l'invention des turbocodes, la majorité des standards sans fil avancés tels que WiMAX et 3GPP-LTE adopte différents schémas de turbocodes afin d'obtenir une transmission fiable sur les canaux bruités. Chaque standard exploite différents types de turbocodes convolutifs (TCC) binaires ou non-binaires, pour lesquels la longueur des blocs se situe entre quelques dizaines et quelques milliers de bits (40-6144) comme détaillé dans le tableau 1.1.

Standard	UMTS (3GPP)			IEEE 802.16 (WiMAX)	
	WCDMA	HSDPA	LTE	WiMAX Mobile	WiMAX fixe
Spécification	Rel.99	Rel.5	Rel.8	802.16e	802.16d
Débit Max.	2 Mbps	14,4 Mbps	100 Mbps	15 Mbps	75 Mbps
TEB	10^{-3}			10^{-5}	
Schéma TCC	simple-binaire			double-binaire	
Bits/bloc	40-5114		64-6144	48-4800	

TABLE 1.1 – L'augmentation du débit de nouveaux standards de réseau sans fil devient de plus en plus importante. Néanmoins, le débit est uniquement de l'ordre maximum de 100Mbps.

L'évolution des standards de réseau sans fil permet d'augmenter le débit de transmission. Cependant, ces débits restent en général encore modestes par rapport aux réseaux filaires, maximum de 100 Mbps pour la norme 3GPP-LTE. Récemment, avec le développement des services sans fil et les besoins de la transmission à très haut débit, comme la transmission de vidéo (DVB) haute définition, un débit de l'ordre du Gbps [6] est exigé. Ainsi, de nombreuses approches sont proposées dans la littérature afin d'augmenter le débit de traitement des turbo décodeurs. Des architectures de turbo décodeurs sont donc implémentées sur ASIC (*Application-Specific Integrated Circuit*) et ont des résultats de débit de plus en plus élevés, comme donné dans le tableau 1.2.

	HSDPA	WiMAX		LTE		Autres		
	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]
Technologie	180nm	130nm	90nm	65nm	130nm	65nm	130nm	90nm
Débit(Mbps)	24	115,4	250	152	390	711	952	1400

TABLE 1.2 – Plusieurs implémentations des turbocodes sont proposées pour la montée en débit. Néanmoins, très peu d'architectures atteignent le débit de l'ordre du Gbps.

La première architecture de turbo décodeur qui peut atteindre un débit supérieur au Gbps a été implémentée sur ASIC [14]. Cependant, l'implémentation de ces turbo décodeurs demeure un défi majeur en termes de complexité matérielle [15]. C'est pourquoi, d'autres techniques doivent être envisagées.

Ces dernières années, l'application du calcul stochastique au décodage des codes correcteurs d'erreurs LDPC (Low Density Parity Check) s'est avérée une méthode efficace qui offre des résultats notables en termes de débit et de la complexité matérielle. Cette technique permet d'atteindre

des débits de l'ordre du Gbps : 1,66 Gbps sur une carte FPGA(*Field Programmable Gate Array*) [16] et 61,3 Gbps sur un ASIC [4]. Ces résultats permettent de nouvelles perspectives de recherche afin de réaliser des architectures stochastiques de turbo décodeur à haut débit avec de complexités compatibles avec les contraintes d'intégration.

Dès lors, ce chapitre s'organise de la manière suivante :

- **La première section** donne une vue d'ensemble des concepts fondamentaux du codage et aussi du décodage pour les turbocodes. Cette section se focalise plus précisément sur les codes convolutifs, particulièrement les codes convolutifs systématiques récursifs circulaires (CRSC). Les turbocodes sont ensuite introduits comme une concaténation en parallèle des codes CRSC, séparés par un entrelaceur. Le principe du décodage itératif des turbocodes est aussi abordé à partir de l'algorithme optimal BCJR [17].

- **La deuxième section** présente le principe du calcul stochastique avec les opérations arithmétiques utilisées pour les turbo décodeurs. Les contraintes du décodage stochastique pour les codes LDPC et les améliorations sont également présentées.

- **La dernière section** introduit l'extension du traitement stochastique au décodage des turbocodes. Après quelques discussions des caractéristiques des opérations stochastiques, le problème le plus important du décodage stochastique des turbocodes est ensuite mentionné dans cette section.

1.2 État de l'art sur les turbo décodeurs à haut débit

1.2.1 Rôle des codes correcteurs d'erreurs en communication numérique

Une chaîne de communication permet de transmettre une quantité d'information donnée avec le moins d'erreurs possible. Les données peuvent être numériques - un fichier texte ou une image, ou analogiques - une source sonore. Cependant, grâce à des flexibilités et des puissances de traitement dans le temps discret, les systèmes de communication modernes sont la plupart numériques. C'est pourquoi, lorsque les données sont analogiques, elles doivent être numérisées avant d'être transmises de l'émetteur au récepteur. La figure 1.1 représente un modèle simplifié d'une chaîne de communication numérique.

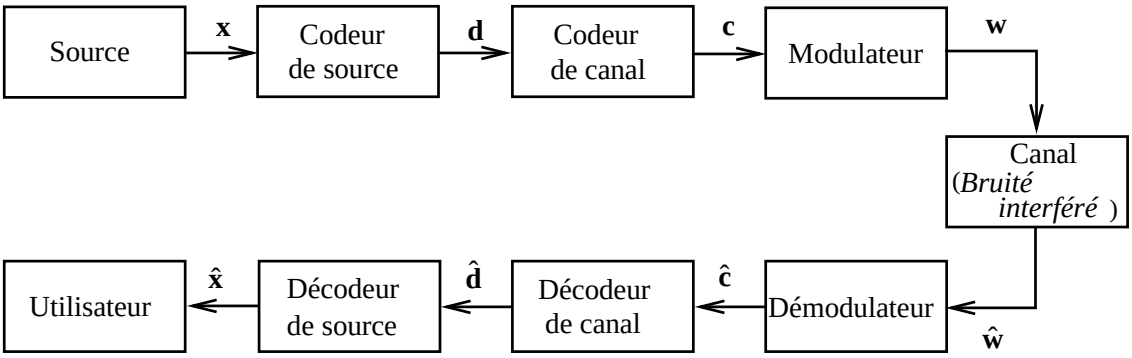


FIGURE 1.1 – Chaîne de transmission numérique.

La chaîne de transmission en Figure 1.1 est constituée d'un milieu de transmission - appelé *canal*, d'un couple codeur-décodeur de source, d'un couple modulateur-démodulateur ainsi que d'un couple codeur-décodeur de canal. La donnée d'information de la source est représentée par une séquence numérique qui est à l'entrée du codeur de source. Durant cette étape, la séquence numérique $\mathbf{x}$ sera compressée au maximum. Ensuite, le codeur de canal ajoute à la séquence compressée de la redondance produite par certaines lois qui permettent au récepteur de vérifier les erreurs et de les corriger. La séquence de sortie du codeur de canal est notée $\mathbf{c}$ avec une longueur n. Elle est envoyée au modulateur, qui transforme la séquence numérique $\mathbf{c}$ en signaux continus $\mathbf{w}$ compatibles avec le milieu de transmission. L'émetteur envoie une donnée au récepteur à travers un milieu de transmission, où la donnée peut être déformée ou interférée. Le canal est dépendant du milieu physique par lequel transitent les informations et dans lequel $\mathbf{w}$ est bruité. Dans les communications mobiles, plusieurs types de canaux de transmission peuvent être utilisés pour les échanges d'information entre l'émetteur et le récepteur. Les techniques de modulation, non détaillés ici, varient de simples techniques - utilisant des symboles complexes discrets - aux techniques plus avancées - Orthogonal Frequency Division Multiplexing (OFDM) ou multiantennes (MIMO).

Le signal analogique $\mathbf{w}$ produit par le modulateur est altéré en un autre signal analogique $\hat{\mathbf{w}}$ qui est le point d'entrée du démodulateur, le premier bloc du récepteur. Le modulateur qui réalise l'opération inverse du modulateur convertit le signal analogique $\hat{\mathbf{w}}$ en signal numérique en bande de base $\hat{\mathbf{c}}$ équivalent. Ce signal est l'estimation du signal $\mathbf{c}$ et est fourni au décodeur de canal. Le

décodeur de canal s'appuie sur la connaissance de l'ensemble des mots de codes pour choisir, à partir d'un mot reçu, le plus vraisemblable. Le résultat du décodage du canal est noté $\widehat{\mathbf{d}}$, et est utilisé par le décodeur de source pour produire l'estimation du signal numérique émis $\widehat{\mathbf{x}}$.

Le bruit modifie le signal émis jusqu'à un niveau tel qu'à la réception, le signal reçu est alors différent du signal émis. Lorsque l'estimation $\widehat{\mathbf{d}}$ est différente du signal émis $\mathbf{d}$, alors, l'estimation $\widehat{\mathbf{x}}$ et le signal numérique $\mathbf{x}$ ne sont pas identiques. Dans ce cas, une erreur de transmission est rencontrée s'il n'y a pas de moyen de protection des données. Une solution est d'introduire un mécanisme de codage/décodage approprié. L'émetteur ajoute au signal numérique émis de la redondance qui est utilisée par le récepteur afin de détecter les erreurs voire les corriger.

A travers l'histoire des communications numériques, parmi les solutions proposées pour la protection des données, les turbocodes sont reconnus comme révolutionnaires au niveau des performances de correction d'erreurs et au principe de décodage itératif. Dans la section suivante, leurs principes fondamentaux sont rappelés.

1.2.2 Principes généraux des turbocodes

En 1948, Shannon a introduit le théorème célèbre : *Si le débit de l'information à l'entrée du canal est inférieur à la capacité du canal, alors il est possible de transmettre le contenu de l'information sur le canal avec une probabilité d'erreur aussi petite que souhaitée*. Ce théorème est d'une importance fondamentale dans le monde des communications numériques. Il donne une limite supérieure que l'on peut atteindre en termes de taux de transmission des données fiables pour un canal de transmission. Cependant, pour prouver ce théorème, Shannon a utilisé un code *aléatoire* de longueur infinie. En pratique, ce code n'existe pas. Depuis, les chercheurs s'efforcent de construire des codes correcteurs d'erreurs de longueurs finies s'approchant du code aléatoire, pour être capable d'atteindre au maximum la limite du théorème de Shannon.

Ainsi, la communauté scientifique a attendu presqu'un demi-siècle pour véritablement s'approcher de la limite de Shannon. En 1992, C. Berrou et A. Glavieux, membre de l'ENST Bretagne, ont inventé les turbocodes qui ont été présentés à la communauté scientifique en 1993 [1]. Ils attirent l'attention grâce à leur principe de décodage qui permet d'obtenir de bonnes performances de correction d'erreurs proches de la limite de Shannon. De plus la complexité de mise en œuvre est acceptable pour une implémentation matérielle.

La construction des turbocodes se base sur la théorie de concaténation. Le premier turbocode est construit par une concaténation en parallèle de deux codes élémentaires séparés par un entrelaceur. Son principe de décodage se base sur un algorithme, *itératif*, où deux unités échangent des informations au cours de chaque itération afin d'améliorer la capacité de correction d'erreurs de chaque unité. Après plusieurs itérations, les deux unités convergent vers un même mot de code, qui est identique au mot de code transmis. Le principe itératif a stimulé la communauté scientifique pour étendre le principe itératif. Il a donc été adopté dans d'autres parties du système de transmission : égalisation [18], détection [19],... Dans la suite, nous allons revoir ensemble brièvement les briques des turbocodes : la concaténation des codes et aussi l'entrelacement.

1.2.2.1 Théorie de la concaténation

En 1957, Shannon a énoncé un principe fondamental : *la probabilité d'erreur du décodage peut être réalisée afin de diminuer exponentiellement en fonction de la longueur de paquet des données qui tend vers l'infini* [20]. Ainsi, plus longue est la séquence de donnée, plus exponentiellement faible est la probabilité d'erreur. La probabilité d'erreur devient la meilleure quand la longueur de la séquence tend vers l'infini. Néanmoins, le théorème de codage démontre que la complexité du schéma de décodage optimal qui simplifie les calculs des vraisemblances sur tous les mots codés transmis croît aussi exponentiellement. Il nécessite aussi une mémoire exponentiellement large suivant la taille de la séquence d'information. C'est pourquoi, l'implémentation d'un tel décodeur optimal est impossible [21].

En 1966, Forney a proposé pendant son travail de thèse un type de code [22], appelé *les codes enchaînés* comme une approche aboutissant à un fort gain de codage, pour une complexité raisonnable, en combinant deux codes simples voire plus. Ce type de code permet d'obtenir une probabilité d'erreur qui décroît exponentiellement et qui est proche de la limite de Shannon. Mais la complexité de décodage augmente tout de même polynomialement avec la longueur de la trame.

Le schéma original de concaténation des codes proposé par Forney est illustré en figure 1.2(a). Ce type de code se compose d'un codeur, dit codeur *extérieur*, connecté à un autre codeur, noté codeur *intérieur*. Il s'agit d'un code *concaténé en série*. Quelques années après, une concaténation classique et efficace proposée par Odenwalder [23], utilise pour code intérieur un code convolutif binaire, et pour code extérieur un code en bloc linéaire de type Reed-Solomon (RS). La NASA a ensuite standardisé la concaténation série des codes en ajoutant un entrelaceur entre les deux codes élémentaires (figure 1.2(b)) qui permet d'augmenter significativement la robustesse des codes concaténés. Dès lors, la concaténation en série des codes correcteurs d'erreurs a été largement utilisée dans tous les types de communications numériques.

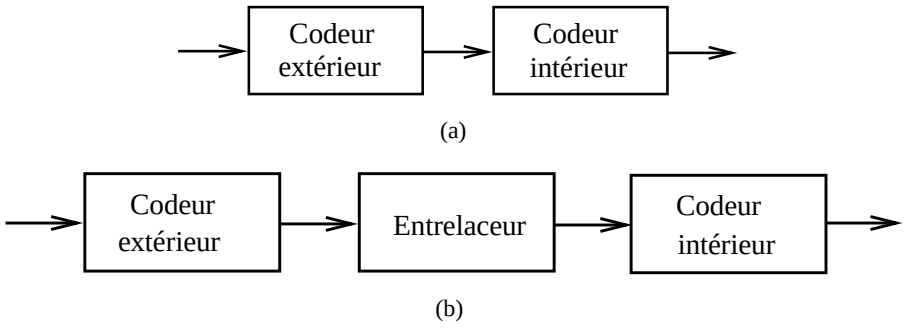


FIGURE 1.2 – Structure des concaténations.

En 1993, une nouvelle architecture de concaténation des codes a été introduite : *la concaténation parallèle de codes* comme le montre la figure 1.3. Elle est associée aux turbocodes [1]. Cette structure originale se compose des deux codes convolutifs systématiques récursifs élémentaires en parallèle qui codent le même message dans l'ordre naturel et dans un ordre permuté.

Dans cette structure de concaténation, le nombre de codes élémentaires utilisés est défini comme la *dimension* du code concaténé. Afin d'atteindre une grande capacité de transmission,

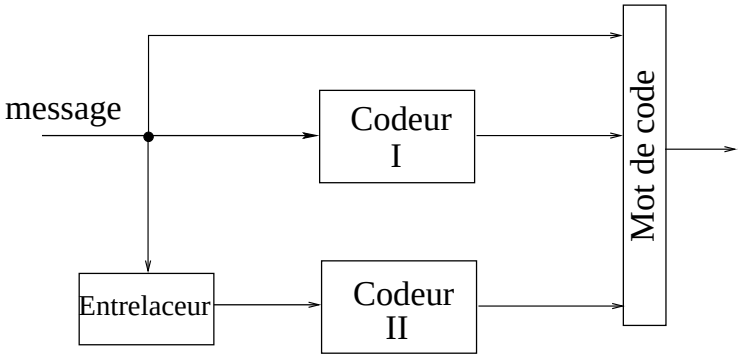


FIGURE 1.3 – Concaténation parallèle des codes.

les codes concaténés peuvent comprendre plusieurs codes élémentaires. Il s'agit alors des codes multi-dimensionnels. En supposant que la dimension est 2 pour le code concaténé, et que les rendements des deux codes composants sont respectivement R_1 et R_2, alors le rendement total $R_p = \frac{R_1 R_2}{1-(1-R_1)(1-R_2)}$ du code concaténé en parallèle (figure 1.3) est plus grand que celui du code concaténé en série $R_s = R_1 R_2$ (figure 1.2(a)). C'est pourquoi, à partir de capacités de correction identiques, le code de concaténation en parallèle donne un meilleur rendement de codage que celui en série. Toutefois, cela implique de ne pas envisager plus de 2 dimensions, car le rendement du code concaténé décroît plus rapidemment dans le cas parallèle. En 1997, en utilisant deux codes convolutifs à 4 états [24], Benedetto a montré que le code de concaténation en série permettait d'atteindre une probabilité d'erreurs des bits inférieure grâce à une meilleure distance minimale de Hamming, tandis que le code de concaténation en parallèle atteignait de meilleure performance par rapport à la limite théorique. Grâce à cette caractéristique, les codes de concaténation en parallèle sont largement utilisées dans les standards de communication actuels. Par exemple UMTS, CDMA2000 et 3GPP-LTE (*Long Term Evolution*) [25] utilisent une concaténation en parallèle des codes convolutifs binaires à 8 états ; DVB-RCT (canal de retour pour la diffusion télévisuelle terrestre) [26], DVB-RCS (canal de retour pour la diffusion télévisuelle satellitaire) [27], WiMax (IEEE 802.16) [28], skyplex (Eutelsat) utilisent une concaténation en parallèle de codes convolutifs double-binaires à 8 états ; les systèmes standardisés pour les transmissions satellitaires par CCSDS [29], INMARSAT utilisent une concaténation en parallèle de codes convolutifs binaires à 16 états. Des architectures mixtes qui combinent des concaténations en parallèle et série sont également proposées [30], [31].

1.2.2.2 Entrelacement

Le rôle de la fonction d'entrelacement est particulièrement important. C'est une des caractéristiques principales du turbo codage [32] qui affecte significativement les performances. La performance d'un turbocode est essentiellement amélioré quand la longueur d'entrelaceur augmente [33]. Au niveau du codage, en 1996, Benedetto [32] et Perez [34] ont prouvé que la performance asymptotique dépendait de la distribution des mots codés de poids faible. Par conséquent, la fonction d'entrelacement permet d'éviter que les deux codeurs composants donnent un mot codé de poids faible, et ainsi augmente la distance minimale de Hamming. Si une séquence d'in-

formation qui génère un mot codé de poids faible à l'entrée d'un codeur élémentaire est entrelacée selon certaines critères, le processus aboutit à un mot codé de grand poids sous l'autre codeur élémentaire.

Au niveau du décodage, l'entrelaceur assure à la sortie de chaque décodeur élémentaire, une dispersion temporelle des erreurs groupées dans des paquets de données. Aussi, elles deviennent des erreurs isolées pour l'autre décodeur qui suit. En augmentant le nombre d'échange des informations du processus de décodage, la probabilité d'erreur des bits s'approche de la limite théorique. La propriété de corrélation du codage concaténé influence le turbo décodage, notamment au niveau de la zone de convergence où le taux d'erreurs binaires chute dès les premières itérations.

La construction d'un bon entrelaceur doit tenir compte des motifs des schémas des codes élémentaires ainsi que des motifs d'erreur du canal de propagation. La conception d'entrelaceurs a été abondamment étudiée dans la littérature. Parmi les classes d'entrelaceurs, on peut citer les entrelaceurs simples et performants tels que *Almost Regular Permutation - ARP* [35] et *Dithered Relatively Prime - DRP* [36] qui ont inspiré les standards DVB-RCS, DVB-RCT et WiMAX (IEEE 802.16). Plus récemment l'entrelaceur *Quadratic Permutation Polynomial - QPP* a été sélectionné pour la norme 3GPP-LTE [25].

1.2.3 Turbocodes convolutifs retenus pour les standards

Parmi différents codes élémentaires proposés dans la littérature, les codes convolutifs sont largement utilisés dans les schémas des turbocodes grâce à leur grande capacité de correction d'erreur et à leur faible coût du schéma de codage et de décodage. De différentes architectures des turbocodes convolutifs sont largement déployées dans les standards modernes : 3GPP-LTE [25], DVB-RCT [26], DVB-RCS [27]. Dans un premier temps, cette section décrit brièvement le principe de turbo codage et turbo décodage convolutifs. Les principes basiques des codes convolutifs et une classe importante des codes convolutifs - classe des codes convolutifs récursifs systématiques circulaires - sont ensuite rappelés. Le principe de l'algorithme de décodage optimal - algorithme MAP - est introduit à la fin de cette section.

1.2.3.1 Turbo codage des codes convolutifs

Le premier schéma de turbo codage convolutif proposé par Berrou [1] est présenté en figure 1.4. Ce code est construit par une concaténation en parallèle de deux codes convolutifs systématiques récursifs (RSC) identiques RSC_1 et RSC_2 séparés par un entrelaceur. Chaque codeur constituant est illustré en figure 1.7. Ce type de code est appelé "turbocode" par référence à son principe de décodage itératif. La séquence d'information d'entrée est codée deux fois par les deux codeurs élémentaires. L'entrelaceur Π est placé avant le deuxième codeur et effectue une permutation sur la séquence d'information. Alors, les deux codeurs composants codent la même séquence d'information mais en ordres différents. Dans le cadre de cette étude, nous nous limitons à l'utilisation d'un entrelaceur de type *S*-random [37].

La séquence d'information **d** est fournie au premier codeur RSC_1 et aussi transmise directement comme séquence systématique. Le codeur RSC_1 produit la première séquence de redon-

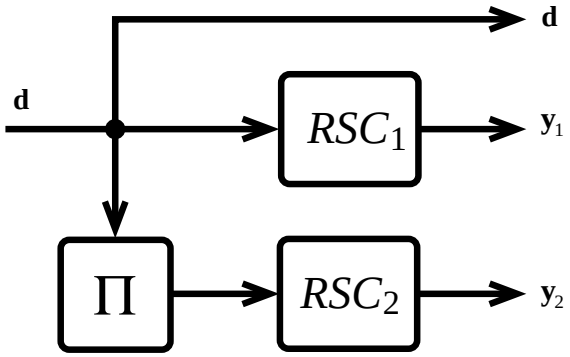


FIGURE 1.4 – Description du schéma de codage.

dance $\mathbf{y}_1$. Parallèlement, la séquence d'information d'entrée est permutée par l'entrelaceur Π. La séquence entrelacée est ensuite codée par le second codeur RSC_2. Ce codeur fournit la seconde séquence de redondance $\mathbf{y}_2$. Le symbole de turbocode possède des bits systématiques et des bits de parité générés par les deux codeurs RSC_1 et RSC_2. Le mot codé par ce schéma à l'instant est défini $c^i = (d^i y_1^i y_2^i)$. Considérant que la séquence d'information en entrée comprend k bits, les deux redondances produisent n_1 et n_2 bits respectivement. Dans ce cas, le rendement du turbocode est donné par :

$$R = \frac{k}{k+n_1+n_2} = \frac{k}{n} \tag{1.1}$$

Après l'étape du codage, la séquence d'information et deux séquences de redondance sont transmises sur le canal bruité et fournies au récepteur.

1.2.3.2 Turbo décodage des codes convolutifs

A la réception, le signal bruité est démodulé et transmis au décodeur. L'algorithme optimal de décodage *Maximum Likelihood - ML* est appliqué pour décoder le vecteur de données reçues basé sur la structure en treillis. Cependant, en raison de l'entrelaceur, le trellis du turbocode possède un grand nombre d'états. Ceci complexifie l'algorithme de décodage *ML* qui s'avère impossible à implémenter en pratique pour une grande longueur d'entrelaceur. C'est la raison pour laquelle, chaque code composant est décodé par un décodeur attitré. Les informations d'entrée et de sortie de chaque décodeur sont relatives à celles du codeur correspondant. Ainsi, un décodeur peut profiter de l'information de sortie de l'autre décodeur comme information d'entrée. Cet échange d'information est considéré comme le principe de décodage itératif, et les informations échangées, nommées *informations extrinsèques*. Il faut souligner que les informations d'entrée de chaque décodeur constitué ainsi que les informations échangées entre les décodeurs sont naturellement pondérées. Chaque décodeur est désigné comme un décodeur à entrées et sorties pondérées (*SoftInput Soft Output* - SISO). Par la suite, le principe de décodage itératif est décrit pour les turbocodes à deux dimensions.

Le schéma original est composé de deux décodeurs SISO, de deux entrelaceurs et d'un désentrelaceur comme illustré sur figure 1.5. $\mathbf{u}$ est la séquence d'information reçue correspondant à la séquence d'information émise $\mathbf{d}$, $\mathbf{v}_1$ et $\mathbf{v}_2$ sont les séquences d'informations bruitées associées res-

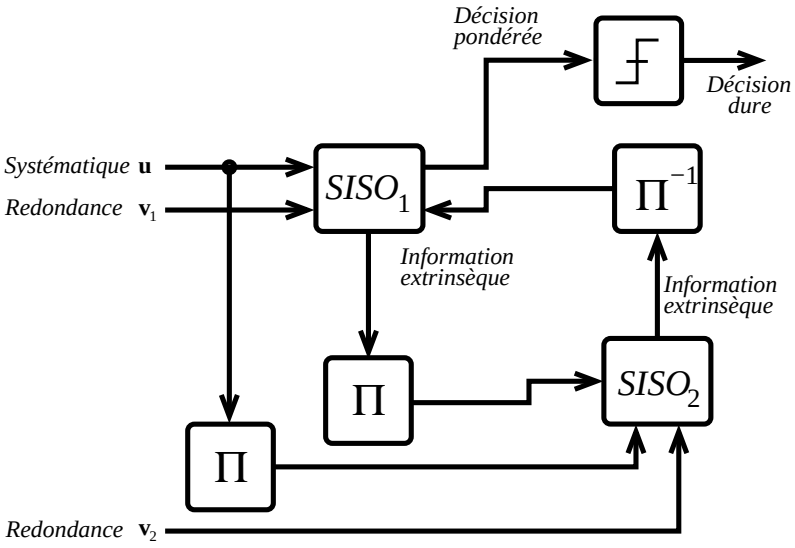


FIGURE 1.5 – Schéma de principe d'un turbo-décodeur.

pectivement aux séquences de redondances $\mathbf{y}_1$ et $\mathbf{y}_2$. L'information échangée entre les décodeurs élémentaires est décrite sous la forme d'une probabilité conditionnelle du bit d'information émis sur les observations à l'entrée du décodeur. Dans le cas du code binaire, l'information extrinsèque du bit d^i avec $i \in (1...k)$ est formulée comme $P_e(d^i = 1|\mathbf{u},\mathbf{v}_1)$ ou $P_e(d^i = 0|\mathbf{u},\mathbf{v}_1)$. Les grandeurs de cette quantité sont très souvent considérées sous une forme logarithmique telles que des *Logarithmes de Rapport de Vraisemblance - LRV*. Par exemple, l'information extrinsèque fournie par le premier décodeur $SISO_1$ et transmise au second décodeur $SISO_2$ est :

$$LRV^{e}_{1\to 2}\left(d^i\right) = \ln\left(\frac{\Pr_e\left(d^i = 1|\mathbf{u},\mathbf{v}_1\right)}{\Pr_e\left(d^i = 0|\mathbf{u},\mathbf{v}_1\right)}\right) \tag{1.2}$$

Les informations extrinsèques produites par un décodeur sont ensuite considérées comme de l'information *a priori* en entrée de l'autre décodeur. Le processus de décodage itératif du turbo-décodeur peut se décomposer en deux étapes :

- Le décodeur $SISO_2$ réalise le décodage à partir de trois contributions : la séquence de redondance $\mathbf{v}_2$, la séquence d'information $\mathbf{u}$ entrelacée qui correspond au processus de codage du second codeur RSC, l'information extrinsèque entrelacée sortie du premier décodeur $SISO_1$. Ensuite, le décodeur $SISO_2$ produit une sortie pondérée qui correspond à la fiabilité de chaque bit décodé. A partir de cette information, l'information extrinsèque correspondant à la séquence $\mathbf{v}_2$ est produite, entrelacée et transmise au décodeur $SISO_1$.
- Le décodeur $SISO_1$ utilise l'information extrinsèque fournie par le second décodeur $SISO_2$ afin de décoder la séquence d'information basée sur ses propres séquences d'observation : $\mathbf{u}$ et $\mathbf{v}_1$. Ce décodeur va ensuite produire une information fiable sur chaque bit décodé correspondant à la séquence $\mathbf{v}_1$. Cette information est par la suite entrelacée et utilisée par le décodeur $SISO_2$ lors de l'itération suivante.

Ces deux étapes d'échange d'informations extrinsèques entre les deux décodeurs composants sont définies comme une *itération* de décodage. Et le processus au cours duquel un décodeur

réalise le décodage sur les séquences reçues et produit l'information extrinsèque pour l'autre décodeur est également défini comme une demi-itération. Au cours des itérations, des erreurs de transmission sont corrigées et les deux décodeurs fournissent leurs séquences d'information décodées qui sont les plus proches possibles de la séquence d'information transmise. Classiquement, le turbo-décodeur prend une décision sur la sortie du premier décodeur $SISO_1$ comme la décision globale.

L'introduction des turbocodes a aussi permis de redécouvrir une autre famille de codes : des codes LDPC (Low-Density-Parity-Check) [2] inventé par Gallager en 1962 [3]. En outre, le principe de décodage itératif a aussi été déployé afin de décoder les turbocodes en bloc [38].

1.2.3.3 Codes Convolutifs

Les premières concepts des codes convolutifs ont été présentées par P.Elias [39] en 1955. La figure 1.6 représente l'exemple du codeur décrit dans sa publication.

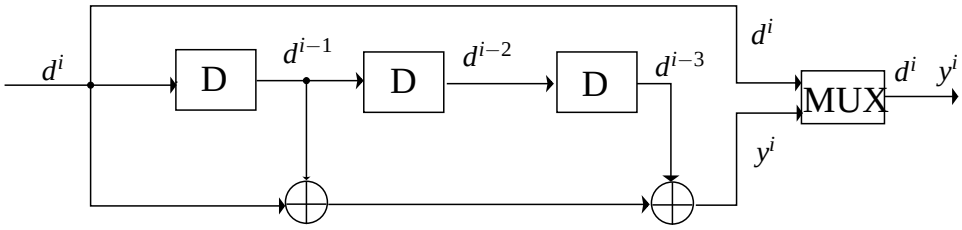


FIGURE 1.6 – Code convolutif systématique des symboles binaires à huit états

Le codeur est conçu autour d'un registre à décalage de ν éléments de mémoires (dans ce cas $\nu = 3$) qui temporise la séquence d'information d'entrée. La quantité $\nu + 1$ est la longueur de contrainte du code.

A chaque instant *i*, le bit d'information d'entrée d^i est fourni au registre, et aussi directement transmis comme une sortie, le code est alors dit *systématique*. Un autre bit d'information en sortie, le bit de redondance y^i, est le résultat de la somme *modulo* 2 (qui est implémenté par des portes logiques *OR exclusif* - XOR) de la donnée entrée à l'instant *i*, d^i avec des données retardées à l'instant $i-1$ et $i-3$ (d^{i-1} et d^{i-3}). Le symbole codé est distinctement composé du bit systématique et du bit de redondance sous la forme $d^i y^i$. Un multiplexeur joue le rôle d'un convertisseur parallèle-série, qui rend le débit des données sorties deux fois supérieure à celui des données entrées.

Le rendement d'un code *R* est défini par le rapport du nombre de bits d'entrée sur le nombre de bits de sortie. Lorsque le symbole d'entrée est construit par *k* bits, le codeur génère au total *n* bits, donc $R = k/n$, dans l'exemple $R = 1/2$.

Le contenu du registre est défini comme l'état s^i du codeur à l'instant *i*. Il est représenté par la forme d'un vecteur $s^i = s_1^i s_2^i ... s_\nu^i$, où $s_j^i = d^{i-j}$ est le contenu de la mémoire *j* à l'instant *i*. Le codeur possède 2^ν values possibles pour les états.

Le code convolutif est dit *récursif* dès qu'il existe une boucle de retour au sein du registre à décalage. Pour le code non récursif, le contenu du premier point de mémorisation dépend seulement des bits d'entrée. Quant au code récursif, le bit du premier point de mémorisation est non

seulement influencé par les bits d'entrée mais aussi par les bits dans les mémoires. Cette caractéristique mène à une implémentation d'un LFSR (Linear Feedback Shift Register). Berrou a montré que les codes convolutifs systématiques récursifs utilisés dans un schéma de turbocode [1] étaient des codes efficaces pour atteindre la limite de Shannon. Cette technique a été adopté, par exemple, par les normes DVB-RCT, DVB-RCS [40].

La figure 1.7 présente le code convolutif systématique récursif (RSC) qui sera utilisé dans ce manuscrit.

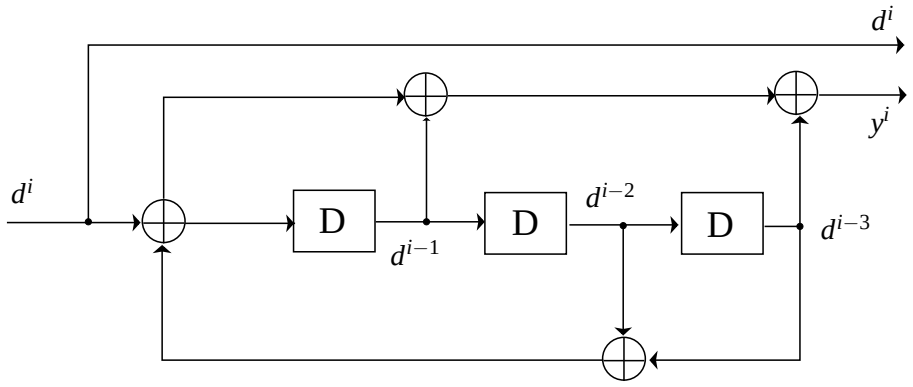


FIGURE 1.7 – Code convolutif systématique récursif

Un code convolutif peut être représenté par plusieurs diagrammes graphiques : diagramme de machine d'états finits, diagramme polynomial, diagramme en arbre... Une des représentations notamment utilisée est la représentation en treillis, qui a été introduite par Forney en 1973 [41]. Ce diagramme est important au niveau de la description des propriétés du codeur convolutif et aussi du décodage de ce code. Le diagramme en treillis pour le code convolutif présenté dans la figure 1.7 est donné dans la figure 1.8.

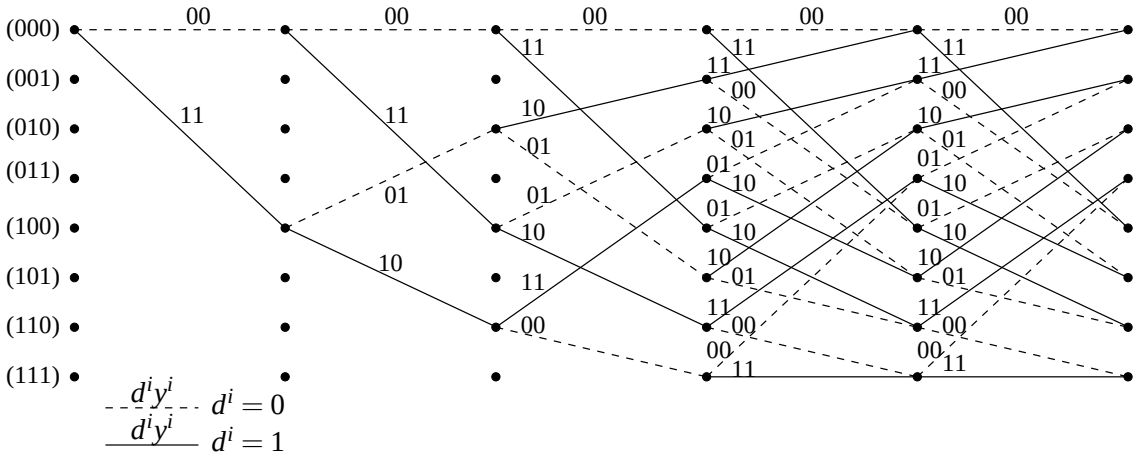


FIGURE 1.8 – Diagramme en treillis associé au codeur convolutif en figure 1.7

Le diagramme en treillis est désigné par deux axes horizontal et vertical. L'axe horizontal présente le temps discret, où les symboles entrent dans le codeur à différents instants. L'axe vertical décrit tous les états successifs possibles (2^v) du codeur. Chaque état du codeur est modélisé

par un *noeud*. A chaque instant i, 2^k transitions possibles partent de chacun de ces états et sont connectés aux états suivants, en fonction de l'entrée d^i. De l'état initial, les connexions entre les états qui sont représentées par les branches sont orientées de gauche à droite. Le diagramme en treillis est construit par la connexion de toutes les branches d'un état aux autres états, en fonction de l'évolution des symboles d'entrée.

Pour une séquence d'information d'entrée au codeur qui est à un état initial fixé (état "tout à zéro" par exemple), l'évolution temporelle du codage peut être vu sur le diagramme en treillis par le passage des branches successives des états connectés, nommé *trajet*. Chaque branche porte l'étiquette $d^i y^j$ qui correspond au symbole codé. Lors de la réception, le décodeur se base sur les observations de la séquence codée et cherche à retrouver le *trajet* emprunté par le codeur pour aboutir à la séquence d'information.

Afin de décoder correctement une séquence d'information, le décodeur doit connaître tous les états et aussi les branches associées à chacun des états du codeur. Malgré que le codeur soit traditionnellement initialisé à l'état "tout à zéro" et connaissant la séquence d'information à longueur finie, le décodeur ne connaît pas l'état final du processus de codage. Ce manque d'information entraîne une dégradation en termes de performance pour la dernière donnée codée. Cela réduit la capacité de protection des données. Ainsi, les derniers symboles codés sont associé à des mots codés à faible poids de Hamming, provoquant une réduction de la performance asymptotique. Ce problème s'accentue quand la longueur de la séquence d'information décroît.

Une des solutions proposées est de fermer le treillis en forçant le codeur pour qu'il atteigne à un état final connu (état "tout à zéro" dans la plupart des cas) en ajoutant à la fin de la séquence d'information des symboles supplémentaires. Cependant, cette technique est plus difficile pour le codeur *récursif* qui correspond à un générateur (pseudo-)aléatoire avec une ligne de rétroaction, comme le notre cas (figure 1.7). Dépendant de la séquence d'entrée, le codeur récursif délivre un état final (pseudo-) aléatoire. Dès lors, il n'est pas facile de déterminer quand l'état final connu est atteint. Pour éviter ce problème d'incertitude de l'état final, une technique a été proposée et adoptée pour le standard 3GPP-LTE [25]. Le code utilisé dans 3GPP-LTE est un code convolutif systématique récursif binaire à 8 états. Pour forcer le codeur à revenir à l'état "tout à zéro", après avoir fini le codage de la séquence d'information, le codeur utilise trois symboles consécutifs "0" qui sont appliqués à l'entrée du codeur. Il correspond au modulo du signal de rétroaction lui-même.

Une autre technique tout aussi efficace est de rendre le code circulaire. Dans ce cas, l'état initial du codeur est identique à l'état final, noté l'état *circulaire*. Le treillis peut être considéré comme un cercle sans discontinuité composé des transitions entre les états à des instants différents. Cette technique de fermeture de treillis est particulièrement avantageuse : le codeur n'a pas besoin d'ajouter des bits supplémentaires. Côté décodeur, le traitement n'exige pas les connaissances de l'état *circulaire* [42]. Ces avantages permettent une égale protection pour tous les symboles émis. La technique de fermeture circulaire est utilisée par exemple dans les standards DVB-RCT [26], et DVB-RCS [27] où les codes convolutifs récursifs systématiques double-binaires à 8 états sont adoptés. La figure 1.9 illustre l'état circulaire du code présenté en figure 1.7.

Dès lors, comment déterminé l'état *circulaire* ? De même, comment s'assurer que le codeur commence le codage à l'état *circulaire* ? La section suivante apporte une réponse à ces deux

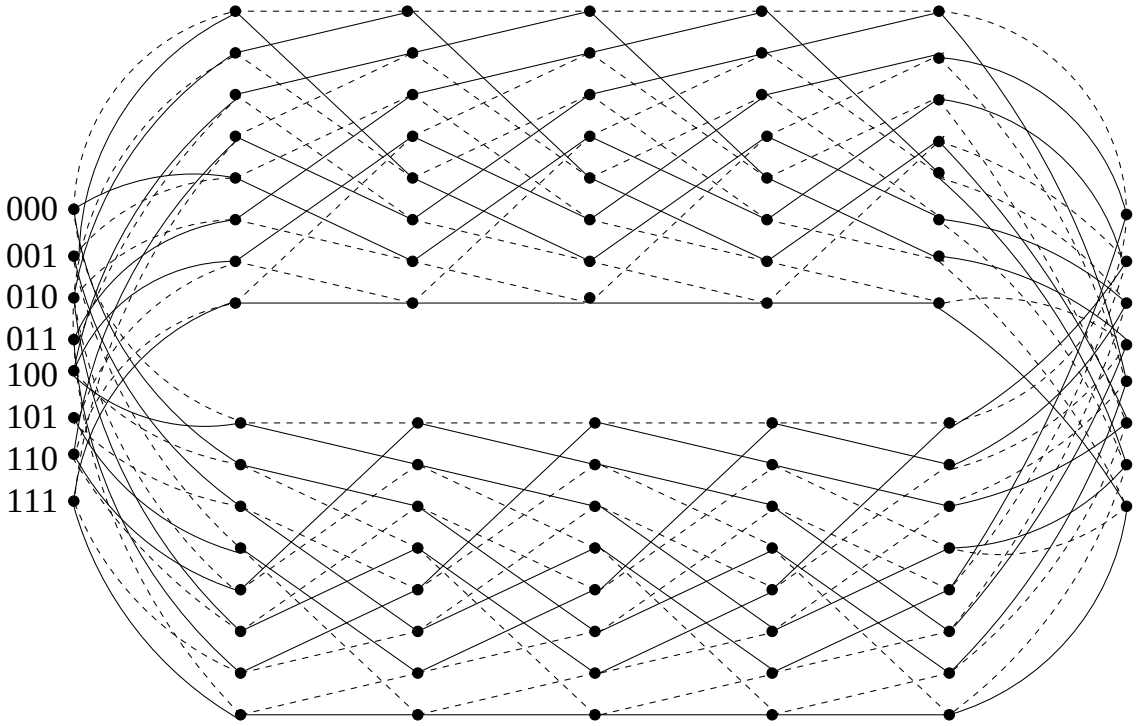


FIGURE 1.9 – Diagramme en treillis circulaire associé au codeur convolutif en figure 1.7.

questions.

1.2.3.4 Codes convolutifs systématiques récursifs circulaires

Le technique d'exploitation du treillis a été présenté en 1986 par le pionnier Ma *et al.* dans [42] pour les codes convolutifs non-systématiques génériques. Elle a été étendue pour les codes récursives par Berrou *et al.* [1] [43]. L'état *circulaire* dépend des caractéristiques du codeur et de la séquence d'entrée [44].

Cette relation peut être exprimée de manière récursive sous la forme de matrice entre l'état $\mathbf{s}^{i+1}$ à l'instant $i+1$ et celui $\mathbf{s}^{i}$ à l'instant i :

$$\mathbf{s}^{i+1} = \mathbf{A}\,\mathbf{s}^{i} + \mathbf{B}\,\mathbf{d}^{i} \tag{1.3}$$

où $\mathbf{A}$ est la matrice des états du codeur et $\mathbf{B}$ est celle d'entrée. Dans le cas du codeur retenu dans la figure 1.7, les matrices $\mathbf{A}$ et $\mathbf{B}$ sont définies comme suit :

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}; \quad \mathbf{B} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{1.4}$$

Lorsque le codeur est initialisé à l'état "tout à zéro" ($\mathbf{s}^0 = 000$), l'état final $\mathbf{s}_0^k$ du processus de codage d'une trame à longueur k est obtenu de la manière suivante :

$$\mathbf{s}_0^k = \sum_{j=1}^{k} \mathbf{A}^{j-1}\mathbf{B}\mathbf{d}^{k-j} \tag{1.5}$$

Par ailleurs, si le codeur est initialisé à l'état quelconque $\mathbf{s}^c$, l'équation 1.3 peut être reformulée :

$$\mathbf{s}^k = \mathbf{A}^k \mathbf{s}^c + \sum_{j=1}^{k} \mathbf{A}^{j-1} \mathbf{B}\, \mathbf{d}^{k-j} \tag{1.6}$$

L'état final $\mathbf{s}^k$ soit identique à l'état initial $\mathbf{s}^c$ et devienne l'état *circulaire* du codeur, si et seulement si :

$$\left(\mathrm{I} - \mathbf{A}^k\right) \mathbf{s}^c = \sum_{j=1}^{k} \mathbf{A}^{j-1} \mathbf{B}\, \mathbf{d}^{k-j} \tag{1.7}$$

Dans cette expression, $\mathbf{I}$ est la matrice identité de dimension $\nu \times \nu$. A partir de cette équation, l'état circulaire est obtenu si et seulement si la matrice $\left(\mathbf{I} - \mathbf{A}^k\right)$ est inversible. En ajoutant l'état final $\mathbf{s}_0^k$ après avoir initialisé le codeur à l'état "tout à zéro", l'état *circulaire* a l'impression suivante :

$$\mathbf{s}^c = \left(\mathbf{I} - \mathbf{A}^k\right)^{-1} \mathbf{s}_0^k \tag{1.8}$$

Toutefois, ce type de code est récursif si $\left(\mathbf{I} - \mathbf{A}^k\right)$ est inversible. Ce codeur possède la propriété que $\mathbf{A}^7$ soit égale à $\mathbf{I}$. Alors, si la longueur de la trame est multiple de 7 ($= 2^\nu - 1$), $\left(\mathbf{I} - \mathbf{A}^k\right)$ devient une matrice nulle - non inversible. Pour les autres longueurs, la matrice $\left(\mathbf{I} - \mathbf{A}^k\right)$ est non nulle, et sa matrice inverse existe. La table suivante résume les états *circulaires* du code convolutif de notre cas. Ils sont calculés en fonction de la longueur de la trame d'entrée et de l'état $\mathbf{s}_0^k$ correspondant. Ces états sont exprimés dans la base de 10.

k mod 7 / $\mathbf{s}_0^k$	1	2	3	4	5	6
0	0	0	0	0	0	0
1	6	4	3	2	5	7
2	3	5	4	6	7	1
3	5	1	7	4	2	6
4	7	2	1	5	6	3
5	1	6	2	7	3	4
6	4	7	5	3	1	2
7	2	3	6	1	4	5

TABLE 1.3 – Tableau des états circulaires du code convolutif présenté en figure 1.7.

L'équation 1.8 permet de coder de la manière circulaire une séquence d'information en trois étapes :

Étape I : Effectuer les calculs matriciels pour établir la table des états circulaires $\mathbf{s}^c$ (Tab.1.3).

Étape II : Réaliser le codage de la séquence d'information en initialisant le codeur à l'état "tout à zéro". Puis, trouver l'état *circulaire* en utilisant le tableau 1.3.

Étape III : Recoder la séquence d'information avec l'état initial égal à l'état *circulaire*.

La technique proposée ci-dessus implique d'appliquer deux processus de codage à une même séquence d'information. En 2000, Pietrobon [45] a expérimenté cette technique, et a suggéré une autre solution qui réduit le délai du codage en ajoutant un entrelaceur derrière le codeur. Cette solution correspond aux trois étapes suivantes :

Étape I : Construire le tableau des états circulaires $\mathbf{s}^c$ (Tab.1.3) telle que la solution proposée.

Étape II : Initialiser l'état du codeur à "tout à zéro" et réaliser le premier codage de la séquence. Ensuite, sauvegarder la séquence codée.

Étape III : Entrelacer la séquence codée. Produire la séquence codée correspondant à l'état circulaire en additionnant à la séquence entrelacée la valeur de l'état circulaire $\mathbf{s}^c$ et de l'architecture du codeur.

1.2.3.5 Algorithme de décodage à entrées et sorties pondérées

En littérature, plusieurs algorithmes sont proposés pour décoder les codes convolutifs. Le premier algorithme a été inventé par Wozencraft [46] et ensuite amélioré par Fano [47] en 1963 avec les entrées et sorties binaires. Quelques années plus tard, Viterbi a introduit un algorithme célèbre [48] qui s'appuie sur la représentation en treillis qui permet de minimiser la probabilité d'erreurs des séquences en acceptant des entrées souples. Toutefois, cet algorithme utilise une *sortie ferme*. Il n'est donc pas utilisable pour un décodage itératif. L'apparition de la concaténation des codes a nécessité l'adaptation de l'algorithme de Viterbi pour accepter des entrées et sorties pondérées (SISO) telles que l'algorithme SOVA [49] et l'algorithme SOVA modifié [50]. Parmi les algorithmes SISO, il est indispensable de rappeler l'algorithme BCJR présenté en 1974 [17] qui est inventé par Bahl, Cock, Jelinek et Raviv. Cet algorithme est aussi appelé algorithme *Maximum A Posteriori - MAP* ou algorithme *Aller - Retour*. Il favorise la maximisation de la probabilité *A Posteriori* de chaque symbole émis par l'évaluation des probabilités de tous les chemins possibles entre l'état initial et l'état final du trellis et ainsi minimise la probabilité d'erreur du symbole émis. Une version étendue de cet algorithme pour décoder les codes circulaires est proposée par Anderson et Hladik [51]. Cet algorithme est la version sous-optimale de l'algorithme MAP, et dit l'algorithme APP.

Algorithme Maximum A Posteriori :

A chaque instant *i*, l'algorithme MAP fournit des probabilités *A Posteriori* - APP $\Pr(d_i|\mathbf{u},\mathbf{v}_1)$ correspondant à chaque symbole émis $d^i \in (0...(2^m-1))$. Le décodeur prend la décision dure $\hat{d^i}$ qui maximise ces probabilités APP. Pour chaque valeur $j \in (0..(2^m-1))$, ces probabilités peuvent s'exprimer en fonction des vraisemblances conjointes $\Pr(d_i = j,\mathbf{u},\mathbf{v}_1)$:

$$\Pr\left(d^i = j|\mathbf{u},\mathbf{v}_1\right) = \frac{\Pr\left(d^i = j,\mathbf{u},\mathbf{v}_1\right)}{\sum_{k=0}^{2^m-1} \Pr\left(d^i = k,\mathbf{u},\mathbf{v}_1\right)} \tag{1.9}$$

En pratique, on calcule les vraisemblances conjointes $\Pr(d^i = j, \mathbf{u}, \mathbf{v}_1)$ pour $j = 0..2^m - 1$, et ensuite, chaque APP est obtenue par normalisation. Lorsque d^i prend seulement les valeurs binaires $j \in (0,1)$, alors :

$$\Pr(d_i = j|\mathbf{u}, \mathbf{v}_1) = \frac{\Pr(d_i = j, \mathbf{u}, \mathbf{v}_1)}{\Pr(\mathbf{u}, \mathbf{v}_1)} = \frac{\Pr(d_i = j, \mathbf{u}, \mathbf{v}_1)}{\Pr(d_i = 1, \mathbf{u}, \mathbf{v}_1) + P(d_i = 0, \mathbf{u}, \mathbf{v}_1)} \tag{1.10}$$

Les probabilités conjointes $\Pr(d_i = j, \mathbf{u}, \mathbf{v}_1)$ sont calculées comme la somme des probabilités conjointes possibles de tous les transitions de treillis de l'état s' à l'état s correspondant au symbole d'entrée $d_i = j$:

$$\Pr(d_i = j, \mathbf{u}, \mathbf{v}_1) = \sum_{(s',s)/d^i(s',s)=j} \Pr\left(s^i = s', s^{i+1} = s, \mathbf{u}, \mathbf{v}_1\right) \tag{1.11}$$

Bahl [17] a montré que la structure en treillis permettait de décomposer les probabilités conjointes $\Pr\left(s^i = s', s^{i+1} = s, \mathbf{u}, \mathbf{v}_1\right)$ en trois probabilités différentes correspondant aux trois métriques, conformément à l'équation 1.12 :

- Métrique récurrente *aller* (forward) $\alpha^i(s)$: la vraisemblance d'un état du treillis à l'instant i selon son passé : $\alpha^i(s') = \Pr\left(s^i = s', \mathbf{u}^{k<i}, \mathbf{v}_1^{k<i}\right)$.
- Métrique récurrente *retour* (backward) $\beta^{i+1}(s)$: la vraisemblance d'un état du treillis à l'instant $i+1$ selon son futur : $\beta^{i+1}(s) = \Pr\left(\mathbf{u}^{k>i}, \mathbf{v}_1^{k>i}|s^{i+1} = s\right)$.
- Métrique $\gamma^i(s',s)$: la vraisemblance d'une branche entre deux états du treillis : $\gamma^i(s',s) = \Pr\left(s^{i+1} = s, \mathbf{u}^i, \mathbf{v}_1^i|s^i = s'\right)$.

$$\Pr\left(s^i = s', s^{i+1} = s, \mathbf{u}, \mathbf{v}_1\right) = \alpha^i(s')\gamma^i(s',s)\beta^{i+1}(s) \tag{1.12}$$

A partir des équations (1.9) et (1.12), les probabilités *a posteriori* APP s'expriment :

$$\Pr\left(d^i = j|\mathbf{u}, \mathbf{v}_1\right) = \frac{\sum_{(s',s)/d^i(s',s)=j} \alpha^i(s)\gamma^i(s',s)\,\beta^{i+1}(s)}{\sum_{(s',s)} \alpha^i(s)\gamma^i(s',s)\,\beta^{i+1}(s)} \tag{1.13}$$

Les métriques récurrentes aller et retour en Eq.1.12 sont récursivement calculées :

$$\begin{aligned} \alpha^i(s) &= \sum_{s'=0}^{2^\nu-1} \alpha^{i-1}(s')\gamma^{i-1}(s',s) \quad \text{pour} \quad i \in (1..k) \\ \beta^i(s) &= \sum_{s'=0}^{2^\nu-1} \beta^{i+1}(s')\gamma^i(s,s') \qquad \text{pour} \quad i \in (k-1..0) \end{aligned} \tag{1.14}$$

Afin de calculer les probabilités récurrentes de l'équation 1.14, une connaissance de l'état de départ et de l'état final du codage est nécessaire. Par exemple, si l'état $\mathbf{S}_0$ de départ du codeur est connu, alors, $\alpha_0(\mathbf{S}_0) = 1$ et $\alpha_0(s) = 0$ pour tout $s \neq 0$. Si l'état est inconnu, les métriques récurrentes $\alpha_0(s)$ sont initialisées à la même probabilité. La même règle est également appliquée pour l'état final $\mathbf{S}_k$ du treillis. De plus, afin d'éviter le problème de débordement, les métriques récurrentes sont régulièrement normalisées.

La métrique de vraisemblance d'une branche de l'équation 1.12 est nulle si aucune transition entre les états s' et s existe. Dans le cas contraire, elle peut s'exprimer comme :

$$\gamma^i\left(s',s\right) = \Pr_a\left(d^i = j\right)\Pr\left(\mathbf{u}^i,\mathbf{v}_1^i|\mathbf{d}^i,\mathbf{y}_1^i\right) \tag{1.15}$$

où $\Pr_a(d^i = j)$ est la probabilité *a priori* correspondant au symbole émis $d^i = j$ qui dépend de la statistique de la source. Dans le cas d'une source équiprobable, tous les symboles possèdent la même probabilité de transmission. Alors, $\Pr_a(d^i = j) = 1/2$ pour une source binaire équiprobable. En outre, dans le cas d'un canal gaussien *Additive White Gaussian Noise - AWGN*, la probabilité $\Pr\left(\mathbf{u}^i,\mathbf{v}_1^i|\mathbf{d}^i,\mathbf{y}^i\right)$ est donnée par :

$$\Pr\left(\mathbf{u}^i,\mathbf{v}_1^i|\mathbf{d}^i,\mathbf{y}_1^i\right) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{\left\|\mathbf{u}^i-\mathbf{d}^i\right\|^2+\left\|\mathbf{v}_1^i-\mathbf{y}^i\right\|^2}{2\sigma^2}\right) \tag{1.16}$$

avec σ^2 la variance du bruit additif blanc gaussien. En pratique, seuls les termes spécifiques correspondant à la transition considérée et qui ne s'éliminent pas par la division dans l'expression 1.10 sont retenus. Par conséquent, l'expression (1.16) est réécrite comme suit :

$$\Pr\left(\mathbf{u}^i,\mathbf{v}_1^i|\mathbf{d}^i,\mathbf{y}_1^i\right) = \exp\left(\frac{\left\langle\mathbf{u}^i,\mathbf{d}^i\right\rangle+\left\langle\mathbf{v}_1^i,\mathbf{y}_1^i\right\rangle}{\sigma^2}\right) \tag{1.17}$$

Dans le cadre du turbo décodage, le turbo-décodeur demande aux deux décodeurs convolutifs d'échanger des informations extrinsèques. C'est pourquoi, l'algorithme MAP doit être modifié pour produire l'information extrinsèque. Pour cela, l'expression (1.15) doit tenir également compte de l'information extrinsèque produite par l'autre décodeur élémentaire :

$$\gamma^i\left(s',s\right) = \Pr_a\left(d^i = j\right)\Pr_e^{in}\left(d^i = j|\mathbf{u},\mathbf{v}_2\right)\Pr\left(\mathbf{u}^i,\mathbf{v}_1^i|\mathbf{d}^i,\mathbf{y}_1^i\right) \tag{1.18}$$

où l'information *a priori* $\Pr_a(d^i = j)$ est remplacée par $\Pr_a\left(d^i = j\right)\Pr_e^{in}\left(d^i = j|\mathbf{u},\mathbf{v}_2\right)$ et $\Pr_e^{in}\left(d^i = j|\mathbf{u},\mathbf{v}_2\right)$ est l'information extrinsèque fournie par le second décodeur composant.

L'information extrinsèque générée par ce décodeur élémentaire est exprimée par :

$$\Pr_e^{out}\left(d^i = j|\mathbf{u},\mathbf{v}_1\right) = \frac{\sum\limits_{(s',s)/d_i(s',s)=j}\alpha^i\left(s'\right)\beta^{i+1}\left(s\right)\gamma_e^i\left(s',s\right)}{\sum\limits_{(s',s)}\alpha^i\left(s'\right)\beta^{i+1}\left(s\right)\gamma_e^i\left(s',s\right)} \tag{1.19}$$

avec $\gamma_e^i(s',s)$ la métrique de branche modifiée qui ne tient plus compte des informations disponibles dans le décodeur auquel l'information extrinsèque est adressée. Dans le cas du turbo décodage des codes convolutifs concaténé en parallèle, cette métrique est obtenue pour un canal AWGN à partir de l'expression (1.17) en éliminant l'information systématique :

$$\gamma_e^i(s',s) = \exp\left(\frac{\left\langle\mathbf{v}_1^i,\mathbf{y}_1^i\right\rangle}{\sigma^2}\right) \tag{1.20}$$

Malheureusement, cet algorithme a une grande complexité d'implémentation matérielle, car il nécessite des opérations multiplication et division. C'est pourquoi, un algorithme sous-optimal de l'algorithme BCJR appelé l'algorithme Max-Log-MAP [52] est plutôt considéré.

1.2.4 La montée en débit, un défi

Dès la présentation des turbocodes, de nombreuses recherches ont tenté de mettre en œuvre une architecture matérielle qui réponde aux besoins des systèmes de communication modernes : augmentation du débit de transmission, réduction de la latence, maîtrise de la complexité matérielle et de la consommation. Dans notre contexte d'implémentation matérielle d'un turbo-décodeur, nous nous focalisons principalement sur les performances de décodage pour un débit maximum.

Le débit d'un turbo-décodeur est défini comme le nombre de bits moyen décodé par seconde. Il est déterminé par plusieurs paramètres d'implémentation. Par exemple, le délai moyen du traitement de la trame, le nombre moyen d'itérations de décodage et la longueur de la trame.

Ainsi, l'expression du débit d'une implémentation d'un turbo-décodeur est donné en [53], et s'exprime par :

$$D_s = f_{clk}.\alpha(\Phi).m.\frac{\Phi}{2.I_{\max}.\chi_u} \quad \text{[bits/s]} \tag{1.21}$$

où f_{clk} est la fréquence d'horloge, m est le nombre de bits par symbole, Φ est le nombre de ressources de calcul, I_{max} est le nombre d'itération de décodage maximum, χ_u est le nombre d'opérations élémentaires requis pour produire un symbole m-binaire décodé et $\alpha(\Phi)$ est défini comme l'activité du turbo-décodeur. L'activité correspond à l'utilisation moyenne des ressources de calcul pendant l'exécution de l'algorithme et dépend de l'algorithme de décodage et de l'architecture matérielle mise en œuvre.

Différentes investigations sont envisageables pour favoriser l'augmentation du débit comme :

- **Réduire le nombre d'itérations** I_{max} : L'utilisation d'un critère d'arrêt permet au turbo-décodeur de s'arrêter lors du processus itératif dès que le critère est satisfait. Différents critères peuvent être considérés : l'analyse de l'entropie croisée [54–58] ; l'utilisation d'un diagramme EXIT [59, 60] et d'un diagramme d'évolution de densité [61] ; la considération de l'algorithme de décodage itératif comme un système dynamique non-linéaire à temps discret [60]. Cependant, cette approche est très peu mise en œuvre en pratique.
- **Multiplier le nombre des ressources matérielles** Φ : L'augmentation des ressources de calcul est couramment employée pour augmenter le débit d'une architecture. L'objectif de cette méthode est de multiplier linéairement les ressources de calcul d'un décodeur avec l'augmentation de débit. Néanmoins, cette technique doit tenir compte du compromis entre le débit du décodeur avec la complexité matérielle qui est naturellement limitée du fait du circuit.
- **Améliorer l'activité de l'architecture caractérisée par** $\alpha(\Phi)$: Une augmentation de l'activité influence efficacement le débit de l'architecture. L'activité d'une architecture d'un turbo-décodeur dépend du nombre de ressources Φ et de l'algorithme utilisé.
- **Maximisation de la fréquence d'horloge** : L'augmentation de la fréquence d'horloge maximale entraîne une augmentation proportionnelle du débit de l'architecture correspondante. Cependant, dépendant du type de circuit (FPGA, ASIC,...), le délai de propagation à

travers le chemin critique est généralement imposé par les opérateurs élémentaires configurés, en résulte une limitation de la fréquence maximale d'horloge.

Dans la thèse de D. Gnaedig [53], les atouts et les limitations de différents approches sont exposés. Il montre dans le cadre d'un turbo décodage à grand débit que l'utilisation du parallélisme des ressources de calcul favorise l'augmentation de l'activité du décodeur et du débit. Lorsque le nombre de ressource est multiplié par un facteur ρ, la complexité matérielle est élevée de ρ fois. Le débit de l'architecture correspondant est amplifié de ρ fois. Une classification de différentes techniques de parallélisme pour les turbo-décodeurs ainsi que les avantages et les inconvénients sont donnés dans la thèse de d'O. Muller [62]. D'une manière générale, lorsque les turbo-décodeurs sont plus parallélisés, ils convergent mieux. Ce constat a également été observé lors de l'implémentation du turbo-décodeur analogique [63].

Récemment, plusieurs implémentations utilisant le parallélisme du turbo décodage avec différents paramètres sont largement détaillées. Des implémentations réussissent à obtenir un grand débit de décodage pour les standards cibles. Un paramètre qui évalue l'efficacité d'implémentation est introduit - l'efficacité architecturale. Il est défini comme le rapport de la rapidité d'exécution sur la complexité surfacique en assurant des performances équivalentes en termes de taux d'erreur : $E_c = \frac{D_s}{C}$ (Mbps/mm^2). Où D_s le débit du turbo-décodeur et C sa complexité surfacique. Le tableau 1.4 ci-dessous récapitule différentes implémentations sur ASIC d'architectures de turbo-décodeurs qui ont obtenu de grands débits de décodage. Le tableau 1.4 présente l'efficacité archi-

	[14]	[12]	[11]	[64]	[65]	[66]
Standard	-	-	LTE	-	LTE/WiMAX	LTE
Technologie	90 nm	65 nm	130 nm	130 nm	130 nm	90 nm
Taille de bloc	4096	6144	6144	4096	4096	6144
Max. Parallélisme	32	32	8	16	8	8
Max. Itération	8	6	5.5	8	8	8
Fréquence(Mhz)	175	200	320	80	250	275
Débit(Mbps)	1400	711	391	160	186	130
Débit norm.(Mbps)	1400	533,3	268,8	160	186	130
Surface(mm^2)	9,61	N/A	3,57	17,81	10,7	2,1
Surface norm.(mm^2)	9,61	N/A	1,71	8,55	5,14	2,1
E_c(Mbps/mm^2)	145,68	N/A	157,2	18,7	36,2	61,9

TABLE 1.4 – Comparaison de différents implémentations de turbo-décodeurs parallèles.

tecturale de différents turbo-décodeurs. Toutefois, une comparaison directe des implémentations est difficile car différents paramètres interviennent : la technologie, la fréquence, la précision arithmétique,... Alors, pour faciliter la comparaison, la surface de chaque décodeur est normalisée par une mise à l'échelle à la technologie 90nm avec les facteurs d'échelle de 2 et 0,48 respectivement pour les technologies de 65 nm et 130 nm. Les débits équivalents sont linéairement mis à l'échelle pour un maximum d'itération 5,5. Cependant, la méthode de conversion universelle de la fréquence d'horloge entre les différentes technologies n'est pas disponible. Les résultats

démontrent qu'en utilisant le parallélisme, les turbo-décodeurs 3GPP-LTE proposés atteignent un meilleur débit par rapport au standard courant, et il permet aussi d'inspirer de nouvelles architectures de turbo-décodeurs parallèles à grand débit, faible complexité. Pourtant, en littérature, avec nos connaissances, très peu d'architectures de turbo-décodeur parallèle peuvent obtenir simultanément un grand débit (Gbps) mais avec faible coût. Par conséquent, d'autres méthodes doivent être considérées.

Parmi les solutions apparues, l'approche du décodage stochastique s'avère prometteuse pour réaliser une architecture à grand débit avec une complexité de circuit faible. Le potentiel de cette méthode de décodage stochastique pour complexité faible et grand débit a été récemment démontré par l'implémentation des décodeurs LDPC sur ASIC en 2010 - l'implémentation d'un décodeur LDPC (2048,1723) sur la technologie 90nm à 500Mhz atteint un débit de 61,3Gbps [4]. En comparaison avec les autres travaux, le décodeur stochastique LDPC offre une efficacité architecturale compétitive. Son efficacité architecturale est seulement inférieure de 4% à celle de l'architecture la plus efficace présentée en même année [67]. Le résultat obtenu prouve que l'approche stochastique permet de construire un décodeur entièrement parallèle qui est capable de fournir un grand débit et une complexité raisonnable pour le décodage itératif de codes correcteurs d'erreurs puissants.

Exploitant ces deux idées : l'approche stochastique et le décodage en parallèle, logiquement, est-il possible de réaliser un décodeur stochastique parallèle des turbocodes qui peut atteindre un haut débit et une faible complexité ? Les deux sections suivantes vont tenter de répondre à cette question en introduisant pas à pas l'approche stochastique de décodage des codes LDPC et les premières briques de construction d'un turbo-décodeur stochastique.

1.3 Principe du traitement stochastique pour le décodage de codes LDPC

La technique de calcul stochastique est d'abord présentée dans les années 1960 par Gaines [5]. Elle a été appliquée avec succès aux différentes applications comme l'implémentation des réseaux de neurones [68] et des systèmes de contrôle de moteur [69]. Elle a récemment été utilisée pour décoder des codes utilisant des graphes. Le décodage stochastique fût d'abord considéré pour des codes de petite taille : les codes (7,4) Hamming en 2003 par Gaudet *et al.* [70, 71]. La première implémentation du décodage stochastique des codes (16,8) LDPC a été décrite par Warren Gross en 2005 [72]. Plus tard, un algorithme de décodage stochastique basé sur la représentation de treillis a été appliqué pour décoder des turbocodes en bloc (256,121) [73]. Une approche de décodage plus efficace a été employée pour des codes LDPC par Sharifi *et al.* en 2006 [74]. Puis, en 2007, l'implémentation sur FPGA d'un décodeur LDPC(1056,528) produit un débit maximum de 1,66 Gbps. Il s'agit du premier décodeur stochastique atteignant un débit supérieur au Gbps. Récemment, en 2010, un débit notable de 61,3Gbps a été obtenu par une implémentation d'un décodeur LDPC(2048,1723) sur cible ASIC [4]. En comparaison avec l'implémentation conventionnelle en virgule fixe, le décodage stochastique peut produire une performance optimale pour des codes LDPC de petite taille, et une performance quasiment optimale pour des codes LDPC

similaires à celles des standards. Il est à noter que sa complexité d'architecture s'approche à celle de l'architecture la plus efficace d'aujourd'hui.

Dans le calcul stochastique, les informations s'expriment par des séquences de bits - les séquences de Bernoulli. La probabilité d'occurrence de bits à "1" dans la séquence est égale à la probabilité que le message transmet. Ainsi, des opérations arithmétiques classiques comme multiplication, division et addition sont remplacées par des opérations logiques utilisant des portes logiques simples. Dès lors, le décodage stochastique se déroule par l'échange des informations binaires en série, au lieu de propager des messages probabilistes entre les noeuds de graphe.

La suite de la section détaille le décodage stochastique des codes LPDC par l'introduction des concepts de base du traitement stochastique, ainsi que les blocs logiques utilisés pour les opérations arithmétiques. Ensuite, on va aborder les attributs non-idéaux des opérations stochastiques pour le décodage des codes LDPC. Finalement, la section traite les solutions proposées pour optimiser un décodage stochastique de code LDPC.

1.3.1 Principe du calcul stochastique

1.3.1.1 Représentation stochastique de l'information

Le calcul stochastique est une technique qui associe à chaque probabilité une séquence discrète aléatoire des valeurs numériques $\{a_i\}$ où $i = 1...L$ avec L la longueur de la séquence. a_i prend les valeurs binaires $(0,1)$. Ces bits constituent une séquence de Bernoulli, dans laquelle la probabilité représentée est contenue dans la statistique du flux stochastique $\{a_i\}$. Lors de sa génération, la probabilité du symbole "1" est équivalente à la probabilité que ce flux représente. Un

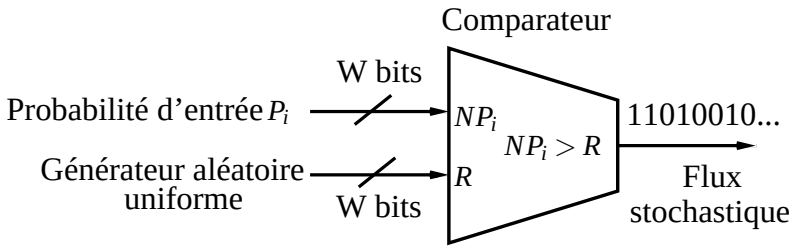


FIGURE 1.10 – Génération d'un flux stochastique représentant une probabilité P_i en utilisant un comparateur et un générateur (pseudo-)aléatoire à distribution uniforme.

comparateur peut être utilisé afin de convertir une probabilité en flux stochastique [5, 75] comme illustré dans la figure 1.10. Dans ce cas, la probabilité P_i est mise à l'échelle à W bits de largeur sur un bus de NP_i. Le résultat est comparé avec un R de la même largeur W qui contient des valeurs aléatoires variant dans le temps selon une distribution uniforme. La sortie du comparateur est égale à "1" si NP_i est supérieure à R ($NP_i > R$). Elle est égale à "0" dans le cas contraire. La probabilité de production d'un bit "1" dans le flux généré est équivalente à $\frac{NP_i}{2^W}$ et le nombre total de bit "1" présent dans le flux désigne la probabilité P_i. Par exemple, si une séquence binaire de longueur 20 bits contient 4 bits à "1", alors la probabilité correspondante est 0,2. De même, si 5 bits sont à "1" dans une séquence de longueur 10 bits, la séquence porte un message de probabi-

lité 0,5. Cependant, de nombreuses représentations peuvent être considérées pour coder une même probabilité, comme l'exemple montré en figure 1.11. Cela signifie que les positions d'occurrence du bit "1" dans la séquence ne sont pas importantes. En utilisant la représentation stochastique,

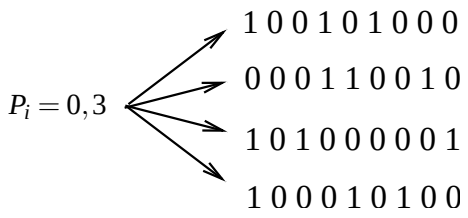


FIGURE 1.11 – Quelques exemples de représentation d'une probabilité $P_i = 0,3$.

la probabilité est convertie en une série de bits, c'est pourquoi, un de ses avantages est de ne nécessiter qu'une seule équipotentielle pour représenter le signal de communication entre les éléments de calcul. Dès lors, les opérations arithmétiques sur les probabilités sont remplacées par des opérations logiques des plus simples sur des séries de bits. Cet avantage permet d'atteindre une fréquence d'horloge de fonctionnement élevée. De plus, lors d'une présence de bruit à un moment, un seul bit stochastique dans le flux est affecté au lieu de l'intégralité du message interprété. Alors, une représentation par des flux stochastiques est robuste lors de la présence de bruit binaire [68].

Une des contraintes de l'approche stochastique est qu'en présence d'une probabilité faible, une large précision est nécessaire. Par exemple, seuls 10 bits sont vraisemblablement suffisants au domaine des virgules fixes pour exprimer une probabilité de 0,001. Par contre, au moins 1000 bits stochastiques sont nécessaires pour représenter cette probabilité. Une meilleure précision de calcul stochastique est obtenue en observant plus longtemps la séquence des bits. Par ailleurs, les traitements stochastiques reposent sur les changements dans la statistique des flux plutôt que sur la précision des séquences stochastiques. Ceci entraîne que la précision des calculs stochastiques ne peut pas être aussi importante.

1.3.1.2 Opérations arithmétiques stochastiques

L'approche stochastique permet de transformer des opérations arithmétiques complexes sur les probabilités comme la multiplication ou la division par des opérations logiques simples utilisant des portes logiques élémentaires, des bascules JK ou des multiplexeurs. En définissant que les flux des bits stochastiques $\{a_i\}$ et $\{b_i\}$ représentent les probabilités d'entrée $P_a = \Pr(a_i = 1)$ et $P_b = \Pr(b_i = 1)$, le flux $\{c_i\}$ représente la probabilité de sortie de $P_c = \Pr(c_i = 1)$ qui est le résultat des opérations stochastiques sur les flux des probabilités P_a et P_b.

La multiplication :

La multiplication de deux séquences de Bernoulli est effectuée par une porte logique AND. Etant donnée la table de vérité de l'opération AND, le bit de sortie est égal à "1" si les deux entrées sont à "1". Le résultat $P_c = \Pr(C = 1) = \Pr(A) * \Pr(B)$ est illustré par la figure 1.12. La probabilité représentée par la fraction des bits à "1" sur la sortie de la porte est égale au produit

de deux probabilités représentées par les deux flux stochastiques sur les entrées de la porte. Le résultat est valide si les deux flux stochastiques en entrée sont totalement indépendants.

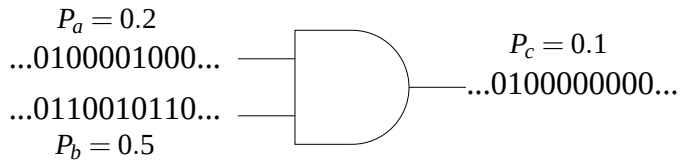


FIGURE 1.12 – Multiplication de deux flux stochastiques indépendants.

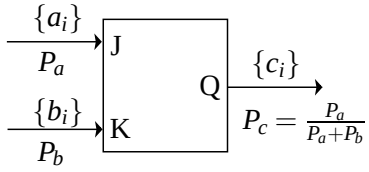
Si l'on veut effectuer une multiplication d'une probabilité avec elle-même $P_c = P_a * P_a = P_a^2$, il faut décorréler les deux séquences d'entrée. Pour ce faire, il est possible de retarder la séquence $\{a_i\}$ d'un coup d'horloge en utilisant une bascule D- Flipflop. [5, 68].

La division :

Naturellement, aucun opérateur permet directement une division arithmétique entre deux valeurs flottantes. C'est également le cas de la division stochastique entre deux séquences de Bernoulli. Cependant, des solutions à partir des structures simples basées sur des approximations acceptables sont proposées par Gaines [5]. Une de ces solutions est de traiter la division entre les deux séquences de Bernoulli $\{a_i\}$ et $\{b_i\}$ si $P_a \ll P_b$. Dans ce cas, le résultat de la division P_c s'exprime par :

$$P_c = \frac{P_a}{P_b} \approx \frac{P_a}{P_a + P_b} \tag{1.22}$$

Ainsi, l'opération division entre les deux flux stochastiques est effectivement équivalente à une opération normalisation entre les flux. Cette opération peut être modélisée par une bascule JK simple qui est illustrée en figure 1.13. Le bit sortant de la bascule est égal au bit entrant J si les deux entrées J et K sont différentes. Si les bits entrant sont égaux à "0", le bit de sortie précédent c_{i-1} est considéré. Enfin, l'inverse du bit de sortie précédent $\overline{c_{i-1}}$ est pris si les deux entrées sont égales à "1". Ces deux cas génèrent alors un effet de retenue, le bit de sortie à l'instant i est corrélé au bit de sortie à l'instant $i-1$.



a_i	b_i	c_i	
0	0	c_{i-1}	corrélation entre
1	1	$\overline{c_{i-1}}$	les bits sortant
0	1	0	$c_i = a_i$
1	0	1	bit regénératif

FIGURE 1.13 – Division stochastique entre deux flux stochastiques.

L'addition :

L'opération addition entre des flux stochastiques est plus complexe à réaliser que les opérations multiplication et division. Cela s'explique par le fait que le résultat de cette opération n'est nécessairement pas dans l'intervalle de probabilité [0 ;1]. Des méthodes de mise à l'échelle des probabilités d'entrée de ces opérations sont nécessaires afin d'assurer que le résultat soit dans l'intervalle [0 ;1]. L'opération addition à partir des coefficients d'échelle est très simple à réaliser

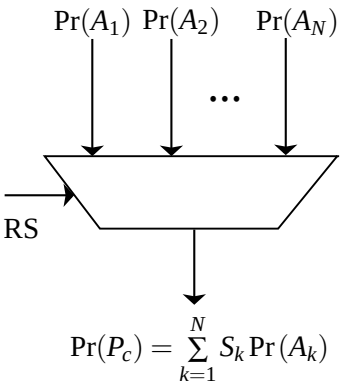


FIGURE 1.14 – Additionneur stochastique est implémenté par un multiplexeur.

matériellement en utilisant un multiplexeur (MUX). En considérant que les N flux stochastiques $\{a_i\}_k, k = 1...N$ représentent alors respectivement N probabilités d'entrée $\Pr(A_k)$. Le multiplexeur sélectionne au hasard une entrée k avec une probabilité S_k où $\sum_{k=1}^{N} S_k = 1$ et produit une sortie avec une probabilité équivalente à la somme de mise à l'échelle de tous les probabilités d'entrée $\Pr(P_c) = \sum_{k=1}^{N} S_k \Pr(A_k)$. Le multiplexeur est décrit dans la figure 1.14 avec RS un signal de contrôle qui effectue la sélection aléatoire. En pratique, ce signal doit être piloté par un générateur (pseudo-)aléatoire. Les valeurs de RS sont faciles à déterminer ou à implémenter lorsque N est une puissance de 2. Si N n'est pas une puissance de 2, alors des signaux supplémentaires - des signaux "0" - sont ajoutés afin d'assurer N comme une puissance de 2. Dans ce cas, le résultat obtenu est sous-optimal.

1.3.2 Décodage stochastique de codes LDPC

En profitant des atouts de l'approche stochastique - simplicité du calcul et de la structure matérielle correspondante - le décodage stochastique est applicable à des codes LDPC. Différentes implémentations sur une carte FPGA ont abouti à des débits très élevés : 706 Mbps [76] ; 1,66 Gbps [16]. Le premier décodeur stochastique sur cible ASIC a atteint un débit notable : 61,3 Gbps [4]. De plus, ce décodeur stochastique utilise une très faible complexité de l'architecture. Son efficacité architecturale est seulement plus petite de 4% par rapport à l'architecture la plus efficace en littérature. Ces résultats démontrent que le décodage stochastique est une des approches les plus efficaces pour atteindre une architecture de décodeur des codes correcteurs d'erreurs à haut débit et faible coût. Le reste de cette section présente brièvement la construction d'un décodeur stochastique pour des codes LDPC. Il commence par la construction des codes LDPC, ensuite le principe de décodage stochastique des codes LDPC par une analyse du processus de convergence (le problème de *corrélation*). Et enfin des méthodes de mémorisation sont successivement détaillées.

Les codes LDPC sont des codes en bloc linéaires construits à partir de codes élémentaires très simples : les codes de parité. Les codes LDPC sont caractérisés par une matrice à faible densité, dite matrice de parité (ou matrice de contrôle) $\mathbf{H}$ de taille $m * n$ avec $m = n - k$. Cette matrice contient un faible nombre de valeurs égales à "1". Cette matrice peut être vue comme un système

linéaire de m équations de parité de longueur n. Les mots de code $\mathbf{c}$ définis par $\mathbf{H}$ sont les mots binaires dont les n bits vérifient simultanément les m équations de parité $\mathbf{H}\mathbf{c}^t = \mathbf{0}$. Un exemple de matrice $\mathbf{H}$ est le suivant :

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \tag{1.23}$$

Les codes LDPC peuvent être représentés par un graphe bipartite. Ce type de graphe est composé des deux classes de noeuds différents : les noeuds de variable et les noeuds de parité. Chacun des noeuds de variable correspond à une colonne de $\mathbf{H}$. L'ensemble des noeuds de variable contribue aux bits des mots de code. Chaque noeud de parité est équivalent à une ligne de $\mathbf{H}$, et tous les noeuds de parité correspondent aux équations de parité. Chaque branche (arc) connectant un noeud de variable à un noeud de parité correspond à un "1" dans la matrice de parité.

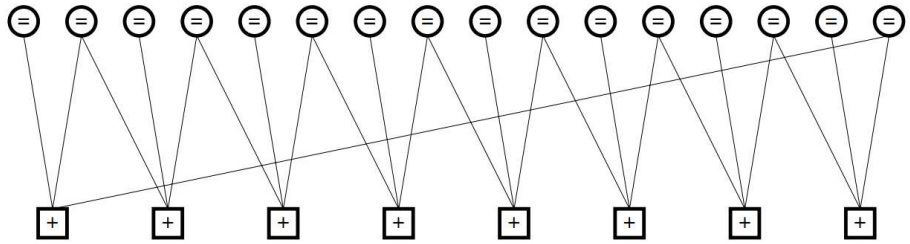


FIGURE 1.15 – Graphe bipartite du code LDPC(16,8) [72] ayant pour matrice $\mathbf{H}$ celle de l'expression 1.23.

Les codes LDPC sont décodés à l'aide d'un algorithme de décodage itératif dit algorithme de la propagation de croyance (*Belief Propagation - BP* en anglais) [77]. La première étape consiste à calculer les vraisemblances des bits reçus. Puis un traitement successif des noeuds de variable et ensuite des noeuds de parité réalisés [77] par des échanges d'information sur les arcs du graphe bipartite est appliqué. Les codes LDPC sont également des codes puissants qui fournissent une performance proche de la limite de Shannon [78].

Pour un code LDPC, le nombre de "1" dans la matrice de parité correspond au nombre d'arcs du graphe de bipartite, et est relativement faible. Cependant, lorsque la taille des matrices utilisées par les codes LDPC devient importante, le nombre de branches du graphe bipartite correspondant augmente notablement [79]. Ce problème rend le processus de décodage complexe et entraîne une grande consommation de surface et des difficultés de routage. Ce problème est un des défis principaux de réalisation des circuits des décodeurs LDPC [80].

Le décodage stochastique des codes LDPC débute par la conversion des vraisemblances des

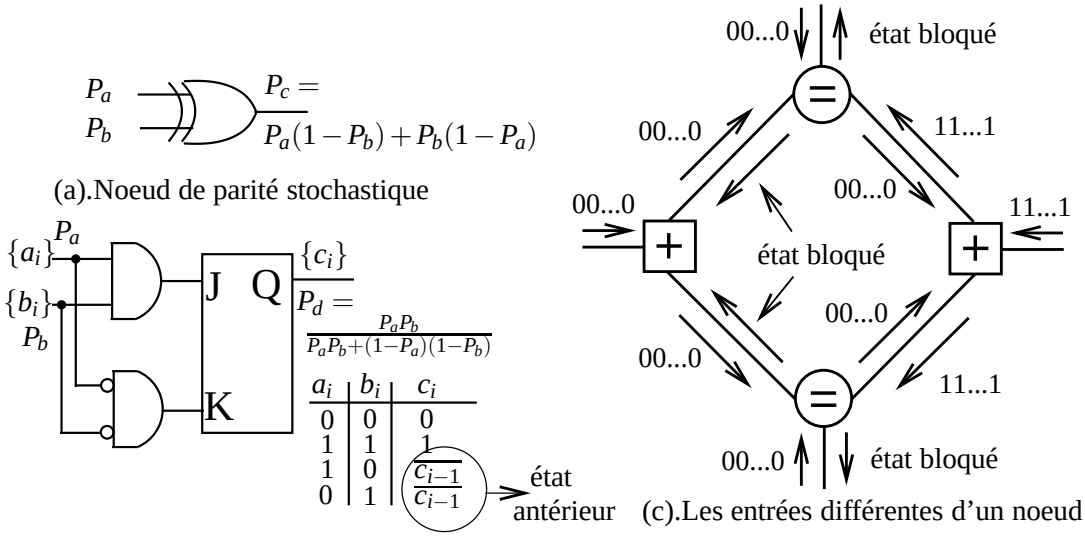


FIGURE 1.16 – Les noeuds stochastiques du graphe de Tanner du codes LDPC [81] et le problème de blocage pour un graphe avec des cycles.

bits reçus en des séquences de Bernoulli. Chaque séquence de bits est ensuite propagée à travers le graphe bipartite. Dès lors un seul fil est nécessaire pour présenter un arc entre un noeud de variable et un noeud de parité. La figure 1.16 présente les architectures stochastiques des noeuds de variable et de parité. Cet avantage simplifie significativement la complexité des noeuds et réduit également le problème de congestion de routage. Ceci implique que durant chaque cycle de décodage, le process de décodage stochastique est effectué par l'échange d'un bit entre un noeud de variable et un noeud de parité sur l'arc de graphe correspondant. Chaque cycle est défini comme un *cycle de décodage* (decoding cycle - DC en anglais) qui est différent d'une itération de décodage dans l'algorithme par propagation de croyance. Le processus de décodage stochastique des codes LDPC s'arrête après un nombre maximum de cycles de décodage, ou lorsque la matrice de contrôle **H** est vérifiée [16].

1.3.2.1 Problème de corrélation

Lors d'un décodage stochastique, les opérations sont effectivement traitées lorsque les bits d'un flux entrant dans un élément de calcul stochastique sont indépendants des bits de l'autre flux entrant. Toutefois certains éléments de calcul stochastique peuvent introduire de la dépendance donc de la corrélation bit par bit au niveau des messages sortant. Dans ce cas, les messages deviennent de plus en plus altérés pour d'autres calculs stochastiques. C'est le cas d'une bascule JK pour laquelle les deux entrées sont identiques (figure 1.13 et 1.16). En effet, le bit de sortie à l'instant i est corrélé au bit de sortie à l'instant $i-1$.

En 2005, C. Winstead a analysé cette problématique lors du décodage stochastique des codes en graphe tels que les codes Hamming [82] mais aussi des codes LDPC [81]. Il l'a appelé *problème de verrouillage* (*latching* en anglais). Ce problème a pour origine l'existence de cycles courts dans le graphe qui introduisent des corrélations dans les messages, qui bloquent le graphe dans des

états. Ces états sont appelés les *états bloqués* (*hold state* en anglais) et sont conservés pendant plusieurs cycles ce qui complexifie la convergence du décodeur. Ce message corrélé est dit *flux conservatif*. Le problème de corrélation s'accentue pour un grand rapport signal-sur-bruit (RSB) auquel la probabilité correspondant reçue à partir du canal est proche de "1" ou de "0". Dans ce cas, les bits dans la séquence Bernoulli équivalente sont égaux à "1" (ou "0") quasiment en permanence.

1.3.2.2 Amélioration du problème de corrélation

Après avoir montré la nature du problème de corrélation, des solutions ont été proposées afin d'améliorer l'activité de transition des bits dans le flux stochastique conservatif. Cette section va présenter brièvement des solutions mises en œuvre dans la littérature afin de diminuer la décorrélation des messages.

Technique *NDS* :

L'idée de mise à l'échelle est d'assurer un même niveau d'activité de commutation pour tous les blocs des LLRs d'entrée du décodeur en utilisant des coefficients appropriés - les coefficients *NDS* (*Noise-Dependant Scaling*). Le premier coefficient proposé fut la *valeur maximum de LLR* pour décoder les codes Hamming(16,11) [82]. Plus tard, Sharifi Tehrani a développé cette technique où les LLR sont mises à l'échelle par des facteurs qui sont proportionnels aux rapports signal sur bruit (RSB), alors permettent un niveau similaire de l'activité de commutation pour les différents RSB. Cette technique a été appliqué en [74, 76, 83, 84].

Technique *Supernode* :

Cette technique a été premièrement introduit par Winstead pour le décodage stochastique du turbocode en bloc (256,121) [73]. Ensuite, elle a été simplifiée par Gross [85] afin de décoder les codes LDPC(16,8). Un Supernode est une structure qui observe le flux stochastique entrant et régénère la même probabilité correspondant de telle sorte que les corrélations et les dépendances soient supprimées. Chaque supernode consiste en un compteur et un générateur de valeurs aléatoires. Le compteur accumule le nombre de bit "1" durant certains cycles dans le message stochastique, et le générateur aléatoire est utilisé afin de régénérer un nouveau message stochastique à partir des bits "1" accumulés. Cette technique permet de casser la corrélation des messages mais elle rend le circuit plus complexe en raison du nombre de générateurs et de compteurs insérés.

Technique des mémoires *EM* :

Les *mémoires d'arc* (*Edge Memories - EMs* en anglais) sont efficaces pour décoder de longs codes en associant à la technique de mise à l'échelle mentionné avant [74]. Chaque EM se compose de registres à décalage de M bits et est introduite pour chaque branche sortant du graphe liée aux noeuds de variable. Cette technique permet de garder une trace des sorties précédentes d'un noeud de variable en favorisant les nouvelles sorties. Ainsi, on stocke sur un arc une probabilité représentative des bits régénératifs du flux de sortie. L'architecture de la technique de mémoire *EM* est détaillée en figure 1.17. Une bascule de type D-Flip-Flop est nécessaire, quand à elle pour

chaque état, pour assurer la synchronisation de l'architecture.

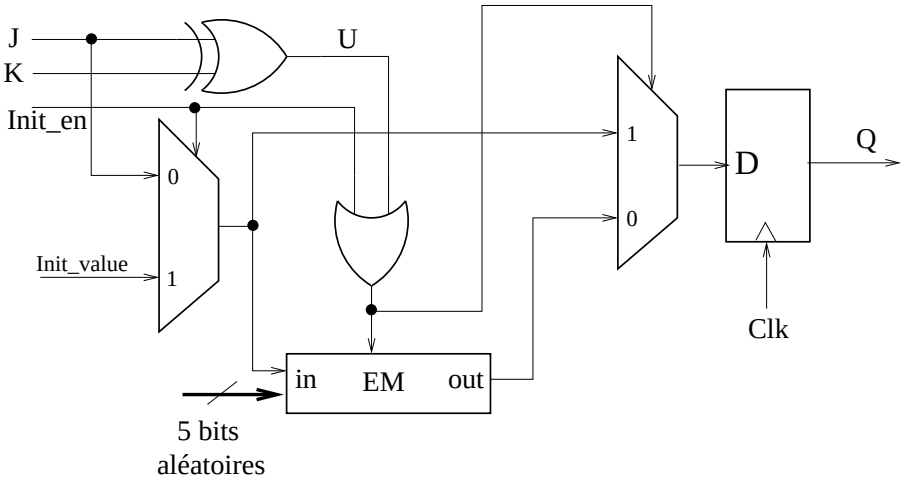


FIGURE 1.17 – Architecture de la technique des mémoires *EM*.

A chaque cycle, une EM est mise à jour par le bit sortant de la bascule JK quand le noeud de variable correspondant ne tombe pas à l'état bloqué (équivalent au cas où les entrées de la bascule JK sont différentes). Lorsque le noeud de variable est dans un état bloqué, un seul bit est aléatoirement sélectionné à partir des bits précédemment stockés dans l'EM et transmis comme le bit sortant. Ainsi, en utilisant les mémoires EM, le bit sortant à chaque cycle est seulement produit à partir de l'état non bloqué. Le mécanisme de génération aléatoire des bits stochastiques est donc établi. Cette technique permet de réduire la corrélation dans le flux stochastique, et d'augmenter le niveau d'activité de commutation aléatoire. Ce principe a été démontré et utilisé pour décoder des codes LDPC dans [16].

Technique *RHS* :

Cette technique correspond à un nouvel algorithme de décodage stochastique - l'algorithme RHS (*Relaxed Half Stochastic*) - qui combine les éléments de l'algorithme du décodage stochastique conventionnel et celui du décodage conventionnel BP. En clair, les noeuds de parité utilisent la technique de décodage stochastique tandis que les noeuds de variable utilisent les principes de mise à jour de l'algorithme BP [86, 87]. Dans ce cas, les entrées des noeuds de variable sont estimées à partir des bits de flux stochastiques entrant, puis ces probabilités sont converties en des valeurs LLR. Après les traitements arithmétiques au sein des noeuds de variable, les probabilités calculées sont transformées en flux stochastiques. Le reste de l'algorithme est déroulé dans le domaine stochastique. En particulier, l'échange entre les noeuds de parité et les noeuds de variable est réalisé à partir des flux stochastiques. Ce nouvel algorithme permet d'atteindre une meilleure performance asymptotique, proche de l'algorithme de décodage idéal. De plus, il donne une faible complexité par rapport à la technique des mémoires EM.

Technique des mémoires TFM :

Si chaque mémoire d'arc est associée à une branche sortant du graphe, alors le nombre de

mémoires EMs pour décoder des codes LDPC présents dans les standards devient inconsidérable. Par exemple, un coût de milliers de mémoires est requis pour les codes LDPC de taille $n > 1000$ et rend une grande complexité matérielle, en particulier lorsqu'un décodage stochastique en parallèle est envisagé. Une approche de mémorisation alternative a été proposée afin de remplacer des mémoires EMs : des mémoires TFM (*Tracking Forecast Memories* - en anglais) [88]. Ainsi, chaque TFM est attribuée à une branche sortant du graphe. Cette technique mesure la probabilité du flux stochastique sortant du noeud de variable et met à jour la mémoire correspondante en s'appuyant sur la trace de l'observation dans le passé. Cela permet de caractériser les sorties récentes. Cette technique diminue significativement la complexité matérielle pour des performances similaires à l'approche EM. Pourtant, le nombre des TFMs appliquées aux branches sortant de graphe est égal le nombre des EMs utilisées, alors encore élevé. Par conséquent, Sharifi Teharani *et al.* ont proposé une autre approche de mémoire basée sur la même idée de TFM : des mémoires MTFM (*Majority-based Tracking Forecast Memories* - en anglais) pour décoder les codes LDPC (2048,1723) [4]. Le technique MTFM qui reprend le principe de fonctionnement de TFM, seule la règle de mise à jour de la mémoire change. Elle est basée sur la majorité des bits regénératifs sortant. Les MTFMs sont seulement assignées à des noeuds de variable et alors réduisent forcément la complexité.

1.4 Décodage stochastique des turbocodes

Dans la section précédente, nous avons abordé les concepts généraux du décodage stochastique et aussi les techniques de mémoires proposées pour décoder des codes LDPC. A l'aide de ces techniques, les décodeurs stochastiques des codes LDPC ont montré des résultats notables en termes de performance et d'efficacité architecturale. Une question se pose alors ici : *peut-on appliquer l'approche stochastique et les techniques de mémoire afin de construire un turbo-décodeur stochastique* ?

En effet, le décodage des turbocodes possède des caractéristiques proches de celles des codes LDPC :

a. Ces deux familles de codes traitent des opérations arithmétiques multiplication, division et addition sur des probabilités récupérées à partir du canal.

b. Les codes LDPC propagent des messages probabilistes sur un graphe bipartite. Les turbocodes propagent des messages probabilistes sur un treillis.

c. Les codes LDPC effectuent l'échange bilatéral en même temps des messages probabilistes entre les noeuds de variable et les noeuds de parité, tandis que les turbocodes effectuent l'échange des messages probabilistes dans les deux sens *aller* et *retour* sur le treillis.

Grâce à ces similitudes de décodage entre les deux familles de codes, nous pouvons constater qu'une construction d'un turbo-décodeur stochastique est réalisable. La suite de cette section montre l'algorithme de décodage stochastique basé sur le treillis pour les turbocodes. Puis, un résumé de la complexité éventuelle de cet algorithme est introduit en termes d'opérations stochastiques. Des caractéristiques des opérations stochastiques pour le décodage stochastique des turbocodes sont enfin mentionnées.

1.4.1 Principe du décodage stochastique des turbocodes

Le principe de décodage stochastique des turbocodes est similaire à celui d'un décodage des turbocodes en virgule flottante : les deux décodeurs SISO s'échangent des probabilités sur le treillis et améliorent leur performance par l'échange des probabilités extrinsèques comme illustré en figure 1.18. La figure 1.18(b) résume l'ensemble des étapes de calcul du décodeur stochastique $SISO_1$. Le processus d'échange des flux stochastiques est illustré par des lignes pointillées. L'entrée de ce décodeur est deux séquences reçues à partir du canal. Ce décodeur fonctionne dans le domaine stochastique. C'est pourquoi, il doit commencer par convertir les données issues du canal en probabilités *a priori* équivalentes qui sont conservées dans une mémoire, grâce à un module ***Conv2P***.

Le processus de décodage stochastique se déroule par la conversion de ces probabilités en flux stochastiques à l'aide d'un module ***ConvP2S*** qui se compose de générateurs (pseudo-)aléatoires uniformes et des comparateurs. L'échange d'information entre deux décodeurs SISO est l'échange itératif des flux stochastiques extrinsèques représentant des probabilités extrinsèques. Chaque décodeur SISO binaire est équipé de deux compteurs correspondant à deux symboles possibles. L'occurrence de bit à "1" à l'entrée de chaque compteur représente la probabilité *a posteriori* du symbole correspondant. Pour un décodeur stochastique SISO, il faut initialiser des métriques récurrentes *aller/retour* et des informations extrinsèques qui sont préalablement inconnues. Donc, elles sont initialisées à des valeurs équiprobables. A chaque étape de traitement, un des bits du flux stochastique sortant du module ***ConvP2S***, ce qui constitue un *cycle de décodage - DC*.

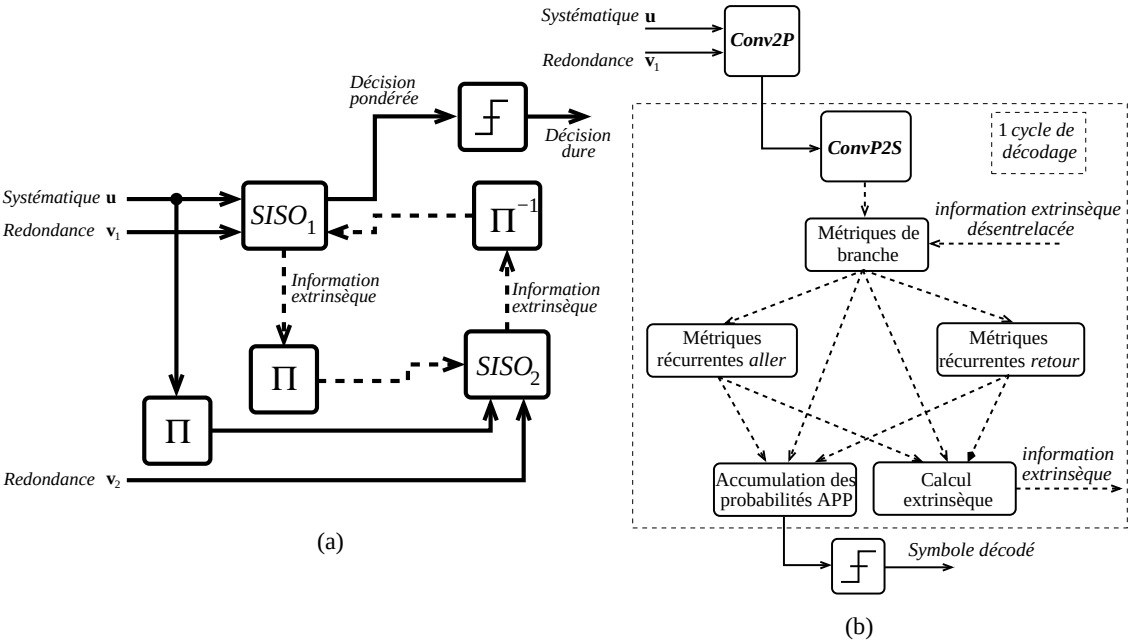


FIGURE 1.18 – (a) Schéma typique de principe d'un turbo-décodeur. (b) Schéma d'un décodeur stochastique $SISO_1$.

DC_{max} est défini comme le nombre de cycles de décodage maximal, i est le cycle de décodage courant.

Algorithme de décodage stochastique des turbocodes basé sur l'algorithme MAP :

Si $i = 0$ - étape d'initialisation - **alors** :

1. Initialiser des flux stochastiques représentant des métriques récurrentes *aller*/*retour* à des valeurs équiprobables.
2. Initialiser des flux stochastiques représentant des probabilités extrinsèques sortant de chaque décodeur à des valeurs équiprobables.
3. Initialiser le contenu des compteurs à "0".
4. Traiter en parallèle des opérations arithmétiques sur des séquences reçues dans le domaine probabiliste (les expressions (1.17) et (1.20)) au sein de deux décodeurs. Normaliser les probabilités obtenues afin d'assurer une même échelle de probabilité. Stocker ces probabilités dans une mémoire - module ***Conv2P***.

Pour $i \leftarrow 1$ **à** DC_{max} - traitement stochastique en parallèle au sein de deux décodeurs *SISO* - **faire :**

1. **Entrelacer et désentrelacer les flux stochastiques extrinsèques sortant :** Le flux stochastique extrinsèque sortant du premier décodeur est entrelacé et propagé au décodeur $SISO_2$. Celui sortant du second décodeur est désentrelacé et propagé au décodeur $SISO_1$.
2. **Convertir des probabilités *a priori* en flux stochastiques - module *ConvP2S* :** Les deux décodeurs génèrent de propres flux stochastiques représentant des probabilités exprimées dans les équation (1.17) et (1.20).
3. **Calculer des métriques de branche :** Les deux décodeurs réalisent des calculs stochastiques des métriques de branche(l'expression(1.15)).
4. **Calculer des métriques récurrentes :** Ces flux des métriques de branche se propagent sur le treillis pour trouver des métriques récurrentes *aller* et *retour* (l'expression(1.14)). Les métriques récurrentes stochastiques sont ensuite normalisées.
5. **Calculer des probabilités *a posteriori* :** Chaque décodeur *SISO* calcule les deux flux stochastiques décrivant les deux probabilités *a posteriori* possibles (l'expression (1.11)). A chaque cycle de décodage, quand un bit dans un flux stochastique *a posteriori* est calculé, le compteur correspondant le capte et est incrémenté d'une unité.
6. **Calculer des probabilités extrinsèques :** Les informations extrinsèques stochastiques sont calculées et normalisées (l'expression (1.19)) dans chaque décodeur élémentaire. Elles seront utilisées durant le prochain cycle de décodage par l'autre décodeur élémentaire.

Si $i = DC_{max}$, **prendre la décision** : les valeurs dans les deux compteurs du décodeur $SISO_1$ sont comparées. Lorsqu'une valeur est plus grande que l'autre, le symbole correspondant est choisi comme le résultat de la décision.

Le nombre de cycles de décodage maximum DC_{max} est déterminé en se basant sur la meilleure performance de décodage.

1.4.2 Estimation de la complexité de décodage stochastique pour l'algorithme MAP

Dans cette section, nous évaluons la complexité de calcul stochastique d'un turbocode basé sur l'algorithme MAP. La complexité du calcul d'un décodeur correspond à des opérations basiques imposées par l'algorithme comme des multiplications, divisions et additions dans une section de treillis. Dans notre code simple-binaire CRSC (m=2), le treillis est composé de $(2^{\nu} = 8)$ états, avec $2^m = 2$ branches arrivant à chaque état (figure 1.19). Alors, chaque section du treillis possède 16 branches.

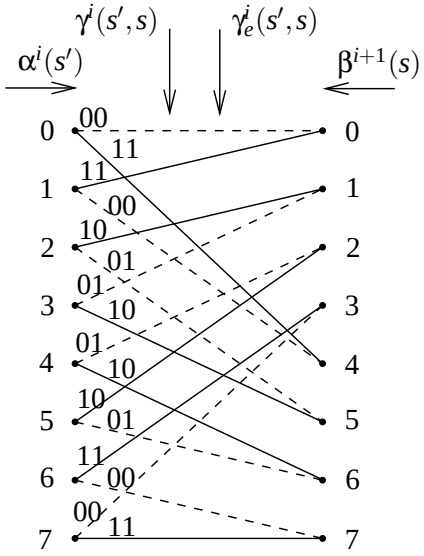


FIGURE 1.19 – Une section du treillis.

Calcul stochastique des métriques de branche :

Le processus stochastique dans cette étape se déroule après la conversion des probabilités en flux stochastiques. La conversion doit être effectuée pour toutes les métriques de branche possibles de $\gamma^i(k)(k = 0 \rightarrow 15)$ et de $\gamma_e^i(l)(l = 0 \rightarrow 15)$. Cependant, chaque codeur élémentaire produit un symbole codé de 2 bits, y compris 1 bit de redondance. Donc, il a besoin seulement de $(2^2 = 4)$ métriques de branche différentes $\gamma^i(j)(j = 0 \rightarrow 3)$ et 2 métriques de branche de redondance $\gamma_e^i(j)(j = 0 \rightarrow 1)$ représentés respectivement par 4 flux stochastiques $\gamma^i(k)(k = 0 \rightarrow 3)$ et 2 flux stochastiques $\gamma_e^i(l)(l = 0 \rightarrow 1)$. Par ailleurs, 4 métriques de branche $\gamma^i(j)(j = 0 \rightarrow 3)$ exigent des multiplications pour tenir compte des informations extrinsèques fournies par l'autre décodeur SISO. En conclusion, à chaque section, cette étape de calcul nécessite 4 multiplications stochastiques.

Calcul stochastique des métriques récurrentes :

Les flux stochastiques interprétant des métriques de branche se propagent sur les branches du treillis (un bit par branche et par direction). Il servent à trouver des flux stochastiques représentant des métriques récurrentes *aller* et *retour*. Le règle de mise à jour des métriques récurrentes *aller* d'après l'expression 1.24 comporte 2 multiplications et une addition de 2^m chemins concurrents

d'entrée $\alpha^{i-1}(s')\gamma^{i-1}$ pour chaque état.

$$\begin{array}{lll} \alpha^i(s) & = \sum_{s'=0}^{2^\nu-1} \alpha^{i-1}(s')\gamma^{i-1}(s',s) & \text{pour} \quad i \in (1..k) \\ \beta^i(s) & = \sum_{s'=0}^{2^\nu-1} \beta^{i+1}(s')\gamma^i(s,s') & \text{pour} \quad i \in (k-1..0) \end{array} \tag{1.24}$$

Ensuite, ces métriques récurrentes doivent être toutes normalisées. Pour le faire, chaque état nécessite une addition de ($2^\nu = 8$) entrées et une division. Donc, ($2^\nu = 8$) additions de 8 entrées et 8 divisions sont exigées. Ainsi, le calcul des métriques récurrentes *aller* à chaque section requiert ($2^{\nu+m} = 16$) multiplications stochastiques, ($2^\nu = 8$) additions stochastiques de 2 entrées, 2^ν additions stochastiques de 8 entrées et ($2^\nu = 8$) divisions stochastiques. Des opérations impliquées dans le sens *retour* sont similaires à celles dans le sens *aller*, la complexité associée est identique.

Calcul stochastique des probabilités *a posteriori* :

Le calcul des probabilités *a posteriori* selon l'expression (1.25) exige des multiplications et additions stochastiques.

$$P\left(d^i = j|\mathrm{u},\mathrm{v}_1\right) = \sum_{(s',s)/d^i(s',s)=j} \alpha^i(s)\gamma^i\left(s',s\right)\beta^{i+1}(s) \tag{1.25}$$

Ce calcul est traité pour toutes les branches et nécessite $2^{\nu+m}$ multiplications stochastiques pour trouver $2^{\nu+m}$ métriques de branche *a posteriori* $\alpha^i(s)\gamma^i(s',s)\beta^{i+1}(s)$. Par ailleurs, le calcul en équation 1.25 pour chaque symbole binaire j=(0,1) implique la somme de 2^ν valeurs concurrentes correspondant à 2^ν chemin concurrents. Alors, il faut 2 additions stochastiques de ($2^\nu = 8$) entrées. La division stochastique de deux probabilités *a posteriori* (en expression 1.13) peut être supprimée lorsque le décodeur élémentaire est équipé de deux compteurs qui accumulent les flux stochastiques représentant des probabilités *a posteriori*. Dans ce cas, la décision dure se réfère à une sélection de la valeur de compteur la plus grande en utilisant un comparateur. Le symbole binaire correspondant à la valeur maximum est le symbole décodé.

Calcul stochastique des probabilités extrinsèques :

Le calcul d'information extrinsèque correspondant à chaque symbole binaire $j = (0,1)$ est exécuté d'après l'expression(1.19).

$$\mathrm{Pr}_e^{out}\left(d^i = j|\mathbf{u},\mathbf{v}_1\right) = \frac{\sum_{(s',s)/d_i(s',s)=j} \alpha^i(s')\beta^{i+1}(s)\gamma_e^i(s',s)}{\sum_{(s',s)} \alpha^i(s')\beta^{i+1}(s)\gamma_e^i(s',s)} \tag{1.26}$$

Des multiplications stochastiques sont requises pour multiplier les flux stochastiques récurrentes *aller* $\alpha^i(j)(j = 0 \to 7)$ et *retour* $\beta^{i+1}(k)(k = 0 \to 7)$ avec le flux stochastique de redondance $\gamma_e^i(l)(l = 0 \to 1)$. Le nombre de multiplications est égal à celui de branches (égal à 16). Ensuite, les résultats de multiplication obtenus doivent être accumulés afin de fournir des probabilités extrinsèques. Deux additions stochastiques de 8 entrées sont indispensables pour produire deux probabilités extrinsèques correspondant à deux symboles binaires possibles. Deux divisions sont

assumées pour effectuer la normalisation des probabilités extrinsèques.

Complexité globale :

Le tableau 1.5 résume l'estimation de la complexité en termes des opérations stochastiques correspondant à une section de treillis d'un décodeur stochastique SISO basé sur l'algorithme MAP. Le décodeur binaire à 8 états est considéré dans notre cas.

	Addition	Multiplication	Division
Métriques de branche		4	
Métriques récurrentes (*aller*/*retour*)	16	16	8
Probabilités extrinsèques	2	16	2
Probabilités *a posteriori*	2	16	
Total	36	68	18

TABLE 1.5 – Complexité de calcul d'une section d'un décodeur stochastique binaire SISO basé sur l'algorithme MAP.

Mémoires requises :

Le décodeur stochastique a besoin d'un jeu de mémoires pour garder les probabilités *a priori* mises à l'échelle par W bits. Ces mémoires sont actuellement inévitables. De plus, ce type de décodeur possède des caractéristiques proches de celles des codes LDPC. Par conséquent, afin d'obtenir une bonne performance de décodage, certaines types de mémoires peuvent être indispensables pour supprimer le problème de corrélation. Les chapitres suivants présentent en détail différentes quantités de mémoires nécessaires selon de différentes techniques de décodeurs proposées.

Lors qu'un turbo-décodeur de deux dimensions, sa complexité de calcul contient donc une double complexité d'un décodeur élémentaire présenté en tableau 1.5 et aussi la complexité des entrelaceurs. Ayant pour but d'augmenter la vitesse d'exécution, une architecture de décodage de multiples turbo-décodeurs en parallèle avec plus grande complexité est considérée. Lors une architecture entièrement parallèle est favorisée, sa complexité est quasiment linéaire au nombre de décodeurs intégrés M. Par conséquent, la complexité d'une section de treillis de multiple décodeurs stochastiques SISO est égale à la complexité d'un décodeur SISO multipliée par un facteur M.

1.4.3 Caractéristique des opérations pour le décodage stochastique de turbocodes

Comme déjà mentionné ci-dessus, le turbo-décodeur stochastique nécessite un grand nombre d'opérations pour une section de treillis afin de produire un symbole estimé. Cette section relève les caractéristiques des opérations stochastiques en termes de contribution à la convergence du décodeur.

Les opérations division :

Elles sont associées à des bascules JK. Comme déjà montré dans les sections précédentes, l'inconvénient principal relié aux bascules JK est le problème de *corrélation*. Dès lors, le décodeur stochastique basé sur l'algorithme MAP proposé dans ce cadre profite de l'avantage des techniques de mémoires introduites en section 1.3.2.2. Une mémoire est assignée à chaque flux stochastique qui représente la probabilité récurrente *aller* (ou *retour*) correspondant pour se prémunir contre la corrélation. D'une façon similaire, des mémoires sont associées aux flux stochastiques utilisés pour des calculs de l'information extrinsèque.

Les opérations multiplication :

La multiplication de deux flux stochastiques indépendants est traitée par une porte logique AND à deux entrées. De même, la multiplication d'un jeu de N probabilités $\Pr_0, \Pr_1, ..., \Pr_{N-1}$ peut être réalisée par N flux stochastiques mutuellement indépendants qui sont les entrées d'une porte logique AND à N entrées. La probabilité de sortie de la porte logique est égale à $\Pr_c = \prod_{i=0}^{N-1} \Pr_i$. A chaque instant, le bit de chaque séquence d'entrée de la porte logique AND contribue systématiquement au bit sortant de l'opération. De plus, la multiplication par une porte AND ne corrèle pas le flux sortant, contrairement à la division par une bascule JK.

Les opérations addition :

L'addition de N probabilités nécessite une mise à l'échelle pour que le résultat reste dans l'intervalle $[0;1]$, représentable par un flux stochastique. Le résultat de cette mise à l'échelle est égal à $\Pr_s = \sum_{i=0}^{N-1} \frac{1}{N} \Pr_i$. Pourtant, l'utilisation d'un multiplexeur avec une probabilité de sélection aléatoire de $1/N$ ne permet pas à chaque entrée de contribuer systématiquement au bit de sortie à chaque instant. C'est pourquoi, une séquence binaire plus longue est indispensable pour conserver la précision recherchée. Plus le nombre d'entrée du multiplexeur est élevé, plus large est la séquence de sortie requise. Par exemple, si seuls 10 bits sont suffisants pour deux séquences Bernoulli interprétant les deux probabilités 0,2 et 0,5. Par contre, le résultat de la somme de deux probabilités $\Pr_s = \frac{1}{2}(0,2+0,5) = 0,35 = \frac{7}{20}$ nécessite une longueur minimum d'environ 20 bits pour présenter une bonne précision.

Cette contrainte est problématique pour le processus de décodage basé sur l'algorithme MAP dans notre cas, car de nombreuses additions stochastiques à deux ou huit entrées sont inévitables pour normaliser des métriques récurrentes, pour calculer des probabilités extrinsèques ainsi que pour calculer des probabilités *a posteriori*. Dès lors, le traitement des additions stochastiques à partir de multiplexeurs ralentit significativement le processus de convergence du décodeur. Ce problème s'accentue lors d'une implémentation entièrement parallèle de décodeurs stochastiques SISO. Dans ce cas, le nombre de multiplexeurs à huit entrées nécessaire est multiplié par M - le nombre des décodeurs stochastiques SISO implémentés. Pour cette raison, des solutions doivent être investiguées pour remplacer des multiplexeurs de grande taille.

1.5 Conclusion

Dans ce premier chapitre, nous avons rappelé brièvement les concepts de base des turbocodes qui sont intégrés dans la plupart des standards de communication modernes. L'algorithme de décodage exploite les opérations dans le domaine log-probabiliste, et permet d'obtenir un gain de performance élevée. Des résultats récents montrent que le parallélisme de décodage permet de concevoir un turbo-décodeur puissant en termes d'efficacité architecturale. Plusieurs implémentations ont été proposées même pour les standards courants : 3GPP-LTE et WiMAX. Néanmoins, très peu d'architectures atteignent le débit du Gbps avec cette technique en raison de la complexité matérielle.

En 1967, Gaines a montré que des circuits stochastiques simples étaient capables d'effectuer des opérations arithmétiques complexes ce qui permit d'ouvrir un nouveau chemin de recherche pour la réalisation de décodeurs des codes puissants dès 2003 [71]. Cependant, cette approche a été appliquée avec succès seulement pour des décodeurs stochastiques de petites tailles. Dans les années récentes, S. Sharifi Tehrani, par des résultats obtenus sur FPGA et ASIC, a prouvé que des architectures matérielles de décodeurs stochastiques pour des codes LDPC pouvaient avoir des caractéristiques intéressantes pour la montée en débit. En particulier, l'approche stochastique a permis un décodage entièrement parallèle de codes LDPC, ce qui n'est généralement pas si simple pour les autres techniques proposées en littérature.

Inspirés par les résultats intéressants de décodage stochastique de codes LDPC en littérature, nous avons amené l'idée de concevoir un décodage stochastique pour les turbocodes. Cette idée se base sur des similitudes entre un code LPDC et un turbocode, de la représentation en graphe au principe d'échange de flux stochastiques entre les noeuds du graphe. Nous avons également présenté le principe de décodage stochastique de turbocodes basé sur l'algorithme MAP où les flux stochastique se propagent dans les deux sens aller/retour sur le treillis d'un décodeur élémentaire SISO. Des limitations en termes de vitesse de convergence d'un turbo-décodeur stochastique au niveau des opérations stochastiques sont aussi analysées. Cette problématique devient plus importante, lors d'un décodage de multiple décodeurs SISO en parallèle avec une taille de milliers de bits. Dès lors, de nouvelles techniques doivent être investiguées afin de résoudre ces limitations.

Par conséquent, le deuxième chapitre sera consacré à l'architecture détaillée d'un turbo-décodeur stochastique et à sa performance de décodage théorique. Le troisième chapitre propose des solutions alternatives pour améliorer la convergence du turbo-décodeur stochastique. Le dernier chapitre présente les premiers résultats d'implémentation de notre architecture sur FPGA en termes de performance atteinte et de complexité matérielle.

CHAPITRE 2

Décodage stochastique de turbocodes basé sur l'algorithme MAP

Sommaire

2.1 Introduction

Dans le chapitre précédent, nous avons montré qu'il était possible de concevoir un turbo-décodeur stochastique basé sur l'algorithme MAP. Lorsque l'information probabiliste reçue à partir du canal est convertie en des séquences de Bernoulli, alors, le processus d'échange des messages au sein du décodeur stochastique se déroule par l'échange des bits entre les états du treillis à travers un réseau de portes logiques. La construction de ce type de réseau nécessite des connaissances sur chaque module le constituant. Ainsi, dans ce chapitre, nous allons introduire le contexte de construction d'un turbo-décodeur stochastique par des descriptions de modules élémentaires associés à l'algorithme MAP.

Dans la première section, le concept de l'architecture du turbo-décodeur stochastique MAP est introduit avec la propagation des flux stochastiques au sein du turbo-décodeur. Puis, la description du décodeur SISO est détaillée. Les quatre sections suivantes détaillent la construction de chaque module. La dernière section commence par le résultat de performance en termes de taux d'erreur et des discussions, et se termine par la mesure de la complexité et du débit de l'architecture du turbo-décodeur stochastique.

2.2 Architecture du turbo-décodeur convolutif stochastique

2.2.1 Modèle d'un turbo-décodeur stochastique

La section 1.4.1 du chapitre 1 a montré que le principe du décodage stochastique pouvait être directement appliqué aux turbocodes grâce à l'échange de messages probabilistes dans le domaine stochastique. La figure 2.1 illustre l'architecture d'un turbo-décodeur stochastique simple-binaire qui se compose d'une mémoire ROM, d'un module *série-parallèle*, de deux décodeurs stochastiques SISO, des entrelaceurs et d'un module de contrôle. Les symboles démodulés sont successivement convertis en des probabilités *a priori* à l'aide d'une mémoire ROM. Ces symboles sont quantifiés sous la forme (p,q) avec p le nombre de bits total de représentation en virgule fixe, et q le nombre de bits de la partie fractionnaire. La sortie de mémoire ROM est des probabilités représentées sur W bits. Ces probabilités sont stockées en série dans des mémoires et envoyées en parallèle à deux décodeurs stochastiques élémentaires SISO par le module *série-parallèle*. Le module *contrôle* assure l'ordonnancement du turbo-décodeur.

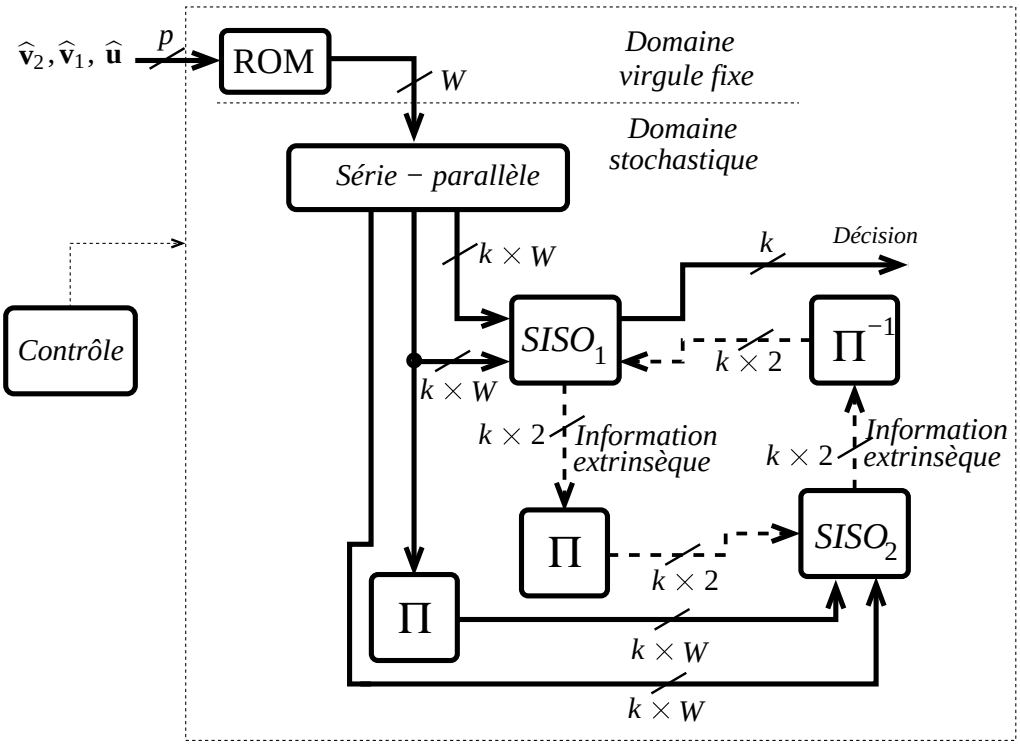


FIGURE 2.1 – Architecture d'un turbo-décodeur stochastique.

L'utilisation de la mémoire ROM a un avantage : l'augmentation de la précision des symboles entrants ou des probabilités sortantes n'affecte pas la complexité de quelques modules de calcul constituants du décodeur stochastique, par exemple, des métriques récurrentes et des entrelaceurs implémentés. En effet, la communication entre différents modules d'un décodeur est assurée par des lignes d'un seul bit à travers un réseau des portes logiques. Cependant, l'augmentation de la précision entraîne seulement l'augmentation de la taille de mémoire ROM, des générateurs

(pseudo-)aléatoires et des comparateurs. Ce n'est pas le cas pour les traitements dans le domaine des virgules fixes basés sur l'algorithme MAP ou plutôt sur sa version simplifiée subMAP. Dans ce cas, l'augmentation de la précision implique l'augmentation de la complexité des entrelaceurs et opérateurs. De plus, pour les algorithme MAP et subMAP, l'augmentation de la précision augmente potentiellement la latence de chaque itération. Du coup, l'échange des messages entre les états exige plus de temps. C'est pourquoi l'augmentation de la précision peut réduire le débit de décodage.

Toutes les probabilités arrivent en même temps à deux décodeurs stochastiques SISO, auxquels elles sont transformées en flux stochastiques dans le domaine stochastique. Ces flux stochastiques se propagent au sein de deux décodeurs stochastiques SISO pour effectuer le processus de décodage. Ces deux décodeurs stochastiques SISO s'échangent des flux stochastiques extrinsèques afin d'améliorer la performance de décodage. Cet échange est représenté par des lignes de $2*k$ bits qui correspond à deux probabilités extrinsèques de deux symboles possibles du code binaire et correspond également à une séquence d'information de taille k bits. Finalement, les symboles décodés du décodeur stochastique $SISO_1$ sont sortis en parallèle et sont considérée comme la décision du turbo-décodeur.

2.2.2 Architecture d'un décodeur stochastique élémentaire SISO

Comme déjà expliqué au chapitre 1, le décodage stochastique exige des opérations stochastiques pour l'algorithme MAP basé sur une représentation en treillis. Dans notre cas, les deux codes composants sont circulaires où le treillis commence et termine par le même état. Ainsi, le décodeur est dit *décodeur stochastique APP*. La figure 2.2 décrit l'architecture du décodeur stochastique APP. L'échange des messages stochastiques entre plusieurs sections du décodeur APP est en particulier illustré. Le nombre de sections est égal au nombre de symboles à décoder. Chaque section comprend 5 modules.

Le module Γ reçoit des probabilités $\Pr(\widehat{u}^i)$ et $\Pr(\widehat{v}^i)$ quantifiées sur W bits. Ces probabilités sont respectivement associées à des informations souples $\widehat{u}^i$ et $\widehat{v}^i$ correspondant à $i^{ème}$ symbole émis d^i et à son bit de redondance y^i. Ces probabilités sont ensuite transformées en deux flux stochastiques qui sont propagés pour évaluer les métriques de branches. Puis les métriques récurrentes *aller* du module *A* et les métriques récurrentes *retour* du module *B* sont estimées. Ces deux modules impliquent un traitement récursif où les vecteurs de métriques d'état α^i et β^{i+1} et les métriques de branche sont utilisées pour produire les vecteurs de métriques d'état α^{i+1} et β^i. Le module *DEC* observe les séquences stochastiques *a posteriori* et détermine la décision finale $\widehat{d^i}$ de chaque symbole binaire d^i émis. Un module supplémentaire est requis lorsque le décodeur SISO est une part du turbo-décodeur : le module Ext. Ce module calcule les vecteurs de probabilités extrinsèques sortant Pr_e^{out} qui sont subséquemment utilisées par le module Γ du deuxième décodeur SISO comme les vecteurs de probabilités extrinsèques entrant Pr_e^{in}. Tous les modules sont connectés et échangent des flux stochastiques à travers un réseau de portes logiques correspondant à la représentation sous forme de treillis du code. L'échange des flux stochastiques entre les métriques récurrentes *aller/retour* de deux sections adjacentes est établi par des bus de taille 8 bits correspondant aux 8 états possibles de treillis. Chaque étape de décodage correspond au

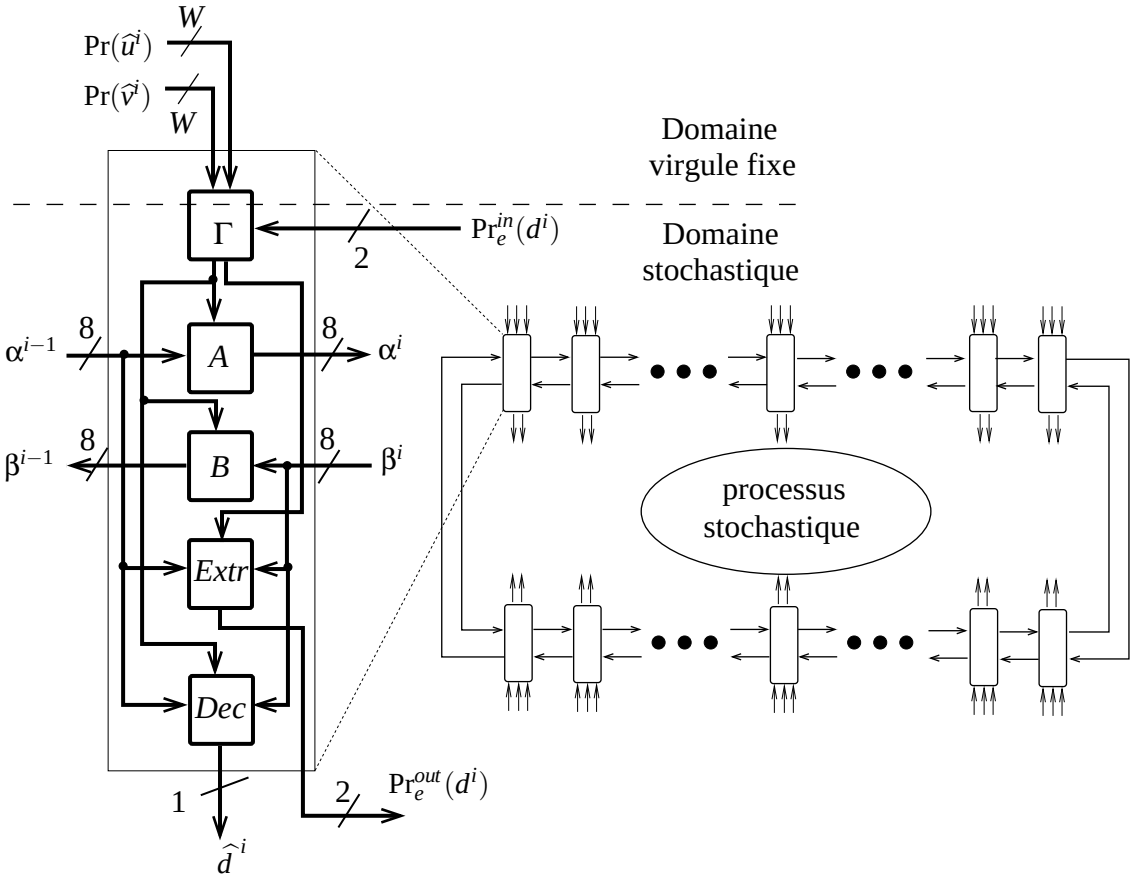


FIGURE 2.2 – Architecture du décodeur stochastique circulaire APP.

traitement d'un nouveau bit pour chaque module stochastique. Elle fournit à la sortie du module *Ext* deux bits correspondant à deux probabilités extrinsèques possibles et produit à la sortie du module *DEC* un bit représentant la probabilité *a posteriori* pour la prise de décision. On appelle cette étape élémentaire *un cycle de décodage*. Le processus de décodage stochastique s'achève quand le nombre maximum de cycles de décodage est atteint. Les sections suivantes détaillent successivement les différents modules du décodeur stochastique APP.

2.2.3 Interprétation des probabilités issues du canal

L'algorithme MAP applique des opérations arithmétiques sur des probabilités. Mais, les probabilités fournies à l'algorithme doivent être préalablement déterminées à partir de la sortie du canal lors d'une étape de démodulation. c^i est défini comme le symbole binaire transmis à l'instant i. $\widehat{c}^i$ est la sortie du canal démodulée correspondant au symbole c^i (figure 1.1). Lorsqu'une modulation BPSK (*Binary Phase Shift Keying*) est considérée, c'est-à-dire, chaque bit $b \in (0;1)$ lors de l'émission est respectivement converti à un symbole s de la constellation $\chi = (-1;1)$ et est modulé ensuite et transmis sur le canal de propagation. Un canal AWGN (*Additive White Gaussian Noise*) est caractérisé par une variance σ^2, la probabilité conditionnelle $\Pr(\widehat{c}^i|c^i)$ est alors

égale à :

$$\Pr(\widehat{c}^i|c^i=b)=\frac{1}{\sqrt{2\pi\sigma^2}}\exp(-\frac{1}{2\sigma^2}\left\|\widehat{c}^i-s\right\|^2) \tag{2.1}$$

Le Logarithme du Rapport de Vraisemblance (LRV) du symbole $\widehat{c}^i$ conditionnel à c^i qui représente l'information pondérée de la fiabilité sur la décision conditionnelle s'exprime :

$$\mathcal{L}(c^i|\widehat{c}^i)=\ln\left(\frac{\Pr(c^i=1|\widehat{c}^i)}{\Pr(c^i=0|\widehat{c}^i)}\right) \tag{2.2}$$

Ainsi, la décision sur le bit c^i est donnée par le signe du LRV $\mathcal{L}(c^i|\widehat{c}^i)$ et l'information de fiabilité de cette décision correspond à $\left|\mathcal{L}(c^i|\widehat{c}^i)\right|$. De plus, conformément à la formule de Bayes :

$$\mathcal{L}\left(c^i|\widehat{c}^i\right)=\ln\left(\frac{\Pr(c^i=1,\widehat{c}^i)}{\Pr(c^i=0,\widehat{c}^i)}\right)=\ln\left(\frac{\Pr(\widehat{c}^i|c^i=1)\Pr(c^i=1)}{\Pr(\widehat{c}^i|c^i=0)\Pr(c^i=0)}\right) \tag{2.3}$$

Si les bits de la source sont générés de la manière uniforme, $\Pr(c^i=0)=\Pr(c^i=1)=1/2$, alors :

$$\mathcal{L}\left(c^i|\widehat{c}^i\right)=\ln\left(\frac{\Pr(\widehat{c}^i|c^i=1)}{\Pr(\widehat{c}^i|c^i=0)}\right) \tag{2.4}$$

En combinant les équations (2.1) et (2.4), on obtient :

$$\mathcal{L}(c^i|\widehat{c}^i)=L_c.\widehat{c}^i \tag{2.5}$$

où $L_c=\left(\frac{2}{\sigma^2}\right)$ est la valeur de fiabilité du canal [89]. L'expression (2.5) montre que LRV du symbole émis à l'instant i est une fonction proportionnelle à la sortie du canal démodulée correspondant. Donc, au lieu de propager des messages dans le domaine probabiliste sur le treillis, une des solutions est de propager des messages dans le domaine logarithmique LRV. Dans ce cas, les probabilités sont directement fournies au décodeur sous la forme de LRV, et le décodeur réalise des opérations sur ces valeurs LRV [89].

En outre, les probabilités conditionnelles $\Pr(c^i=1|\widehat{c}^i)$ et $\Pr(c^i=0|\widehat{c}^i)$ peuvent être déduites du LRV par :

$$\Pr(c^i=1|\widehat{c}^i)=\tfrac{1}{1+\exp(-L_c.\widehat{c}^i)}\quad\text{et}\quad\Pr(c^i=0|\widehat{c}^i)=1-\Pr(c^i=1|\widehat{c}^i) \tag{2.6}$$

De plus, la probabilité du symbole reçu démodulé correspondante au symbole émis $\Pr(\widehat{c}^i|c^i=0)$ s'exprime :

$$\Pr(\widehat{c}^i|c^i=0)=\frac{\Pr(\widehat{c}^i)-\Pr(\widehat{c}^i|c^i=1).\Pr(c^i=1)}{\Pr(c^i=0)} \tag{2.7}$$

A partir des équations 2.4, 2.5 et 2.7, on obtient :

$$\ln\frac{\Pr(\widehat{c}^i|c^i=1).\Pr(c^i=0)}{\Pr(\widehat{c}^i)-\Pr(\widehat{c}^i|c^i=1).\Pr(c^i=1)}=L_c.\widehat{c}^i \tag{2.8}$$

Donc, on déduit :

$$\Pr(\widehat{c}^i|c^i=1)=\frac{\frac{\Pr(\widehat{c}^i)}{\Pr(c^i=0)}.\exp(L_c.\widehat{c}^i)}{1+\frac{\Pr(c^i=1)}{\Pr(c^i=0)}.\exp(L_c.\widehat{c}^i)}=\frac{\frac{\Pr(\widehat{c}^i)}{2}.\exp(L_c.\widehat{c}^i)}{1+\exp(L_c.\widehat{c}^i)} \tag{2.9}$$

De manière similaire :

$$\Pr(\widehat{c}^i|c^i=0)=\frac{\frac{\Pr(\widehat{c}^i)}{\Pr(c^i=1)}.\exp(-L_c.\widehat{c}^i)}{1+\frac{\Pr(c^i=0)}{\Pr(c^i=1)}.\exp(-L_c.\widehat{c}^i)}=\frac{\frac{\Pr(\widehat{c}^i)}{2}.\exp(-L_c.\widehat{c}^i)}{1+\exp(-L_c.\widehat{c}^i)} \tag{2.10}$$

où $\frac{\Pr(\widehat{c}^i)}{2}$ est un facteur commun. Il peut être éliminé par des étapes de normalisation. C'est pourquoi :

$$\Pr(\widehat{c}^i|c^i=1)=\frac{\exp(L_c.\widehat{c}^i)}{1+\exp(L_c.\widehat{c}^i)} \quad \text{et} \quad \Pr(\widehat{c}^i|c^i=0)=1-\Pr(\widehat{c}^i|c^i=1) \tag{2.11}$$

2.2.4 Mise à l'échelle des sorties du canal démodulées

Comme déjà mentionné dans le chapitre 1, une des limitations importantes lors de la propagation des probabilités au sein du décodeur stochastique est due au problème de *corrélation*. Ce phénomène est particulièrement critique à fort rapport signal sur bruit - RSB. Dans ce cas, les LRV reçus deviennent élevés et les probabilités correspondant sont proches de 1(ou 0). Dans le cadre d'un décodage stochastique pour les turbocodes basé sur l'algorithme MAP, avant d'être utilisées pour calculer les probabilités, les sorties du canal sont mises à l'échelle à l'aide d'un coefficient *NDS* (présenté en section 1.3.2.2 du chapitre 1). Ce coefficient permet d'éviter que les probabilités du canal soient trop proches de 1 (ou 0). L'objectif est d'établir de nouvelles probabilités distantes d'au moins un écart de Δ des probabilités 1 et 0 (figure 2.3). Le coefficient *NDS* est choisi de telle sorte que la valeur de Δ soit relativement faible.

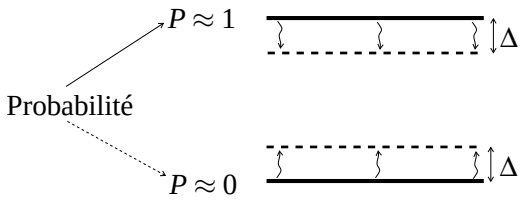


FIGURE 2.3 – Problème de corrélation résolu par un facteur d'atténuation.

Dans notre approche, nous proposons un coefficient *NDS* qui est proportionnel au RSB, pour lequel les sorties du canal reçues sont réduites. Supposons une transmission BPSK sur un canal AWGN, la nouvelle sortie du canal démodulée $\widehat{c}_0^i$ pour le $i^{ème}$ symbole(c^i) émis est calculée par :

$$\widehat{c}_0^i=NDS.\widehat{c}^i=\frac{\sigma^2}{\psi(\sigma).\Omega}\widehat{c}^i \Rightarrow \mathcal{L}'(c^i|\widehat{c}^i)=L_c.\widehat{c}_0^i=\frac{2}{\psi(\sigma).\Omega}.\widehat{c}^i \tag{2.12}$$

où σ est la déviation du bruit, Ω est la valeur maximum de variance du bruit. $\psi(\sigma)$ est un *facteur de correction* dépendant du RSB qui est choisi en se basant sur la meilleure performance TEB (taux d'erreur binaire). Selon notre observation, pour la transmission BPSK, $\Omega=\sigma_0^2$ correspond au RSB = 0, et la valeur $\psi(\sigma)$ est dans l'intervalle $[1;2)$. Le facteur NDS résultant permet de bonne performance de décodage. Ce coefficient est proportionnel au RSB, il assure donc la même activité de transition des bits pour différentes valeurs RSB.

La figure 2.4 montre l'influence du paramètre *NDS* sur la performance de décodage des codes convolutifs (k=200, $R=1/2$). Les coefficients $\Omega=\sigma_0^2=1$ et $\psi(\sigma)=1$, alors $NDS=\sigma^2$. La

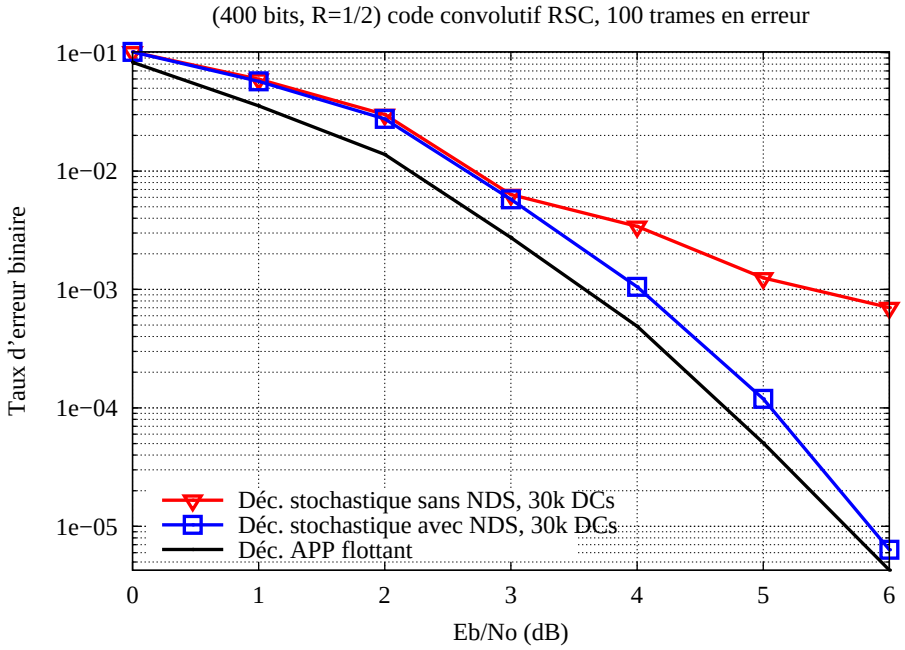


FIGURE 2.4 – Performance du décodage stochastique des codes convolutifs (k=200, $R = 1/2$) en fonction de paramètre *NDS*.

courbe rouge présente la performance de décodage sans ajoutant le coefficient *NDS*. La courbe bleue bénéficie l'avantage du coefficient *NDS*. Les résultats de décodage stochastique sont obtenus avec $DC_{max} = 30$k et sont comparés avec un décodage en virgule flottante (la courbe noire). Comme illustré, le coefficient *NDS* améliore significativement la performance de décodage en comparaison au cas sans *NDS* et permet de s'approcher de la performance de décodage en virgule flottante.

2.3 Module Γ

2.3.1 Rôle du module

L'objectif de ce module est de calculer les vecteurs des métriques de branche γ^i dans le domaine stochastique. Il assure aussi leur propagation pour le calcul des métriques récurrentes (dans les modules *A*, *B*) et pour décider le symbole émis (module *DEC*). Ce module permet aussi de produire les vecteurs stochastiques de probabilités de redondance γ_e^i utilisés pour estimer l'information extrinsèque sortant (module *Ext*). γ^i nécessite la connaissance des probabilités quantifiées $\Pr(\widehat{u}^i)$ et $\Pr(\widehat{v}_1^i)$ associées respectivement à des symboles émis u^i et ${v_1}^i$ et celle du vecteur de probabilités extrinsèques $\Pr_e^{in}(d^i)$ fournies par l'autre décodeur élémentaire dans le schéma turbo. Le module Γ convertit les probabilités $\Pr(\widehat{u}^i)$ et $\Pr(\widehat{v}_1^i)$ en deux flux stochastiques à l'aide de comparateurs et de générateurs des séquences aléatoires.

Pour tout $(j,k) \in (0;1)^2$, l'expression (1.18) du chapitre 1 est réécrite :

$$\gamma^i(j,k) = \frac{\Pr(\widehat{u}^i|d^i=j).\Pr(\widehat{v}_1^i|y_1^i=k).\Pr_e^{in}(d^i=j|\widehat{v}_2^i)}{\sum\limits_{(m,n)\in(0;1)^2} \Pr(\widehat{u}^i|d^i=m).\Pr(\widehat{v}_1^i|y_1^i=n).\Pr_e^{in}(d^i=m|\widehat{v}_2^i)} \tag{2.13}$$

Puisque toutes les métriques de branches ont le même facteur de normalisation qui peut être omis sans modifier l'efficacité de l'algorithme, il est suffisant de produire des métriques sous la forme de l'équation 2.11. En combinant les expressions 2.6, 2.11 et 2.13, on obtient pour tout $(j,k) \in (0;1)^2$:

$$\gamma^i(j,k) = \Pr(\widehat{u}^i|d^i=j).\Pr(\widehat{v}_1^i|y_1^i=k).\Pr_e^{in}(d^i=j|\widehat{v}_2^i) \tag{2.14}$$

Dans notre étude, le codeur binaire élémentaire produit un seul bit de redondance, le vecteur γ_e^i fournit les deux probabilités de redondance possibles correspondant au symbole binaire émis y^i. Ces probabilités peuvent être exprimées en fonction du symbole binaire reçu $\widehat{v}_1^i$, pour chaque $j \in (0;1)$:

$$\gamma_e^i(s',s) = \Pr(\widehat{v}_1^i|y_1^i=j).\delta^{(i,j)}(s',s) \tag{2.15}$$

où $\delta^{(i,j)}(s',s) = 1$ si et seulement s'il y a une transition entre l'état s' à l'instant (i) et l'état s à l'instant $(i+1)$ sur le treillis pour le symbole sortant $y_1^i = j$. Sinon, $\delta^{(i,j)}(s',s) = 0$. Les sorties de ce module sont les flux stochastiques $\gamma^i(j,k)$, $(j,k) \in (0;1)^2$ et $\gamma_e^i(l)$, $l \in (0;1)$.

2.3.2 Architecture du module Γ

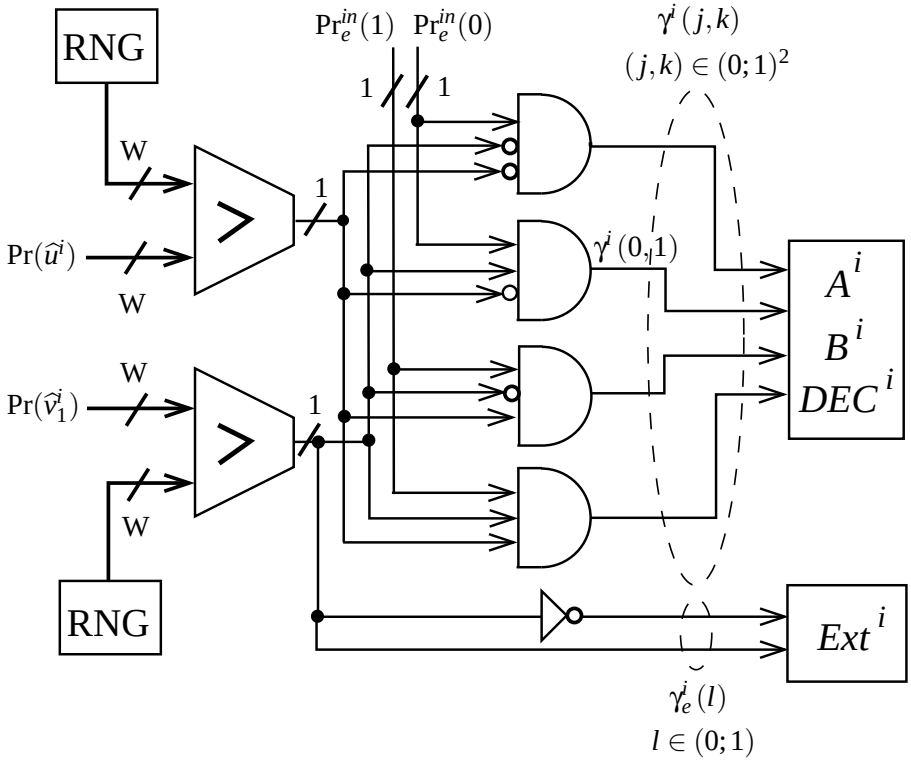


FIGURE 2.5 – Diagramme de bloc du module Γ

L'architecture du module Γ d'une section de treillis est illustrée en figure 2.5. Ce module contient une étape de conversion des probabilités de sortie en parallèle du module *série - parallèle* (figure 2.1) en deux messages stochastiques qui interprètent les informations systématique et de redondance. Afin de transformer les probabilités en des flux stochastiques, le module Γ exige un mécanisme de génération des valeurs aléatoires et des comparateurs. Ce mécanisme se compose de deux générateurs aléatoires de W bits pour convertir deux probabilités *a priori* en deux flux stochastiques. La construction des générateurs aléatoires a été l'objet d'une autre étude approfondie qui sera abordée dans le chapitre 4. Ensuite, ces deux messages stochastiques sont propagés. En exploitant l'expression 2.11, la probabilité $\Pr(\widehat{u}^i|d^i=0)$ est calculée à partir de la probabilité $\Pr(\widehat{u}^i|d^i=1)$ grâce à un inverseur comme le montre la figure 2.5. C'est également le cas pour des probabilités correspondant au bit de redondance et celles extrinsèques entrantes. Par exemple, afin de calculer la métrique de branches $\gamma^i(0,1)$ qui correspond au symbole systématique $j=0$ et au symbole de redondance $k=1$. L'équation est réécrite comme suit :

$$\gamma^i(0,1) = \Pr(\widehat{u}^i|d^i=0).\Pr(\widehat{v}_1^i|y_1^i=1).\Pr_e^{in}(d^i=0|\widehat{v}_2^i) \tag{2.16}$$

En utilisant l'expression 2.11, on déduit :

$$\gamma^i(0,1) = (1-\Pr(\widehat{u}^i)).\Pr(\widehat{v}_1^i).\Pr_e^{in}(d^i=0) \tag{2.17}$$

Ce calcul peut être représenté par une porte logique de trois entrées AND3 comme illustré dans la figure 2.5. Cette étape de calcul est répétée pour calculer les autres flux stochastiques $\gamma^i(j,k)$, $(j,k)\in(0;1)^2$.

En outre, les probabilités $\gamma_e^i(l)$, $l\in(0;1)$ sont calculés seulement à partir de la probabilité *a priori* associée au symbole de redondance et s'expriment comme :

$$\gamma_e^i(1) = \Pr(\widehat{v}^i) \quad \text{et} \quad \gamma_e^i(0) = 1-\Pr(\widehat{v}^i) \tag{2.18}$$

Les flux stochastiques correspondant sont modélisés comme le montre la figure 2.5.

2.4 Modules *A*/*B*

2.4.1 Rôle des modules

Ces deux modules sont utilisés pour calculer récursivement les métriques récurrentes *aller* et *retour*. Ils requièrent les métriques de branche et les métriques récurrentes adjacentes. Considérons par exemple les calculs à la $i^{ème}$ section du treillis (figure 2.6). Sachant que $\alpha^i(s')$ est la métrique d'état *aller* à l'instant (i) associée à l'état s', $\gamma^i(s',s)$ est la métrique de branche entre l'état s' à l'instant (i) et l'état s à l'instant $(i+1)$ du treillis.

Comme montré au chapitre 1, la métrique d'état *aller* $\alpha^{i+1}(s)$ à l'instant $(i+1)$ de l'état s est exprimée comme suit :

$$\alpha^{i+1}(s) = \frac{\sum_{s'=0}^{7}\alpha^i(s')\gamma^i(s',s)}{\sum_{\varepsilon=0}^{7}\sum_{s'=0}^{7}\alpha^i(s')\gamma^i(s',\varepsilon)} \tag{2.19}$$

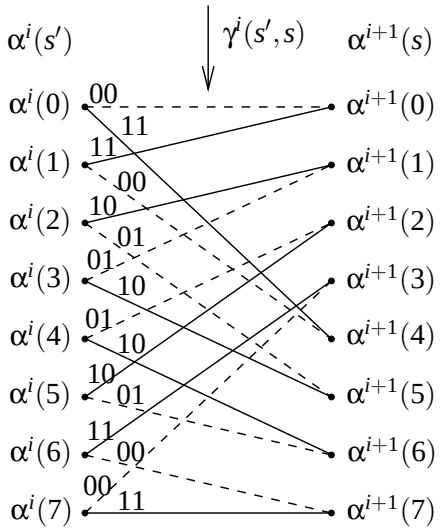


FIGURE 2.6 – Une section du treillis.

L'expression (2.19) signifie que chaque module *A* contient les deux étapes successives de calcul. D'abord le calcul simultané de nouvelles métriques, puis la normalisation de nouvelles métriques reçues comme illustré par le diagramme des blocs (figure 2.7).

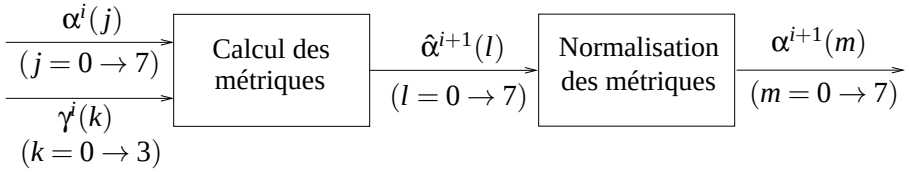


FIGURE 2.7 – Schéma de blocs du calcul des métriques récurrentes *aller* pour la section *i*.

Les entrées du module *A* correspondent à deux vecteurs stochastiques. Le premier vecteur contient huit bits stochastiques à propager $\alpha^i(j)$ $(j = 0 \to 7)$. Chaque bit représente une probabilité d'état *aller* correspondant à l'instant (i). Le second vecteur contient quatre bits stochastiques $\gamma^i(k)$ $(k = 0 \to 3)$ qui sont les sorties du module Γ. De la même façon, les entrées du module *B* sont deux vecteurs stochastiques : $\beta^{i+1}(l)$ $(l = 0 \to 7)$ et $\gamma^i(k)$ $(k = 0 \to 3)$. Les sorties du module *A* et *B* comprennent huit bits interprétant respectivement les nouvelles probabilités d'état à l'instant suivant $\alpha^{i+1}(j)$ $(j = 0 \to 7)$ pour le module *A* et $\beta^i(l)$ $(l = 0 \to 7)$ pour le module *B*. Les deux sous-sections suivantes détaillent les étapes de calcul.

2.4.2 Calcul de nouvelles métriques d'état

Le calcul de nouvelles métriques à l'instant *i* se fait dès que tous les bits stochastiques à l'instant précédent (i) sont disponibles. Chaque $\hat{\alpha}^i(s')$ nécessite une addition et un nombre d'opérations multiplication égal à celui de métriques de branche qui sont connectées à l'état s' du treillis. Par exemple, dans le cas d'un code simple-binaire, la métrique d'état $\hat{\alpha}^{i+1}(1)$ est calculée, avant

l'étape de normalisation, comme suit :

$$\begin{aligned} \hat{\alpha}^{i+1}(1) = &\quad \alpha^i(2).\gamma^i(2,1) + \alpha^i(3).\gamma^i(3,1) \\ \hat{\alpha}^{i+1}(1) = &\quad \alpha^i(2).\gamma^i(2) + \alpha^i(3).\gamma^i(1) \end{aligned} \tag{2.20}$$

La figure 2.8 décrit l'architecture équivalente à ce calcul. Dans ce cas, pour produire une métrique récurrente, deux portes logiques AND2 :1 sont utilisées. Elles permettent de réaliser les multiplications dans le domaine stochastique $\alpha^i(s')\gamma^i(s',s)$. Et afin d'additionner ces deux résultats, un MUX2 :1 par un bit (pseudo-)aléatoire est nécessaire. Ce processus est répété pour trouver toutes les métriques récurrentes.

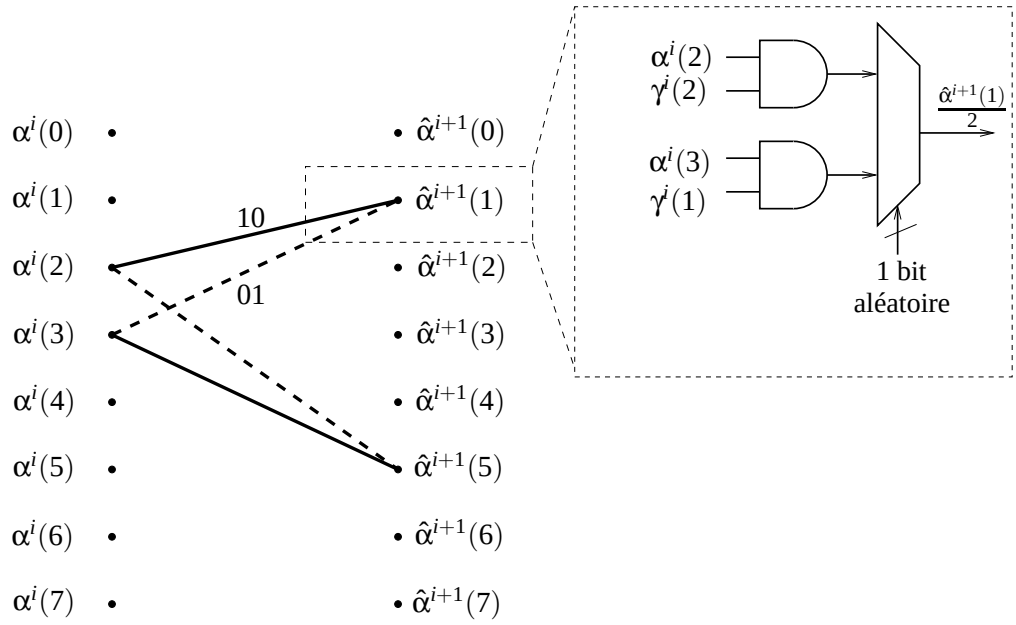


FIGURE 2.8 – Diagramme d'un bloc pour calculer une métrique d'état α.

2.4.3 Normalisation des métriques récurrentes

Dès que toutes les métriques récurrentes sont calculées, elles sont envoyées à un bloc de normalisation illustré en figure 2.7. Ainsi, la normalisation de la métrique d'état 0 à l'instant $(i+1)$ est :

$$\alpha^{i+1}(0) = \frac{\hat{\alpha}^{i+1}(0)}{\sum_{s=0}^{7} \hat{\alpha}^{i+1}(s)} = \frac{\frac{\hat{\alpha}^{i+1}(0)}{8}}{\frac{0+\sum_{s=1}^{7} \hat{\alpha}^{i+1}(s)}{8} + \frac{\hat{\alpha}^{i+1}(0)}{8}} \tag{2.21}$$

Une opération de division représentée par une bascule JK est utilisée pour normaliser $\hat{\alpha}^{i+1}(0)$. Dans ce cas, chaque valeur $\hat{\alpha}^{i+1}(s')$ est mise à l'échelle par un facteur de $1/8$ et un signal 0 est ajouté avant l'opération de division pour être équivalent à l'opération addition de la section 1.3.1.2 du chapitre 1. La valeur $1/8$ peut être convertie en flux stochastique à l'aide d'un vecteur de trois bits (pseudo-)aléatoires. Chaque bit aléatoire est équivalent à une probabilité $1/2$. Un MUX8 :1 est indispensable pour réaliser l'addition des huit bits stochastiques. L'architecture correspondant à la normalisation de la métrique récurrente $\hat{\alpha}^{i+1}(0)$ est présentée en figure 2.9.

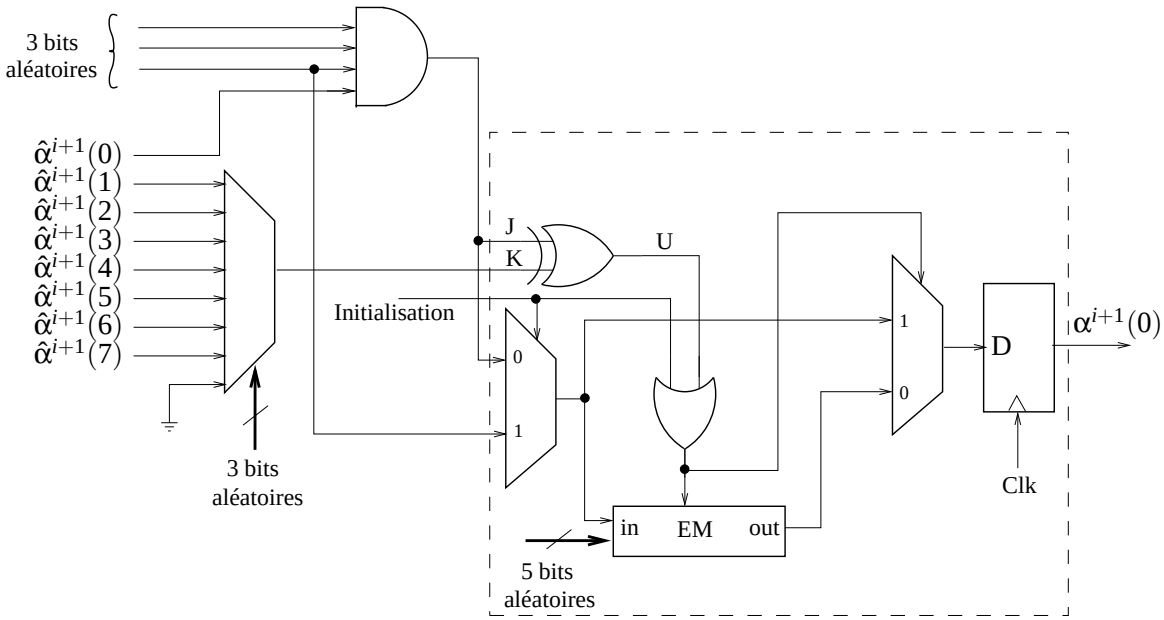


FIGURE 2.9 – Exemple d'une normalisation d'une métrique d'état α.

Afin de contourner le problème de *corrélation*, une structure modifiée de la bascule bénéficiant de l'avantage de la mémoire EM (présentée en section 1.3.2.2 du chapitre 1) est employée. Chaque mémoire EM de 32 bits est associée à chaque état du treillis. Elle requiert un vecteur de 5 bits (pseudo-)aléatoires. La profondeur de la mémoire EM est choisie pour assurer un temps suffisamment long afin d'avoir une confiance sur la probabilité d'entrée à chaque EM [16]. Bien qu'il soit possible d'initialiser les mémoires EMs à partir de l'état "tout à zéro", un état d'initialisation aléatoire de la mémoire EM améliore la convergence d'un décodeur stochastique. C'est pourquoi, le décodeur implémenté contient des mémoires EMs puis sont initialisées avec des bits aléatoires. Durant la phase d'initialisation, les EMs sont initialisées par une série de bits aléatoires qui sont, par exemple, requis pour représenter le facteur d'échelle $1/8$.

Le bit de sortie U de la bascule JK associée à la métrique d'état $\alpha^i(0)$ permet de vérifier quand le flux stochastique résultant tombe dans un état bloqué. S'il n'est pas à l'état bloqué (U=1) un nouveau bit regénératif (J=1) est sélectionné comme le résultat de la normalisation, et la mémoire EM associée est mise à jour avec ce bit regénératif. En revanche, lorsqu'il est dans l'état bloqué (U=0), un bit est choisi par hasard à partir de la mémoire EM correspondante pour le résultat de la normalisation. La sélection aléatoire des bits dans les mémoires EM est assurée par des adresses pseudo-aléatoires qui varient lors de chaque cycle de décodage. En outre, pour augmenter la convergence du décodeur, les bits les plus anciens sont remplacés dans les mémoires par les plus nouveaux bits regénératifs.

Le résultat de la normalisation est finalement transmis au module *Dec* pour évaluer les probabilités *a posteriori*. Ce résultat est également transmis au module *Ext* pour tenir compte des probabilités extrinsèques de sortie.

2.5 Module *Ext*

2.5.1 Rôle du module

Le module *Ext* calcule l'information extrinsèque correspondant au symbole reçu avant sa transmission à l'autre décodeur. Notre code binaire nécessite les métriques récurrentes *aller* et *retour*, ainsi que la probabilité de redondance pour calculer les probabilités des deux valeurs possibles pour le symbole codé $d^i \in \{0;1\}$. Pour chaque $j \in \{0;1\}$:

$$\begin{aligned}\Pr_e^{out}\left(d^i = j|u^i, v_1^i\right) &= \frac{\sum\limits_{(s',s)/d_i(s',s)=b} \alpha^i(s')\,\beta^{i+1}(s)\,\gamma_e^i(s',s)}{\sum\limits_{(s',s)} \alpha^i(s')\,\beta^{i+1}(s)\,\gamma_e^i(s',s)} \\ &= \frac{\sum\limits_{k\in(0\to1)}\sum\limits_{(s',s)\in(0\to7)^2} \alpha^i(s').\beta^{i+1}(s).\Pr\left(\widehat{v}_1^i|y^i=k\right).\delta^{i,j,k}(s',s)}{\sum\limits_{(m,l)\in(0\to1)^2}\sum\limits_{(s',s)\in(0\to7)^2} \alpha^i(s').\beta^{i+1}(s).\Pr\left(\widehat{v}_1^i|y^i=l\right).\delta^{i,m,l}(s',s)}\end{aligned} \quad (2.22)$$

où $\gamma_e^i(s',s)$ est donné dans équation 2.15. La quantité $\delta^{i,j,k}(s',s) = 1$ si et seulement s'il y a une transition entre l'état s' à l'instant (i) et l'état s à l'instant $(i+1)$ correspondant à l'entrée $d^i(s',s) = j$. Sinon, $\delta^{i,j,k}(s',s) = 0$. Les entrées du module *Ext* sont :

- le vecteur de probabilités récurrentes *aller* $\alpha^i(s')(s' = 0 \to 7)$ - sortants du module *A*.
- le vecteur de probabilités récurrentes *retour* $\beta^{i+1}(s)(s = 0 \to 7)$ - sortants du module *B*.
- le vecteur de probabilités de redondance $\gamma^i(k)(k = 0 \to 1)$ - sortants du module Γ.

Les sorties du module *Ext* sont les deux probabilités extrinsèques $\Pr_e^{out}(d^i = j)(j = 0 \to 1)$ qui correspondent aux deux symboles possibles du symbole binaire d^i d'entrée.

2.5.2 Architecture du module *Ext*

A partir de l'équation 2.22, le module *Ext* calcule l'information extrinsèque de sortie. Nous l'avons décomposé en deux sous-modules correspondant à deux étapes de calcul, comme illustré en figure 2.10. Par exemple, la probabilité extrinsèque pour le symbole possible $d^i = 1$ qui est

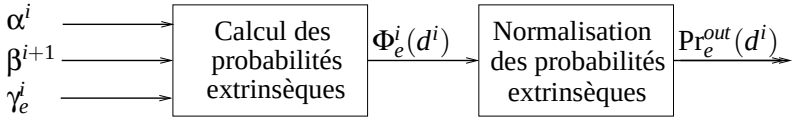


FIGURE 2.10 – Diagramme des blocs pour le calcul de l'information extrinsèque de sortie.

basée sur une section de treillis (dans la figure 2.6), avant l'étape de normalisation, est exprimée par l'équation :

$$\begin{aligned}\Phi_e^i\left(d^i=1\right) = {} & \alpha^i(0).\gamma_e^i(0,4).\beta^{i+1}(4) + \alpha^i(1).\gamma_e^i(1,0).\beta^{i+1}(0) \\ & + \alpha^i(2).\gamma_e^i(2,1).\beta^{i+1}(1) + \alpha^i(3).\gamma_e^i(3,5).\beta^{i+1}(5) \\ & + \alpha^i(4).\gamma_e^i(4,6).\beta^{i+1}(6) + \alpha^i(5).\gamma_e^i(5,2).\beta^{i+1}(2) \\ & + \alpha^i(6).\gamma_e^i(6,3).\beta^{i+1}(3) + \alpha^i(7).\gamma_e^i(7,7).\beta^{i+1}(7)\end{aligned} \quad (2.23)$$

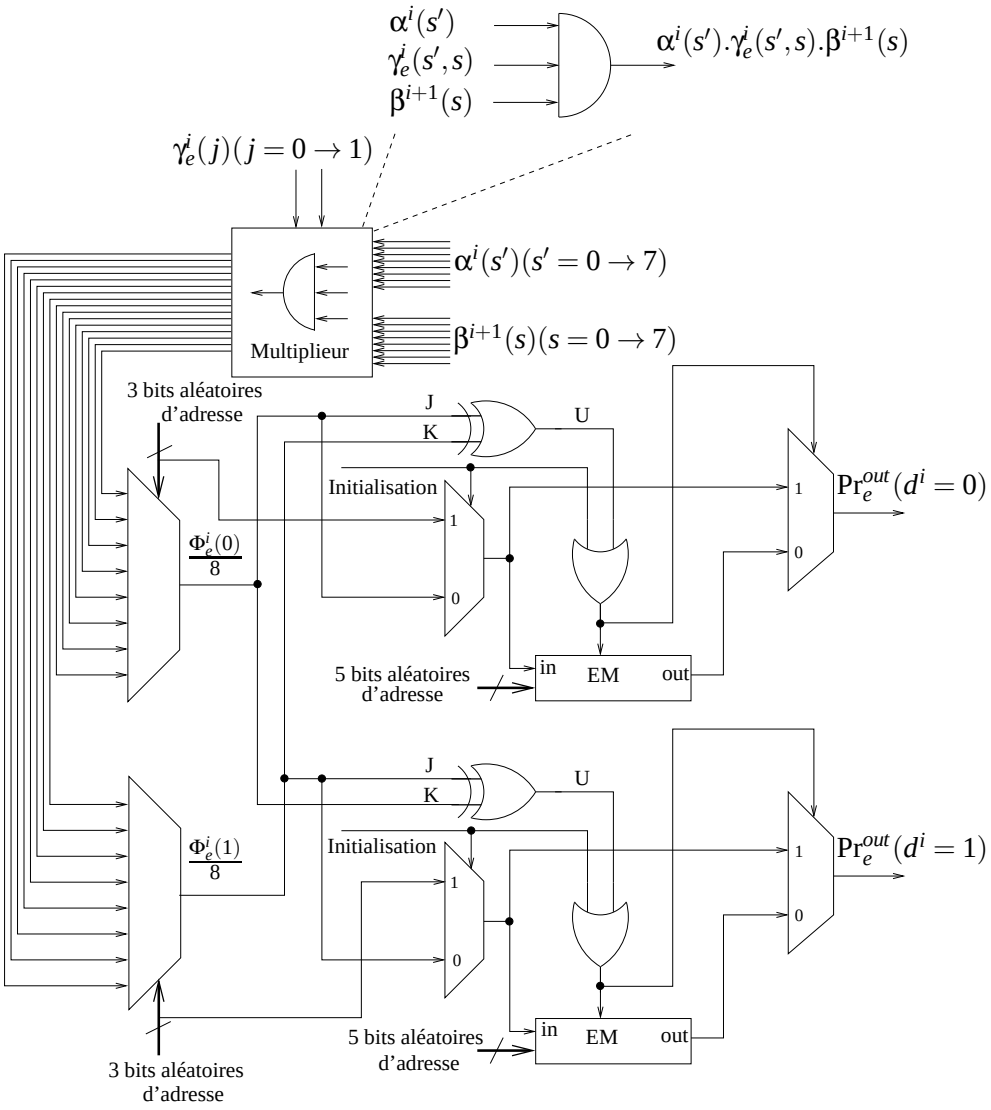


FIGURE 2.11 – Diagramme des blocs du module *Ext*.

Afin d'évaluer cette probabilité, il faut additionner les huit produits. Chaque produit est obtenu par le module *multiplieur* (figure 2.11). Le module *multiplieur* choisit une transition sur le treillis entre l'état s' à l'instant (i) à l'état s à l'instant $(i+1)$ qui est représentée par une ligne continue. Chaque produit est calculé par une multiplication de trois quantités correspond à cette transition : la probabilité récurrente *aller* $\alpha^i(s')$ de l'état s', la probabilité récurrente *retour* $\beta^{i+1}(s)$ de l'état s et la probabilité de redondance entre ces deux états $\gamma_e^i(s',s)$. Chaque produit correspond à une probabilité extrinsèque de branche *a posteriori* et correspond à une porte logique AND à trois entrées. La probabilité extrinsèque du symbole $d^i=0$ est quant à elle représentée par des lignes en pointillés.

Après ce calcul, les probabilités extrinsèques sont normalisées. Au cours de cette étape, un bloc similaire à celui de la figure 2.9 est repris comme le montre le figure 2.11. Le nombre de portes logiques AND est égal au nombre de métriques extrinsèques de branche. De plus, deux

MUX8 :1 assurent les additions stochastiques nécessaires pour obtenir les probabilités extrinsèques des deux symboles possibles $d^i \in \{0,1\}$. Nous regroupons les huit premières probabilités extrinsèques de branche *a posteriori* correspondant au symbole $d^i = 0$, et les huit suivantes correspondant au symbole $d^i = 1$. Ce schéma détaille également la normalisation de deux probabilités extrinsèques, ce qui nécessite deux mémoires de type EM de 32 bits. Dans ce cas, deux vecteurs de 5 bits aléatoires sont nécessaires pour adresser les deux mémoires EM. De plus, deux vecteurs de 3 bits aléatoires sont utilisés pour additionner les flux stochastiques qui représentent les métriques extrinsèques de branche.

2.6 Module *DEC*

2.6.1 Rôle du module

Ce module détermine le symbole le plus vraisemblable associé au symbole d^i. Il calcule les probabilités *a posteriori* correspondant aux deux symboles possibles $b \in (0;1)$ à partir des probabilités récurrentes α^i, β^{i+1} et des probabilités de branche γ^i. Par exemple, la probabilité *a posteriori* du symbole $b = 1$ est donnée par l'expression suivante :

$$\Pr\left(d^i = 1 | u^i, v_1^i\right) = \sum_{(s',s)/d_i(s',s)=1} \alpha^i\left(s'\right) \beta^{i+1}(s) \gamma^i\left(s',s\right) \tag{2.24}$$

La probabilité *a posteriori* du symbole $b = 1$ est la somme des huit probabilités *a posteriori* de branche correspondant à la transition de l'étiquette $d^i y^i = 1y^i$. Chaque probabilité *a posteriori* de branche est associée à une ligne continue sur le treillis. De la même façon, la probabilité *a posteriori* du symbole $b = 0$ est la somme des huit probabilités *a posteriori* de branche correspondant à la transition de l'étiquette $d^i y^i = 0y^i$. Le module *DEC* compare ensuite ces probabilités *a posteriori* pour sélectionner la probabilité la plus grande. La probabilité la plus grande indique que le symbole correspondant est la décision dure du symbole binaire transmis d^i. Les entrées du module *DEC* sont des vecteurs de flux stochastiques qui représentent :

- le vecteur de probabilités récurrentes *aller* $\alpha^i(s')(s' = 0 \rightarrow 7)$
- le vecteur de probabilités récurrentes *retour* $\beta^{i+1}(s)(s = 0 \rightarrow 7)$
- le vecteur de probabilités de branche $\gamma^i(j)(j = 0 \rightarrow 4)$

La sortie du module *DEC* correspond à l'estimation dure du symbole binaire $\widehat{d^i}$.

2.6.2 Solution alternative d'implémentation du module *DEC*

En combinant des expressions 1.9, 1.17, 1.18, 1.19 du chapitre 1, nous pouvons exprimer la probabilité *a posteriori* du symbole $b = 1$ par l'équation suivante :

$$\Pr(d^i = 1 | u^i, v_1^i) = \Pr_e^{out}\left(d^i = 1 | \mathbf{u}, \mathbf{v}_1\right) . \Pr_e^{in}\left(d^i = 1 | \mathbf{u}, \mathbf{v}_2\right) . \Pr(\widehat{u}^i | d^i = 1) \tag{2.25}$$

L'équation 2.25 nous montre qu'il existe une autre solution permettant de trouver facilement la probabilité *a posteriori* du symbole $b = 1$. Ce calcul requiert la connaissance des probabilités extrinsèques sortantes. Dans ce cas, les entrées du module *DEC* sont des vecteurs de flux stochastiques représentant :

- Le vecteur de probabilités extrinsèques entrantes fourni par l'autre décodeur *SISO* élémentaire : $\Pr_e^{in}\left(d^i = j|\mathbf{u},\mathbf{v}_2\right)$, $j \in (0;1)$.
- Le vecteur de probabilités extrinsèques sortantes du module Ext : $\Pr_e^{out}\left(d^i = k|\mathbf{u},\mathbf{v}_1\right)$, $k \in (0;1)$.
- Le vecteur de probabilités *a priori* correspondant au symbole systématique : $\Pr(\widehat{u}^i|d^i = b)$, $b \in (0;1)$.

Une fois les probabilités extrinsèques sortantes disponibles à l'entrée du module *DEC*, la probabilité *a posteriori* du symbole $b = 1$ est alors évaluée par une seule multiplication de trois probabilités. De la même façon, la probabilité *a posteriori* du symbole $b = 0$ est la multiplication de trois probabilités.

Un majeur avantage de cette solution alternative en comparaison avec la solution introduite par l'expression 2.24 est qu'elle requiert une seule porte logique de trois entrées AND3 associée à la probabilité *a posteriori* du symbole $b = 1$, lorsque le calcul classique présenté par l'expression 2.24 exige huit portes logiques de trois entrées AND3 différentes. Dans ce cas, aucun additionneur n'est requis. En conséquence, cette solution permet de diminuer le nombre de bits aléatoires et de MUX8 :1, alors, simplifier la complexité calculatoire.

Néanmoins, l'utilisation des probabilités extrinsèques sortantes afin d'évaluer les probabilités *a posteriori* apporte un inconvénient au niveau du nombre de cycles de décodage DC_{max} : la solution proposée ci-dessus nécessite plus du nombre de cycles de décodage pour obtenir une même performance de décodage par rapport à la solution classique. Cet inconvénient se déroule en raison d'association de la mémoire EM au calcul de la probabilité extrinsèque du symbole $b = 1$. Au début, la probabilité extrinsèque initiale de ce symbole est inconnue, et elle est initialisée à une probabilité de 0,5. Cela signifie que le nombre de bits '1' et de '0' dans cette EM sont égaux. Dans le temps, la mémoire EM doit parvenir à un nombre de cycles DC_{conv} quelconque pour se converger. Durant l'intervalle $[0;DC_{conv}]$, la convergence de la probabilité *a posteriori* du symbole $b = 1$ est influencée non seulement par la convergence des mémoires EMs associées aux métriques récurrentes *aller/retour* mais aussi par la convergence de la mémoire EM associée à la probabilité extrinsèque de ce symbole. Dès lors, le module *DEC* requiert plus de cycles pour converger. Dans la section suivante, à l'aide des simulations fonctionnelles, nous démontrons que le décodage stochastique exploitant cette solution est dégradé par rapport à celui exploitant la solution classique proposée par l'expression 2.24.

2.6.3 Diagramme de bloc du module *DEC*

Le diagramme du module *DEC* selon l'expression classique 2.24 est illustré dans la figure 2.12. Il est composé d'un *multiplieur* qui effectue des opérations multiplication de flux stochastiques représentant des métriques récurrentes *aller* $\alpha^i(s')$, *retour* $\beta^{i+1}(s)$ et des métriques de branche $\gamma^i(s',s)$. Le multiplieur produit des flux stochastiques de métriques de branche *a posteriori* $\alpha^i(s').\gamma^i(s',s).\beta^{i+1}(s)$. Chaque opération multiplication est associée à une porte logique AND de trois entrées. Le nombre de portes logiques AND de trois entrées nécessaires est égal à celui de branches dans une section. Puis, le résultat sortant du multiplieur est décomposé afin de fournir les deux probabilités *a posteriori* $\Pr^i(j)(j \in (0,1))$, en utilisant les deux MUX8 :1.

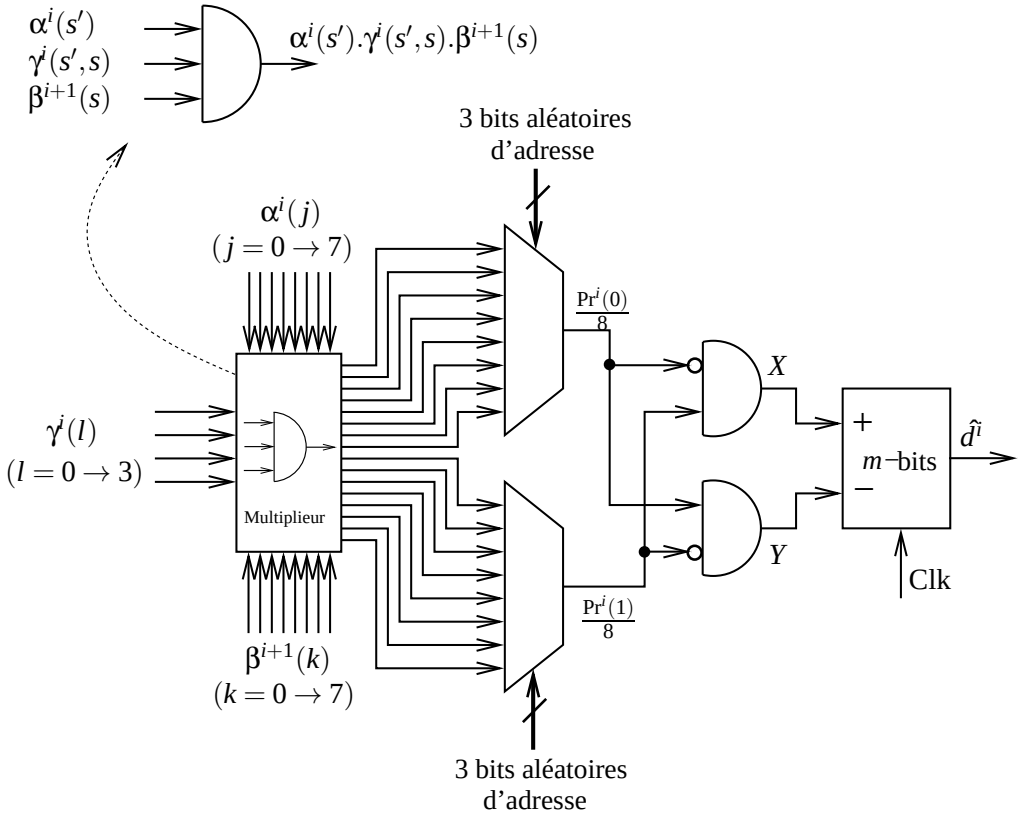


FIGURE 2.12 – Diagramme des blocs du module *DEC*.

Dans le cas d'un code binaire, nous proposons d'utiliser un compteur up/down saturé de m bits pour assurer la prise de décision. Le compteur peut être initialisé pour contenir la valeur 0. Il est incrémenté d'une unité lorsque le bit sortant de la probabilité APP $\Pr^i(1)$ correspondant au symbole estimé $b=1$ est égal à "1" et celui de la probabilité APP $\Pr^i(0)$ correspondant au symbole estimé $(b=0)$ est égal à "0". Cette incrémentation a lieu si le compteur n'a pas atteint à sa limite maximum $(+2^{m-1}-1)$. De manière similaire, si le bit sortant de la probabilité APP $\Pr^i(1)$ est "0" et celui de la probabilité APP $\Pr^i(0)$ est "1", le compteur décroît une unité jusqu'à sa limite minimum (-2^{m-1}). Dans les autres cas, le contenu du compteur n'évolue pas.

Parallèlement, le décodeur stochastique vérifie si le nombre maximum de cycles DC_{max} est atteint pour arrêter le processus de décodage. Le signe de la valeur dans le compteur est utilisé pour la décision du symbole le plus vraisemblable. Si le bit de poids fort (en anglais *Most Significant Bit* - MSB) est égal à "1", le symbole émis est $\hat{d^i}=1$. Sinon $\hat{d^i}=0$. Ce module nécessite 2 vecteurs de 3 bits aléatoires afin d'additionner les deux probabilités *a posteriori*.

2.7 Performance de correction d'erreurs de décodeurs stochastiques

A partir de l'architecture détaillée dans les sections précédentes, nous présentons les performances de différentes versions de décodeurs pour des codes convolutifs binaires et des turbocodes

convolutifs. Puis, nous faisons une estimation de la complexité matérielle de l'architecture proposée en termes d'éléments logiques. Enfin, nous aboutissons à une expression du débit d'un décodeur stochastique.

2.7.1 Performance du décodage convolutif stochastique

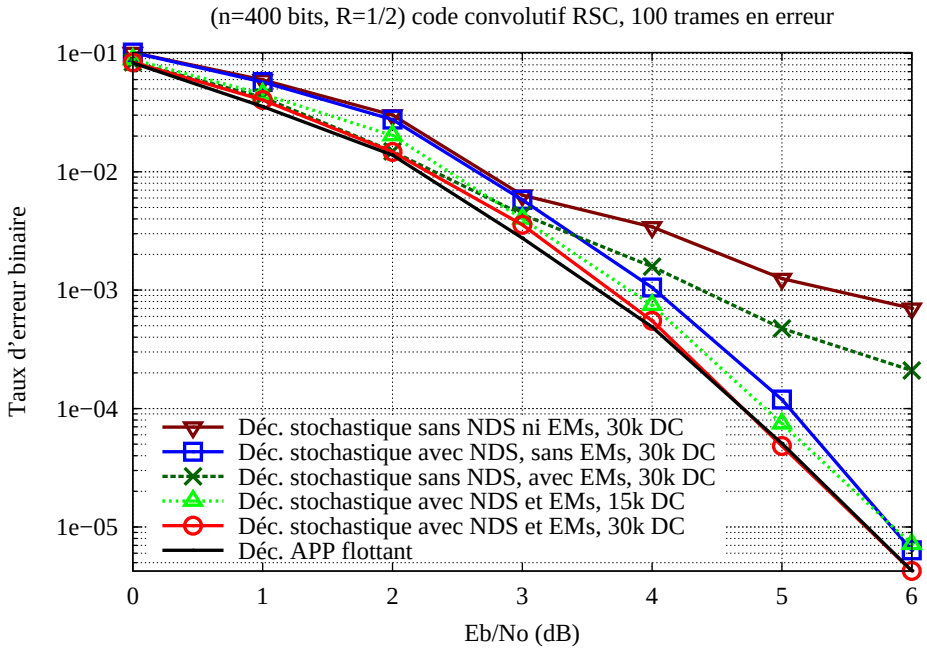


FIGURE 2.13 – Performance du décodage stochastique pour un code convolutif RSC (k=200, $R = 1/2$).

La figure 2.13 nous montre les performances de décodage stochastique de codes convolutifs RSC de longueur $k = 200$ et de rendement $R = 1/2$ en termes de taux d'erreur binaire, pour différents cas d'optimisation. Des probabilités démodulées reçues à partir du canal sont quantifiées suivant le format (p=7,q=4). Les sorties des mémoires ROM sont représentées par $(W = 7)$ bits. Les compteurs du module *DEC* sont des compteurs up/down saturés sur 3 bits. Les résultats sont obtenus en fonction de deux solutions pour éviter le problème de *corrélation*. Dans le cas d'une transmission BPSK sur le canal AWGN, le facteur $\psi(\sigma)$ de *NDS* est égal à 1.0 pour tous les σ. Le coefficient $\Omega = \sigma_0^2 = 1.0$.

Les courbes sont obtenues par des simulations fonctionnelles à partir de langage C++. La courbe noire représente la performance de décodage en utilisant l'algorithme MAP de référence dans le domaine flottant. La courbe marron décrit un décodage sans approche *NDS* ni des mémoires EM. La courbe verte foncée applique des mémoires EM et permet une meilleure performance par rapport au cas sans mémoires EM. La courbe bleu améliore significativement la performance en exploitant l'approche *NDS*. La courbe bénéficie des deux techniques *NDS* et EMs, cependant pour le nombre de DC égal à 15k.

Nous pouvons constater que le décodeur utilisant l'approche NDS et des mémoires EMs (la

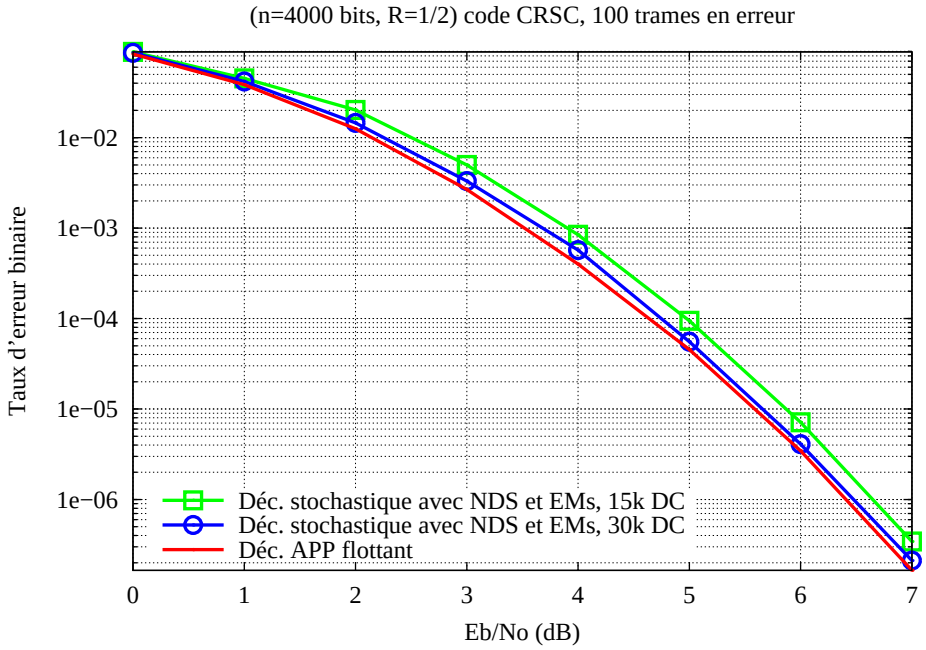


FIGURE 2.14 – Performance du décodage stochastique pour un code convolutif RSC (k=2000, $R = 1/2$).

courbe rouge) aboutit à des performances similaires à celles d'un décodeur APP conventionnel qui exploite l'algorithme MAP dans le domaine flottant. La figure 2.13 illustre également l'impact sur la performance de décodage stochastique lorsque le nombre de cycles de décodage change de 15k DC à 30k DC. Basé sur les performances de décodage obtenues, nous observons que le nombre de cycles de décodage maximum $DC_{max} = 30k$ est suffisant pour obtenir une performance égale à l'optimale en virgule flottante.

La figure 2.14 décrit également la performance de décodage pour des codes convolutifs de taille plus longue (k=2000, $R = 1/2$). La performance de décodage démontre que l'extension du décodage stochastique pour des codes est réalisable en pratique.

2.7.2 Performance du turbo-décodeur convolutif stochastique

Les performance du turbo-décodeur stochastique est donnée dans la figure 2.15 par des simulations fonctionnelles (langage C++). Un turbocode de longueur $k = 200$ et de rendement $R = 1/3$ est considéré. Le format de quantification est de la forme (p=7, q=4). La longueur des bits aléatoires pour une probabilité est de $W = 7$ bits. La technique basée sur les EMs est utilisée pour une longueur de 32 bits. Le coefficient NDS est choisi en basant sur la meilleure performance de décodage. Dans le cadre d'un décodage stochastique de turbocode basé sur l'algorithme MAP et pour une transmission BPSK sur le canal AWGN, le coefficient Ω est égal à $\sigma_0^2 = 1.5$. Le résultat de décodage stochastique est comparé avec le décodage conventionnel utilisant l'algorithme SubMAP dans le domaine flottant avec 6 itérations ce qui est souvent utilisé dans des architectures d'implémentation matérielle de décodage en pratique.

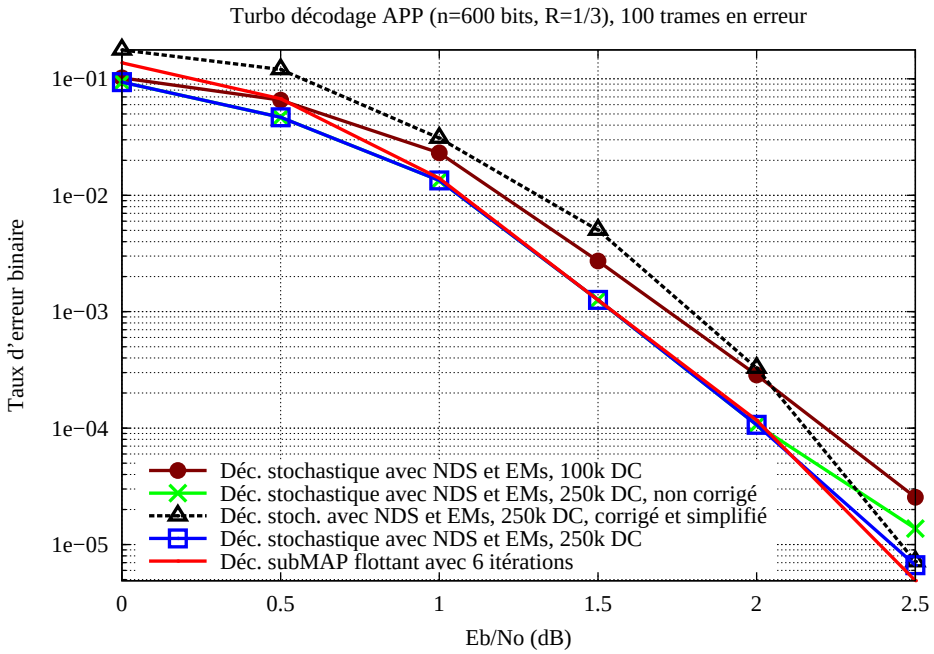


FIGURE 2.15 – Performance du décodage stochastique pour des turbocodes convolutifs (k=200, n=600, $R = 1/3$).

La courbe rouge décrit les performances du décodage utilisant l'algorithme SubMAP en virgule flottante. Les autres courbes présentent les performances des décodeurs stochastiques. La courbe verte, dite *version non corrigée*, correspond aux performances lorsque le *facteur de correction* $\psi(\sigma)$ est égal à 1.0 pour toutes les différentes valeurs de σ. Pour un grand RSB (2,5dB), la probabilité est très proche de 0 et 1. C'est pourquoi, le facteur $\psi(\sigma)$ doit varier en fonction du RSB afin de différentier les probabilités associées à 0 et à 1. La courbe bleu est obtenue pour $\psi(\sigma)$ égal à 1.0 pour RSB = (0.0dB $\rightarrow$ 2.0dB) et $\psi(\sigma) = 1.2$ pour RSB $>$ 2dB. Basé sur notre observation, lorsque la séquence d'information est plus élevée $(k > 200)$, la valeur du facteur $\psi(\sigma)$ nécessaire dans le décodage stochastique des turbocodes est aussi élevée. Ainsi, le turbo-décodeur stochastique a des performances similaires à celles d'un décodage SubMAP. La courbe noire, version *corrigée et simplifiée*, correspond à la performance du turbo-décodeur stochastique utilisant une modification du facteur $\psi(\sigma)$ pour les différents RSB et la simplification du module *DEC*. Dans ce cas, le turbo-décodeur stochastique introduit une perte de $0,15$dB de performance à 10^{-4} par rapport à celui en virgule flottante de référence. Cela justifie que la solution alternative d'implémentation du module *DEC* produit une perte de performance en comparaison avec la solution classique qui a été présentée dans la section précédente.

Cependant, pour obtenir une bonne performance de décodage stochastique, le nombre de cycles maximum DC_{max} doit être supérieur ou égal à 250k. Ce qui est plus important qu'un décodage stochastique des codes convolutifs. Dès lors, des solutions doivent être proposées afin de réduire le nombre de DC.

2.7.3 Complexité matérielle d'un décodeur stochastique SISO

Dans cette sous-section, nous allons estimer la complexité d'un décodeur stochastique basé sur l'algorithme MAP. Les résultats de cette estimation sont résumés dans le tableau 2.1. La complexité d'un décodeur stochastique est donnée en termes d'éléments logiques, de bits aléatoires et de bascules de type D-FlipFlop.

Pour un décodeur stochastique, la complexité matérielle peut s'exprimer à partir de ressources élémentaires. C'est pourquoi, les comparateurs du module Γ sont décomposés en équivalent de portes logiques élémentaires. Un comparateur sur 8 bits requiert 7* (2* AND2 +1 OR2)+ 1 AND2. Et un comparateur 1 bit de signe comprend 2*AND2 + 1 OR2. De plus, chaque section du décodeur nécessite 20 multiplexeurs à huit entrées pour les opérations de type d'addition, et un compteur de 3 bits. En fait, les modules *Ext* et *DEC* possèdent des opérateurs AND3. Chacun de ces opérateurs peut être décomposé en deux portes logiques AND2.

La complexité matérielle d'une section de décodeur stochastique binaire doit être comparée avec celle d'un décodeur conventionnel SubMAP en virgule fixe. Ainsi, le décodeur SubMAP présenté par G.Montorsi dans [90] est considéré. Pour un décodage subMAP, les symboles reçus sont quantifiés sur 5 bits, tandis que l'information extrinsèque et les métriques récurrentes sont toutes quantifiées sur 7 bits. Afin de comparer avec un décodeur SubMAP conventionnel, il faut considérer trois parties principales : traitement, mémoires, et contrôle. Des restrictions importantes des décodeurs sont la taille volumineuse des mémoires nécessaires pour les métriques récurrentes et les conflits d'accès à ces mémoires. Afin de résoudre ce problème, le principe des fenêtres glissantes (*sliding windows* en anglais) a été proposé et implémenté avec succès pour un décodeur SubMAP. Dans cette approche, chaque trame reçue est divisée en plusieurs sous-trames. Chaque sous-trame correspond à une fenêtre glissante. Afin de résoudre ces contraintes d'initialisation, le décodeur doit considérer des coûts supplémentaires en termes de ressources et de latence. Par ailleurs, pour un décodeur stochastique, des générateurs de valeurs aléatoires sont nécessaires pour produire des bits aléatoires. Le nombre de bits aléatoires peut sembler élevé comme le montre le tableau 2.1. Fort heureusement, ils peuvent être largement partagés entre les différents modules sans impact sur les performances TEB.

C'est pourquoi, une comparaison directe du coût entre deux décodeurs est seulement efficace

Module	Ressources matérielles élémentaires							Bits aléatoires	
	AND2	OR2	XOR2	MUX2 :1	MUX8 :1	compteur	D-FF	7 bits	1 bit
Γ	42	14						2	
A/B	40	8	8	24	8		264		96
Ext	32	2	2	4	2		64		16
Dec	36	1			2	1			6
Total	190	33	18	52	20	1	592	2	214

TABLE 2.1 – Complexité d'une section de décodeur stochastique SISO binaire de 8 états avec des multiplexeurs pour les opérations addition.

pour une unité de traitement d'une section de treillis. Dans ce contexte, la comparaison se repose sur des implémentations sur FPGA. La complexité d'un décodeur SubMAP doit être comparée avec celle du tableau 2.1. Pour ce faire, on considère que chaque LUT (Look-Up Table) est assignée à une unité de ressource matérielle élémentaire comme celles données dans le tableau 2.1. Dans ce cas, un décodeur SubMAP en virgule fixe et un décodeur stochastique nécessitent respectivement 633 et 313 LUT. De plus, en termes de bascules, le nombre pour une section peut être diminué de 1398 à 592 si la version stochastique est considérée. Alors, la complexité d'un décodeur stochastique est comparable avec celle d'un décodeur SubMAP en virgule fixe. Pourtant, il faut rappeler que les chiffres de la complexité d'un turbo-décodeur stochastique n'ont pas encore tenu compte la complexité des bits aléatoires. Cela signifie que'un décodage stochastique exige une complexité matérielle raisonnable pour des turbocodes.

2.7.4 Estimation du débit d'un turbo-décodeur stochastique

Nous considérons que notre architecture est synchronisée et contrôlée par une horloge caractérisée par une fréquence f_{clk} (MHz). Le débit d'une architecture D_s est conventionnellement définie comme suit [53] :

$$D_s = f_{clk}.\frac{n_s}{n_c} \quad \text{[Mbits/s]} \tag{2.26}$$

où n_c est le nombre de cycles requis pour produire n_s bits. Dans une architecture de décodage stochastique, n_s est le nombre de symboles sortant du décodeur. En outre, le processus de réception des probabilités en entrée et de production des symboles décodés en sortie du décodeur est tout en parallèle comme indiqué en section 2.2.1. Ainsi, n_c est le nombre de cycles de décodage maximum nécessaire pour chaque décodeur, $n_c = DC_{max}$. C'est pourquoi, l'expression 2.26 est reformulée :

$$D_s = f_{clk}.\frac{k}{DC_{max}} \quad \text{[Mbits/s]} \tag{2.27}$$

Par exemple, si nous supposons que la fréquence de fonctionnement du turbo-décodeur stochastique ($k = 200$, $R = 1/3$) est égale à 500MHz (raisonnable pour une technologie ASIC 65 nm), alors, le débit de ce décodeur est donné par :

$$D_s = 500.\frac{200}{250.000} = 0,4 \quad \text{[Mbits/s]} \tag{2.28}$$

Nous observons que le débit de notre architecture est assez petit ce qui résulte en une faible efficacité architecturale. C'est pourquoi, des techniques fortes sont exigées pour augmenter le débit de décodage. Une des solutions est de transmettre plus de données ($k > 200$ bits). Cependant, lorsque k s'augmente, le treillis ainsi que l'entrelacement deviennent plus longs. Ce problème requiert plus de temps pour échanger les messages entre les différentes sections et entre deux décodeurs SISO pour que les deux décodeurs convergent. Le nombre de DC nécessaire doit s'accroître afin d'atteindre une bonne performance de décodage. Par conséquent, le débit correspondant obtenu n'est sûrement pas élevé.

Par ailleurs, nous constatons que le nombre de cycles $DC_{max} = 250$k est trop grand. Dès lors, une autre solution potentielle est de réduire le nombre de cycles de décodage maximum DC_{max}.

2.8 Conclusion

Dans ce chapitre, nous avons décrit l'architecture d'un turbo-décodeur stochastique conventionnel. La construction des différents modules de l'architecture est détaillée. Les performances de décodage stochastique pour des codes convolutifs et des turbocodes convolutifs sont ensuite fournies. Après l'application des techniques NDS et EMs pour éliminer la corrélation des flux stochastiques, nous obtenons des performances pour le turbo-décodeur stochastique similaires à celles de décodage classique SubMAP avec 6 itérations en virgule flottante.

Par ailleurs, une estimation de la complexité matérielle d'une section de l'architecture d'un décodeur stochastique APP est donnée. Une comparaison de la complexité matérielle avec un décodeur utilisant l'algorithme SubMAP est présentée. Les résultats de comparaison montrent que le décodeur stochastique APP a un coût matériel raisonnable (même plus petit) par rapport à celui d'un décodeur SubMAP conventionnel en virgule fixe. Enfin, ce chapitre fournit une estimation de débit d'un turbo-décodeur stochastique conventionnel. Un exemple à une fréquence de fonctionnement élevée démontre que notre architecture de turbo-décodeur stochastique résulte en un faible débit. Cela est dû à un nombre de cycles de décodage maximum DC_{max} trop élevé. En conséquence, de nouvelles techniques doivent être considérées afin de réduire le nombre de DC_{max} et ainsi obtenir un meilleur débit de décodage.

CHAPITRE 3

Conception de turbo-décodeurs stochastiques à haut débit

Sommaire

3.1 Introduction

Dans le chapitre précédent, nous avons montré qu'un turbo décodage stochastique pouvait être mis en œuvre. Le décodeur proposé atteint une performance similaire à la performance d'un turbo-décodeur utilisant l'algorithme SubMAP en virgule fixe pour 6 itérations. Cependant, ce type de décodeur comprend plusieurs opérations de type addition stochastique représentées par des multiplexeurs de grande taille (20*MUX8 :1+52*MUX2 :1 par section). Comme déjà expliqué au chapitre 1, ces additions requièrent un grand nombre de cycles d'éxécution pour obtenir un bonne précision. En conséquence, le décodeur nécessite un grand nombre de cycles DC_{max} pour converger, ce qui dégrade le débit de décodage. Par ailleurs, dans le chapitre 1, des résultats notables d'implémentation de turbocodes sur une cible ASIC ont également montré que l'utilisation de plusieurs décodeurs SISO en parallèle pouvait permettre une montée du débit.

Ce chapitre s'organise comme suit. Dans un premier temps, nous allons présenter une nouvelle technique. Cette technique concerne un domaine de décodage stochastique - *le domaine exponentiel* [91]. A partir de fonctions exponentielles, des opérations stochastiques addition à plusieurs entrées sont remplacées par des opérations stochastiques simples dans le domaine exponentiel. Nous allons démontrer que cette approche permet de réduire significativement le nombre de cycles DCs. Ensuite, nous aborderons le parallélisme des décodeurs stochastiques SISO. Dans ce chapitre, nous allons introduire une version innovante de parallélisme de décodage stochastique, à savoir, la version *multi-flux*. Pour finir, nous présenterons la contribution conjointe de ces deux techniques pour la conception d'une nouvelle architecture de turbo-décodeur stochastique. Des analyses du point de vue de la performance obtenue ainsi que du débit et de la complexité matérielle seront enfin discutées.

3.2 Décodage stochastique dans le domaine exponentiel

D'abord, ce chapitre détaille le traitement des additions stochastiques dans le domaine exponentiel en utilisant des transformations exponentielles et la propagation de flux stochastiques dans le domaine exponentiel. Il présente également une méthode de récupération des flux stochastiques du domaine stochastique exponentiel au domaine conventionnel. Le traitement entre les deux domaines conventionnel - exponentiel est implémenté à l'aide de fonctions de conversion : exponentielle - logarithmique. Ensuite, nous présentons l'architecture du turbo-décodeur stochastique dans le domaine exponentiel. Enfin, l'intérêt d'approche exponentielle au turbo décodage stochastique en termes de haut débit sera démontré par des résultats obtenus par les simulations fonctionnelles en langage C++.

3.2.1 Conversion en domaine exponentiel

Comme déjà mentionné dans le premier chapitre, les turbo-décodeurs stochastiques exigent un grand nombre de multiplexeurs à plusieurs entrées. Ces multiplexeurs représentent des opérations d'addition stochastique. La séquence de Bernoulli en sortie nécessite une longue observation pour obtenir une bonne précision. Ce n'est pas le cas pour des portes logiques AND dont le flux stochastique de sortie dépend à tout instant de la contribution immédiate de tous les flux entrants. C'est pourquoi nous proposons une solution alternative reposant sur la conversion dans le domaine exponentiel où les multiplexeurs à multiples entrées sont remplacés par des opérateurs stochastiques simples.

L'idée de traitement des flux stochastiques $x_i (i = 1 \rightarrow N)$ dans le domaine exponentiel a été introduite par Jane *et al.* [91]. Cette approche considère un traitement de l'opération **F** dans le domaine exponentiel grâce à des fonctions $\exp(-x)$. Le choix de la fonction $\exp(-x)$ au lieu de $\exp(x)$ vise à s'assurer que le résultat en sortie de cette fonction puisse être représenté par un flux stochastique dans l'intervalle $]0;1[$.

Les valeurs de sortie des modules $\exp(-x)$ sont ensuite traitées par une opération simple **G**. La récupération du flux associé au résultat dans le domaine stochastique conventionnel repose sur l'utilisation d'une fonction $-\log(y)$. Par exemple, si **F** est une opération addition, alors **G** est une opération multiplication. Cette dernière opération peut être modélisée par une porte logique AND à multiples entrées pour une représentation stochastique. Dès lors, aucun multiplexeur à entrées multiples n'est nécessaire (figure 3.1).

3.2.2 Transformation exponentielle

Lorsqu'un flux stochastique peut être évalué dans le domaine exponentiel, la somme de N flux stochastiques peut également être représentée dans le domaine exponentiel par une porte logique AND à N entrées. Le circuit correspondant est décrit en figure 3.2. Chaque module exponentiel est associé à une des entrées de l'opération addition.

$$\prod_{i=1}^{i=N} e^{-x_i} = \exp\left(-\sum_{i=1}^{i=N} x_i\right) \tag{3.1}$$

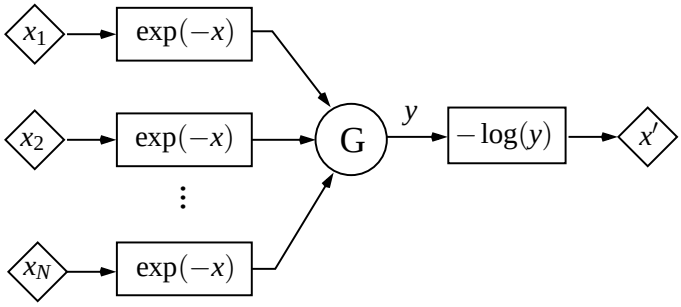


FIGURE 3.1 – Principe du passage dans domaine stochastique exponentiel-conventionnel de l'opération d'addition à multiples entrées.

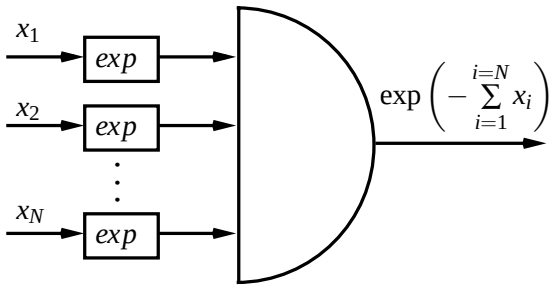


FIGURE 3.2 – Représentation d'addition stochastique dans le domaine exponentiel.

En pratique, cette fonction exponentielle peut être approchée avec seuls quelques premiers ordres d'une série de Tayor. Dans [91], les auteurs ont décrit les premier, deuxième et troisième ordres (l'expression 3.2), qui sont implémentés dans la figure 3.3.

$$\begin{aligned} &\exp(-x) \approx 1-x;\ \ \exp(-x) \approx 1-x+\frac{x^2}{2!} \\ &\exp(-x) \approx 1-x+\frac{x^2}{2!}-\frac{x^3}{3!} \end{aligned} \tag{3.2}$$

Des bascules de type D-FF sont nécessaires pour ralentir le flux stochastique afin de permettre un rebouclage (une multiplication stochastique avec lui-même). Les auteurs ont montré que ces transformations n'étaient pas exactement évaluées. Elles introduisent des erreurs dues aux corrélations successives. Fort heureusement, ils ont également démontré que la précision de cette approximation ne dépendait pas du nombre de probabilités à additionner.

3.2.3 Transformation logarithmique

Dans [91], le résultat de la transformation exponentielle était suffisant pour le traitement des données stochastiques. Néanmoins, dans des architectures de turbo-décodeurs stochastiques, le résultat des opérations addition est ensuite utilisé dans d'autres modules. C'est pourquoi, le flux

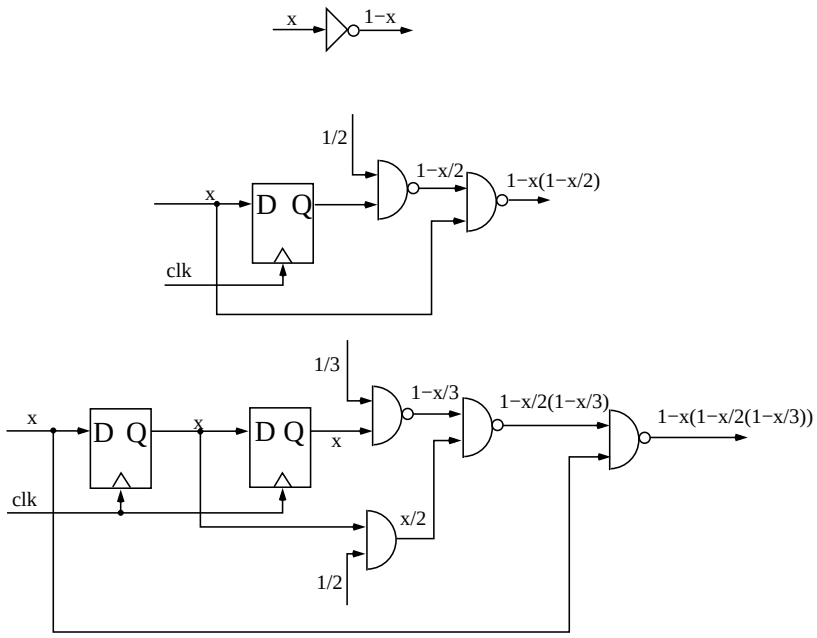


FIGURE 3.3 – Circuits stochastiques à différents ordres du module exponentiel.

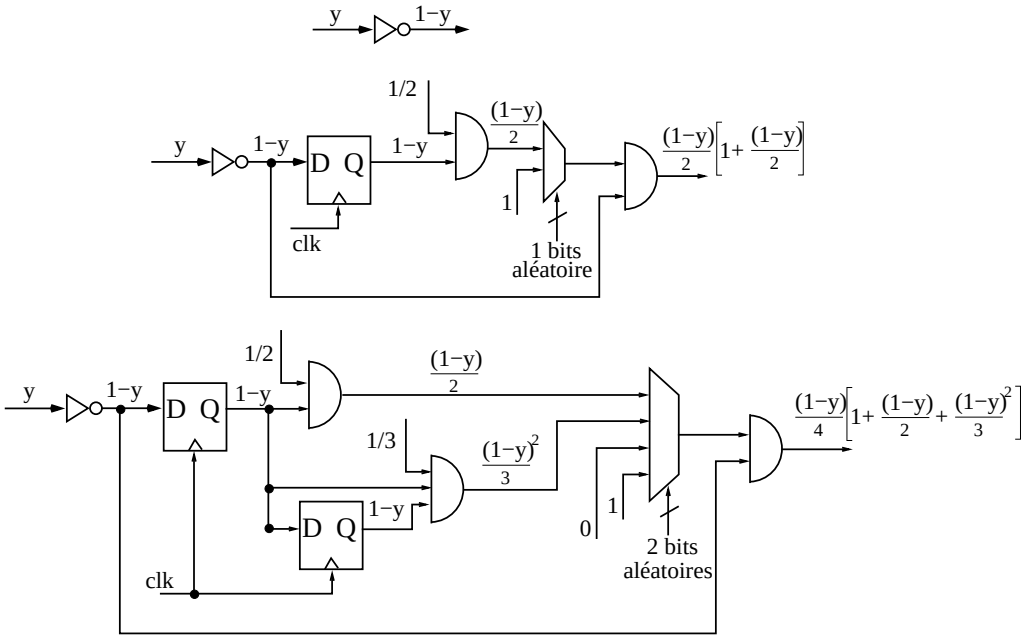


FIGURE 3.4 – Circuits stochastiques correspondant à différents ordres du module logarithmique.

exponentiel doit être converti en un flux conventionnel qui correspond au résultat d'additions stochastiques de N flux. Une fonction logarithmique $-\log(y)$ est donc indispensable pour accomplir cette transformation. Parmi les séries de Taylor détaillée en [92], l'extension décrite par l'équation

3.3 (avec $0 < y \leq 2$) est considérée :

$$-\log(y) = (1-y) - \frac{(1-y)^2}{2} + \frac{(1-y)^3}{3} - ... \tag{3.3}$$

y est un flux stochastique qui représente une probabilité. La condition de cette extension ($0 < y \leq 2$) est donc toujours assurée. Cette fonction est choisie en raison de sa faible complexité matérielle par rapport à d'autres séries. Les architectures matérielles du premier, deuxième et troisième ordre de cette série sont illustrées dans la figure 3.4.

3.2.4 Quel ordre pour les transformations exponentielle et logarithmique ?

Cette sous-section nous montre un principe de choisir les ordres adéquats pour les modules exponentiels et logarithmiques. Le passage dans le domaine exponentiel doit respecter une condition de précision : *une probabilité après le traitement dans le domaine exponentiel doit être récupérée de telle sorte qu'elle soit identique à (ou la plus proche de) la probabilité initiale*. Pour ce qui suit, u_e et u_l définissent respectivement les ordres des modules exponentiel et logarithmique. La condition de précision nécessite une connaissance du couple (u_e, u_l).

Dans le cas simple, nous supposons une architecture d'addition stochastique entre deux probabilités p_1 et p_2 qui sont respectivement représentées par deux flux stochastiques x_1 et x_2. Le modèle exponentiel qui remplace l'opération d'addition stochastique de ces deux probabilités est illustré par la figure 3.5. Pour le traitement dans le domaine exponentiel, nous considérons deux exemples d'ordres des modules exponentiels et logarithmiques : ($u_e = 1$, $u_l = 1$) et ($u_e = 2$, $u_l = 1$).

Pour le premier exemple ($u_e = 1$, $u_l = 1$), nous obtenons :

$$x_s = 1 - (1-x_1)(1-x_2) = (x_1 + x_2) - x_1.x_2 \tag{3.4}$$

La deuxième exemple ($u_e = 2$, $u_l = 1$) nous donne :

$$x_s = 1 - (1 - x_1 + \frac{x_1^2}{2})(1 - x_2 + \frac{x_2^2}{2}) = (x_1 + x_2) + \frac{x_1.x_2}{2}\left(x_1 + x_2 - \frac{x_1.x_2}{2}\right) \tag{3.5}$$

Les deux exemples nous montrent que la probabilité $x_1.x_2$ peut, dans certains cas, être négligeable. Et l'opérateur "+" entre les deux probabilités x_1 et x_2, quant à lui, sert à additionner ces deux flux stochastiques dans le domaine exponentiel. Cette caractéristique peut être élargie pour une addition de plusieurs entrées. En outre, nous constatons également qu'une meilleure approximation peut être obtenue lorsque l'ordre des modules exponentiels deviennent plus grands. C'est aussi le cas du module logarithmique. Ces exemples peuvent être étendus pour plus de deux flux stochastiques à additionner. Cependant, l'augmentation d'ordre d'extension mène également à une augmentation des ressources matérielles consommés. Par conséquent, le choix de ce couple d'ordres appropriés est critique pour assurer un bon compromis entre l'exactitude de probabilité et la complexité. Afin de bien choisir ce couple d'ordres, par la suite, nous allons donner une évaluation des ordres nécessaires pour le passage dans le domaine exponentiel d'un seul flux stochastique.

Dans un premier temps, il faut noter qu'un de nos objectifs est de diminuer le nombre des opérations addition stochastique, notamment des additions stochastiques à plusieurs entrées. Or,

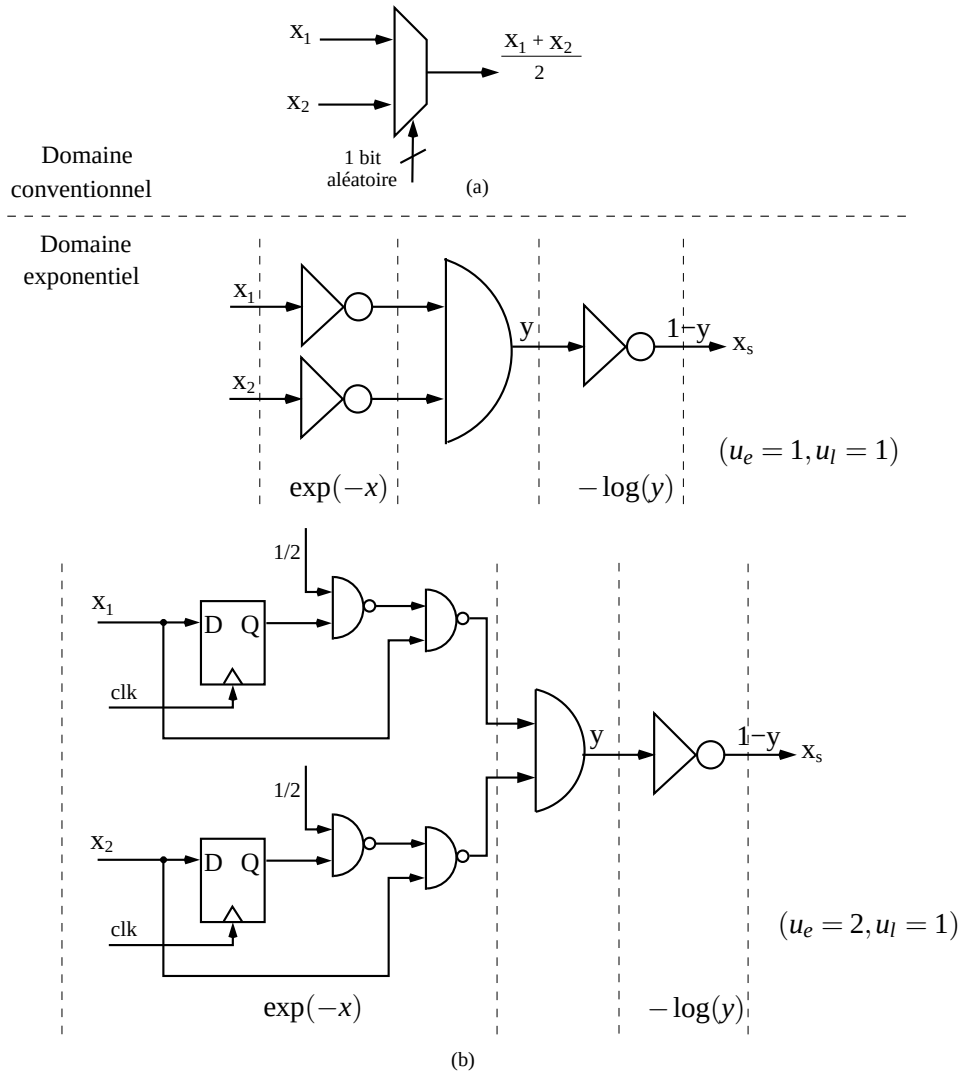


FIGURE 3.5 – Circuits stochastiques représentant l'opération d'addition stochastique dans les deux domaines : (a).Domaine conventionnel. (b).Domaine exponentiel.

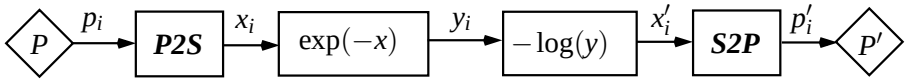
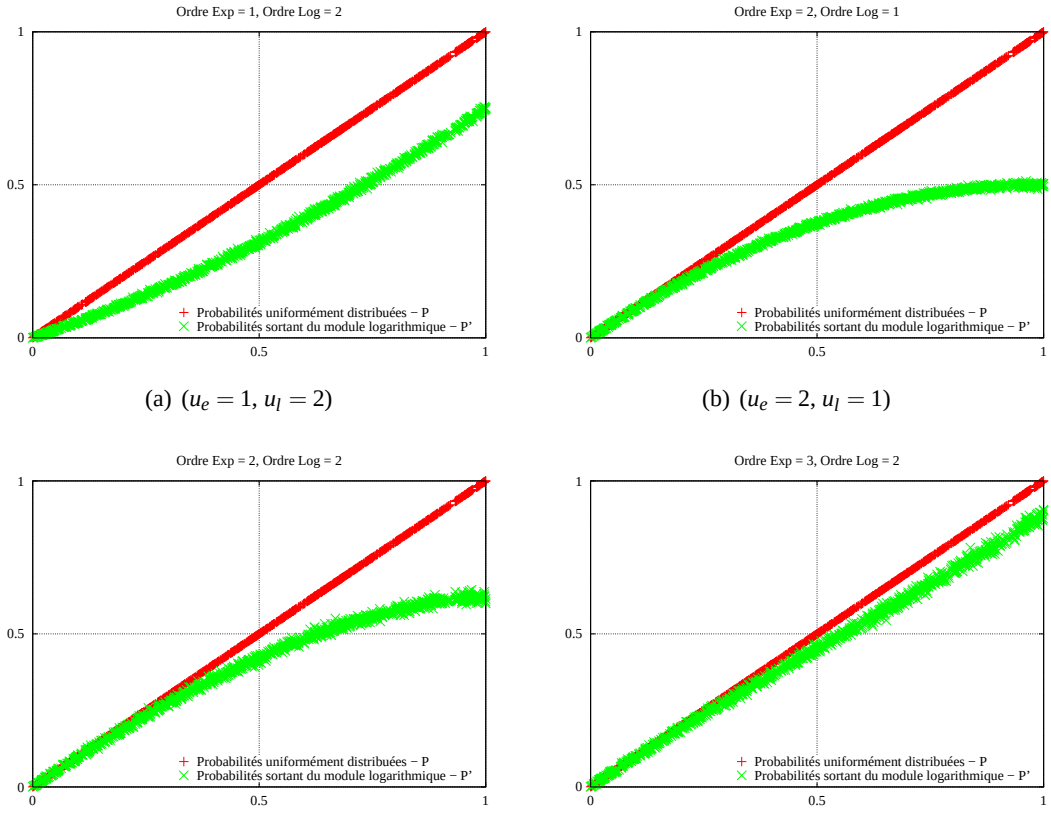


FIGURE 3.6 – Processus d'évaluation des ordres nécessaires aux modules exponentiels et logarithmiques.

il est préférable que l'ordre du module logarithme soit limité au maximum à 4. En effet, pour des plus grands ordres, ce module nécessite des additions stochastiques à 8, 16,... entrées.

La figure 3.6 représente un schéma simple permettant une évaluation des ordres des modules exponentiels et logarithmiques. Dans ce cas, un modèle simple de transformation d'un flux stochastique conventionnel en un flux stochastique exponentiel est étudié. Un module logarithmique est utilisé pour récupérer le flux stochastique conventionnel à partir du résultat de

la conversion exponentielle. Pour cette étude, on considère un modèle simple se composant d'un jeu de probabilités P qui produit 1000 probabilités uniformément distribuées $p_i(i = 1 \rightarrow 1000)$. Chaque probabilité p_i est représentée par un flux stochastique x_i à l'aide d'un convertisseur probabilité-stochastique ***P2S***. Ce convertisseur comprend un comparateur et RNG qui génère des valeurs (pseudo-)aléatoires pendant 10k cycles d'horloge. Le principe de conversion probabilité-flux stochastique a été déjà détaillée en sous-section 1.3.1.1 au chapitre 1. Puis, un convertisseur stochastique-probabilité ***S2P*** sert à récupérer des probabilités à partir des flux stochastiques correspondants en comptant les occurrences de '1'.



(a) $(u_e = 1, u_l = 2)$ (b) $(u_e = 2, u_l = 1)$

(c) $(u_e = 2, u_l = 2)$ (d) $(u_e = 3, u_l = 2)$

FIGURE 3.7 – Evaluation de l'ordre des modules pour le passage dans le domaine exponentiel.

La figure 3.7 décrit des probabilités initiales et celles après le traitement dans le domaine exponentiel en fonction de l'ordre des modules. Le nuage rouge représente des probabilités uniformément distribuées et le vert, celui des probabilités après la sortie du traitement exponentiel. La figure 3.7 nous montre que le couple $(u_e = 3, u_l = 2)$ donne une estimation quasi-identique : le nuage des probabilités récupérées est très proche du nuage des probabilités transmises. En outre, le couple $(u_e = 2, u_l = 2)$ fournit une estimation proche à faibles probabilités et meilleure que le couple $(u_e = 2, u_l = 1)$ à grandes probabilités. Cependant, en tenant compte de la complexité pour l'ordre $u_e = 3$, le module exponentiel exige une conversion du facteur 1/3 en un flux stochastique. Pour ce faire, un générateur des valeurs aléatoires de W bits et un comparateur sont requis. Ce

problème s'accentue lorsque le nombre d'entrées du module exponentiel est élevé. Il complexifie l'architecture d'un décodeur APP où un grand nombre d'opérations addition est inévitable.

Néanmoins, ces résultats ne sont pas suffisants pour déterminer le couple d'ordres (u_e, u_l). Basé sur les résultats fonctionnels de l'approche exponentielle avec différents ordres pour le turbo-décodage qui seront détaillés dans la section 3.4, nous notons que le couple ($u_e = 1$, $u_l = 1$) est suffisant pour présenter l'opération d'addition stochastique dans le domaine exponentiel. Cet intérêt est dû à une bonne approximation de l'équation 3.4 et ces deux modules comprennent seulement des opérations NOT dont la complexité matérielle est moins coûteuse. C'est pourquoi, un modèle approximatif composé d'un module exponentiel de l'ordre $u_e = 1$ et d'un module logarithmique de l'ordre $u_l = 1$ est considéré par la suite.

3.3 Conception des modules d'un turbo-décodeur stochastique intégrant des traitements dans le domaine exponentiel

Cette section introduit l'application du traitement exponentiel dans différents modules d'un turbo-décodeur stochastique. Comme préalablement montré au cours du premier chapitre, des opérations d'addition stochastique à plusieurs entrées apparaissent dans les modules A/B, Ext et DEC. Leurs architectures dans le domaine exponentiel seront successivement détaillées dans cette section.

3.3.1 Module A/B

Comme déjà expliqué au chapitre 2, l'architecture du module A/B se compose de deux étapes successives de calcul : calcul des nouvelles métriques et normalisation de nouvelles métriques obtenues. Ce module exige donc la connaissance des métriques de branche et des métriques récurrentes adjacentes de l'instant précédent. Par exemple, les métriques récurrentes *aller* $\alpha^{i+1}(s)$ pour la section i du treillis sont calculées dans le domaine stochastique conventionnel à partir des métriques récurrentes *aller* $\alpha^i(s')$ et des métriques de branches $\gamma^i(s',s)$. Ce calcul peut s'exprimer comme suit :

$$\alpha^{i+1}(s) = \frac{\sum_{s'=0}^{7} \alpha^i(s')\gamma^i(s',s)}{\sum_{\varepsilon=0}^{7}\sum_{s'=0}^{7} \alpha^i(s')\gamma^i(s',\varepsilon)} \tag{3.6}$$

Les sorties des modules A et B sont les huit bits représentant les nouvelles métriques récurrentes *aller* à l'instant suivant $\alpha^{i+1}(s)(s = 0 \rightarrow 7)$ pour le module A et $\beta^i(s)(s = 0 \rightarrow 7)$ pour le module B. L'architecture du module B est par conséquence similaire à celle du module A.

3.3.1.1 Calcul de nouvelles métriques récurrentes exponentielles

Dans le domaine stochastique conventionnel, avant l'étape de normalisation de nouvelles métriques *aller*, l'équation 3.6 contient des opérations addition représentées par des MUX2 :1 pour calculer les nouvelles métriques récurrentes. Par exemple, la valeur de métrique récurrente de

l'état $\hat{\alpha}^{i+1}(1)$ est donnée par l'équation 3.7 :

$$\begin{aligned} \hat{\alpha}^{i+1}(1) = &\quad \alpha^i(2).\gamma^i(2,1) + \alpha^i(3).\gamma^i(3,1) \\ \hat{\alpha}^{i+1}(1) = &\quad \alpha^i(2).\gamma^i(2) + \alpha^i(3).\gamma^i(1) \end{aligned} \tag{3.7}$$

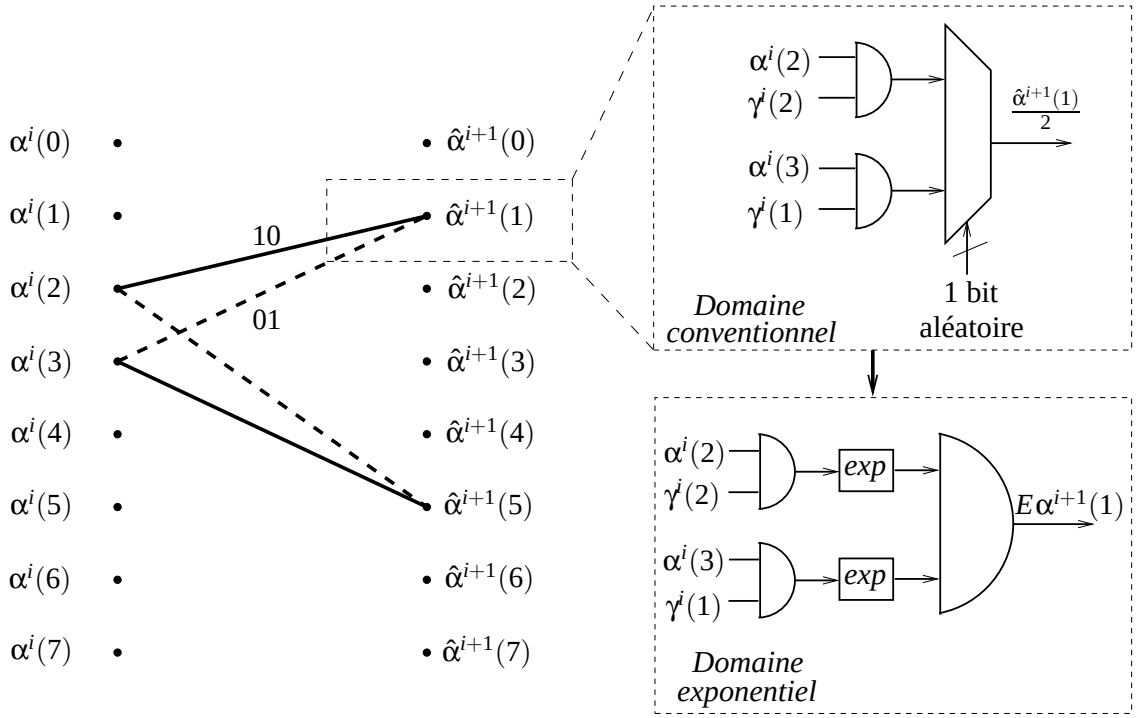


FIGURE 3.8 – Diagramme des blocs de calcul d'une métrique d'état α dans les domaines exponentiel et conventionnel.

Dès lors, le traitement dans le domaine exponentiel du module *A* permet le remplacement des additions. Pour ce faire, toutes les multiplications $\alpha^i(s')\gamma^i(s',s)$ doivent être d'abord calculés. Ensuite, chaque flux résultant se connecte à un module exponentiel pour fournir une métrique récurrente exponentielle $E\alpha^{i+1}(s)(s = 0 \rightarrow 7)$. Cela signifie que la métrique récurrente exprimée en équation 3.7 est décrite en domaine exponentiel comme suit :

$$E\alpha^{i+1}(1) = e^{-\alpha^i(2).\gamma^i(2)} \times e^{-\alpha^i(3).\gamma^i(1)} \tag{3.8}$$

La figure 3.8 illustre l'architecture équivalente à cette expression. Dans ce cas, chaque module exponentiel $\exp(-x)$ est assigné à une des branches du treillis. Le nombre de modules exponentiels est équivalent à celui de branches du treillis, soit 16 modules exponentiels pour le turbocode considéré. Une porte logique AND2 à deux entrées est par ailleurs nécessaire pour multiplier ces deux métriques récurrentes exponentielles. Ce processus de traitement se répète afin de calculer l'ensemble des métriques récurrentes dans le domaine exponentiel.

3.3.1.2 Normalisation des métriques récurrentes exponentielles

Dans le domaine stochastique conventionnel, la normalisation de chaque flux stochastique représentant une métrique d'état est réalisée par une bascule JK (*c.f* chapitre 1). Chaque normalisation requiert la somme de toutes les métriques d'état. Par exemple, la normalisation de métrique récurrente $\alpha^{i+1}(1)$ de l'état 1 à l'instant i s'exprime par l'équation 3.9 :

$$\alpha^{i+1}(1) = \frac{\hat{\alpha}^{i+1}(1)}{\sum_{s=0}^{7} \hat{\alpha}^{i+1}(s)} = \frac{\hat{\alpha}^{i+1}(1)}{\hat{\alpha}^{i+1}(1) + \sum_{j=0, j\neq 1}^{7} \hat{\alpha}^{i+1}(j)} = \frac{J_1^{i+1}}{J_1^{i+1} + K_1^{i+1}} \tag{3.9}$$

En utilisant la représentation exponentielle des métriques récurrentes, la partie K_1^{i+1} du dénominateur de l'équation 3.9 - la somme des probabilités récurrentes - est maintenant remplacée par $E_s\alpha^{i+1}(1)$ qui est décrite dans l'équation 3.10 :

$$E_s\alpha^{i+1}(1) = \prod_{j=0\to 7, j\neq 1} E\alpha^{i+1}(j) \tag{3.10}$$

Il suffit d'utiliser une porte logique AND7 à 7 entrées pour calculer la quantité $E_s\alpha^{i+1}(s)$ de l'état s à l'instant i. Chaque entrée de cette porte AND7 est une métrique récurrente exponentielle $E\alpha^{i+1}(j)$ qui est obtenue à partir de l'expression 3.8 pour tout $(j = 0 \to 7, j \neq s)$.

Les métriques récurrentes à l'instant i sont utilisées par d'autres modules et par les sections qui suivent. Alors, afin de s'adapter au traitement stochastique conventionnel, chaque métrique récurrente exponentielle $E\alpha^{i+1}(1)$ et $E_s\alpha^{i+1}(1)$ doit être exprimée dans le domaine conventionnel, grâce à des modules logarithmiques $-\log$. Chaque métrique récurrente exponentielle nécessite un module $-\log$ pour la conversion inverse. Le processus de récupération des flux stochastiques du domaine exponentiel au domaine conventionnel et de leur normalisation est détaillé dans la figure 3.9.

Afin de normaliser les flux stochastiques récupérés, des architectures de mémoire EM (comme expliqué dans la section 2.4.3 du chapitre 2) sont considérées pour supprimer la corrélation. 8 mémoires EM sont donc requises. Un avantage de l'exploitation de représentation exponentielle est qu'elle n'a pas besoin de vecteurs aléatoires de 3 bits pour le facteur d'échelle $1/8$ comme expliqué dans la figure 2.9 du chapitre 2.

3.3.2 Module *Ext*

Dans le cas d'un code simple-binaire, le module *Ext* calcule deux probabilités extrinsèques de sortie qui correspondent à deux symboles possibles $(d^i = j)$ avec $j \in (0;1)$. Cette étape est établie lorsque les métriques récurrentes *aller* $\alpha^i(j)(j = 0 \to 7)$ et *retour* $\beta^{i+1}(l)(l = 0 \to 7)$ ainsi que les probabilités de redondance $\gamma_e^i(k)(k = 0 \to 1)$ sont disponibles à l'entrée du module *Ext* (figure 3.10).

Les sorties du module *Ext* sont deux probabilités extrinsèques $\Pr^i(n)(n = 0,1)$ qui sont utilisées en tant que probabilités *a priori* par l'autre décodeur SISO. Comme déjà expliqué au chapitre 2, nous divisons ce module en deux étapes successives de traitement : calcul des probabilités extrinsèques et normalisation des probabilités extrinsèques.

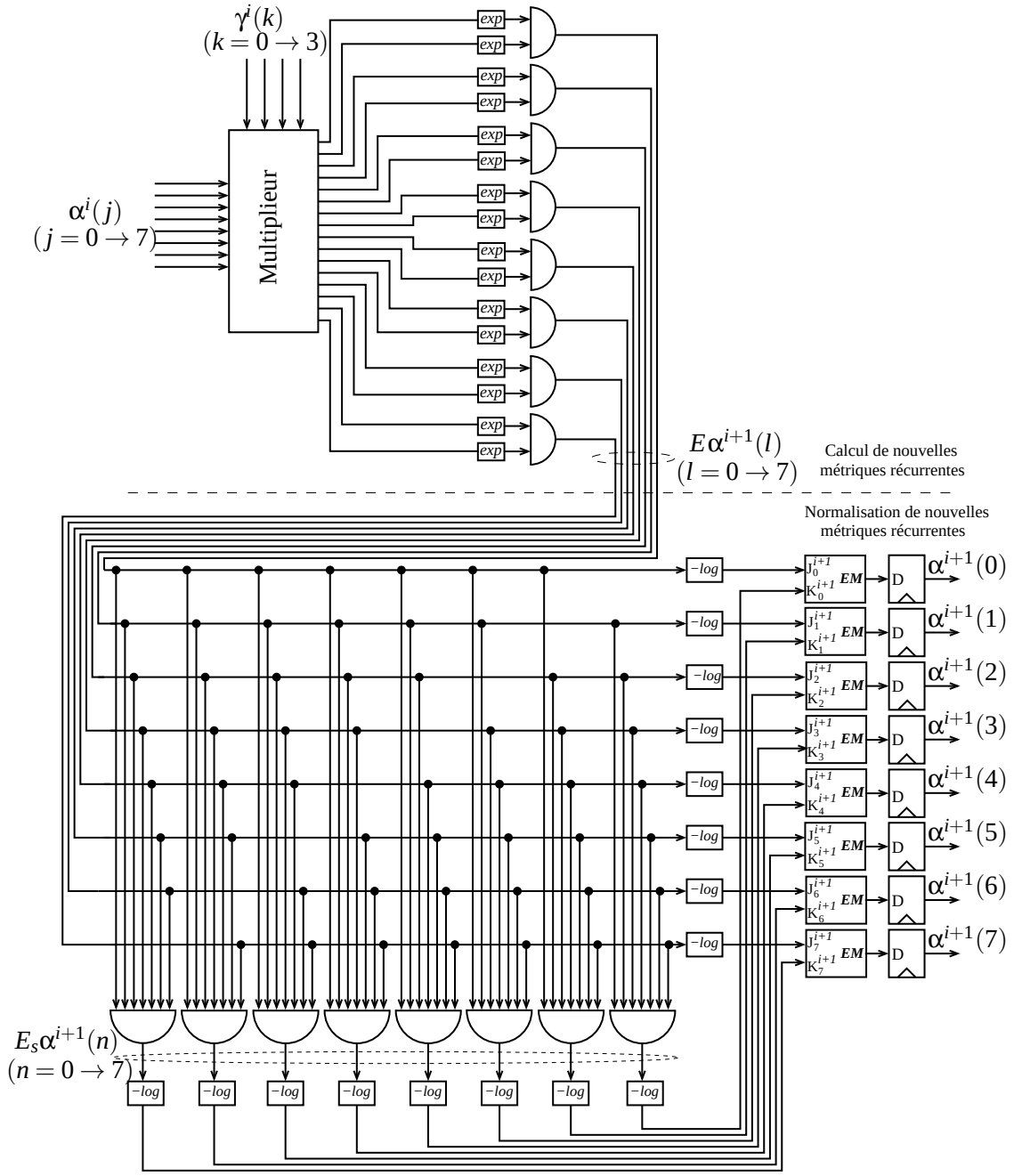


FIGURE 3.9 – Diagramme de l'architecture de traitement exponentiel du module *A*.

3.3.2.1 Opérations successives de traitement

Dans le domaine conventionnel, chaque probabilité extrinsèque, avant l'étape de normalisation à l'instant *i*, est obtenue par une addition de huit probabilités extrinsèques de branche *a posteriori*. Chaque probabilité extrinsèque de branche *a posteriori* est le produit de trois probabilités :

– Probabilité récurrente *aller* : $\alpha^i(s')$ de l'état s' à l'instant *i*.

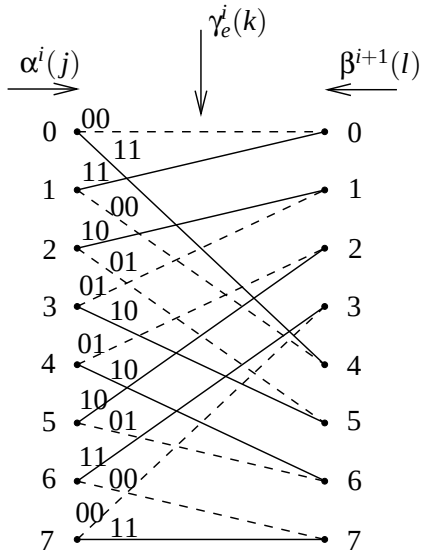


FIGURE 3.10 – Représentation d'une section de treillis.

– Probabilité récurrente *retour* : $\beta^{i+1}(s)$ de l'état s à l'instant $i+1$.

– Probabilité de redondance $\gamma_e^i(s',s)$ entre les deux états s' et s.

Nous reprenons l'exemple présenté en équation 2.23 du chapitre 2. La probabilité extrinsèque du symbole $d^i = 1$, illustrée par des traits pleins sur le treillis de la figure 3.10, s'exprime comme suit :

$$\begin{aligned}\Phi_e^i\left(d^i=1\right) = {} & \alpha^i(0).\gamma_e^i(0,4).\beta^{i+1}(4) + \alpha^i(1).\gamma_e^i(1,0).\beta^{i+1}(0) \\ & + \alpha^i(2).\gamma_e^i(2,1).\beta^{i+1}(1) + \alpha^i(3).\gamma_e^i(3,5).\beta^{i+1}(5) \\ & + \alpha^i(4).\gamma_e^i(4,6).\beta^{i+1}(6) + \alpha^i(5).\gamma_e^i(5,2).\beta^{i+1}(2) \\ & + \alpha^i(6).\gamma_e^i(6,3).\beta^{i+1}(3) + \alpha^i(7).\gamma_e^i(7,7).\beta^{i+1}(7)\end{aligned} \tag{3.11}$$

Un multiplexeur à 8 entrées et un vecteur de 3 bits aléatoires sont nécessaires pour accomplir cette addition. C'est aussi le cas du calcul de la probabilité extrinsèque du symbole $d^i = 0$.

3.3.2.2 Architecture du module dans le domaine exponentiel

La conversion des additions dans le domaine exponentiel est réalisée en supprimant les grands multiplexeurs à entrées multiples. Pour ce faire, un module exponentiel est assigné à chaque probabilité extrinsèque de branche *a posteriori*. Une porte logique AND à 8 entrées est requise. Ses entrées sont connectées à des sorties de modules exponentiels . Par conséquent, la probabilité extrinsèque $\Phi_e^i\left(d^i=1\right)$ de l'expression 3.11 est exprimée dans le domaine exponentiel comme suit :

$$E\Phi_e^i(d^i=1) = \prod_{(s',s)/d^i(s',s)=1} \exp\left(-\alpha^i\left(s'\right).\gamma_e^i\left(s',s\right).\beta^{i+1}\left(s\right)\right) \tag{3.12}$$

Le diagramme en blocs de cette conversion est détaillé dans la figure 3.11. Le module multiplieur sert à multiplier les trois probabilités $\alpha^i(s')$, $\beta^{i+1}(s)$ et $\gamma_e^i(s',s)$ en choisissant les liaisons entre les états s' et s fournies par le treillis. Les huit premières probabilités extrinsèques

de branche *a posteriori* correspondent au symbole ($d^i = 0$). Les huit suivantes correspondent au symbole ($d^i = 1$). Deux modules $-\log$ sont nécessaires afin de transformer les flux stochastiques

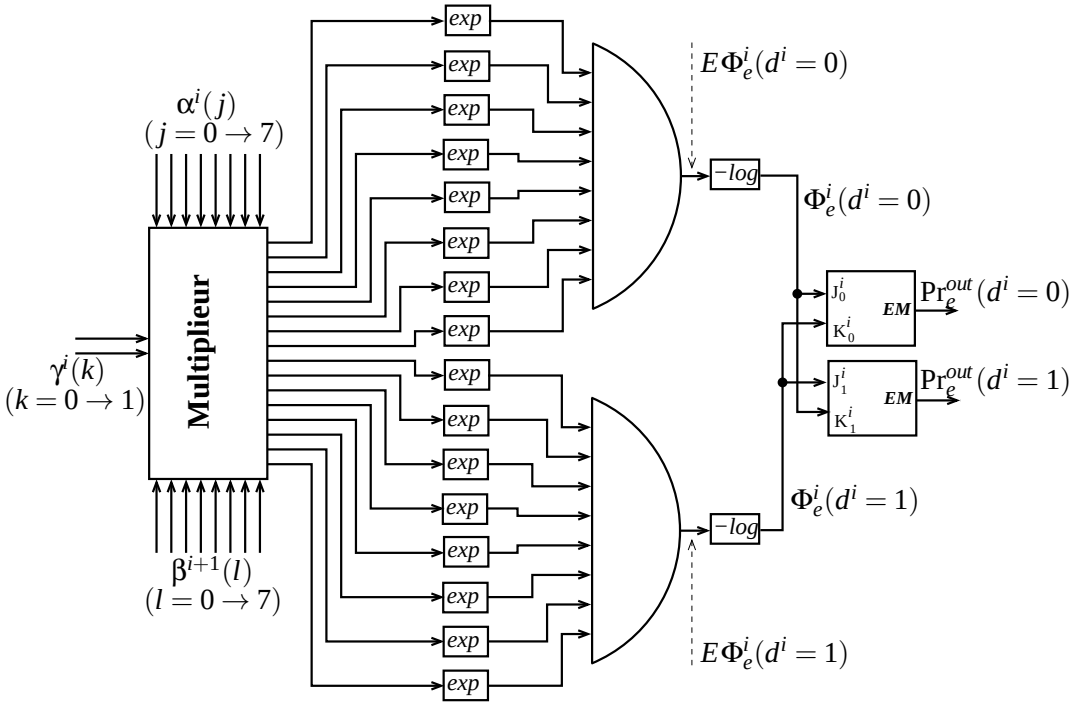


FIGURE 3.11 – Diagramme de l'architecture du module *Ext* dans le domaine exponentiel.

exponentiels correspondant aux deux probabilités extrinsèques exponentielles $E\Phi_e^i(d^i = j)$ avec $j \in (0;1)$ dans le domaine conventionnel. Deux mémoires EMs sont nécessaires afin d'éliminer la corrélation entre deux flux stochastiques représentant deux probabilités extrinsèques de sortie et pour les normaliser.

3.3.3 Module *DEC*

3.3.3.1 Opérations successives de traitement

Ce module calcule les deux probabilités *a posteriori* et ensuite détermine le symbole le plus vraisemblable associé au symbole d^i en choisissant la probabilité *a posteriori* la plus grande. Ces deux probabilités *a posteriori* correspondant aux deux symboles possibles $b \in (0,1)$ sont calculées à partir des probabilités récurrentes α^i, β^{i+1} et des probabilités de branche γ^i lorsque ces probabilités sont disponibles à l'entrée du module *DEC*. La sortie du module *DEC* correspond à l'estimation dure du symbole binaire $\widehat{d^i}$. La sortie du module *DEC* correspond à l'estimation dure du symbole binaire $\widehat{d^i}$. Dans le domaine conventionnel, chaque probabilité *a posteriori* du symbole $b = 1$ est la somme des huit probabilités *a posteriori* de branche correspondant à la transition de l'étiquette $d^i y^i = 1y^i$. Chaque probabilité *a posteriori* de branche est le produit de trois flux stochastiques qui représentent :

– Probabilités récurrentes *aller* $\alpha^i(s')(s' = 0 \rightarrow 7)$

– Probabilités récurrentes *retour* $\beta^{i+1}(s)(s=0\rightarrow 7)$

– Probabilités de branche $\gamma^{i}(j)(j=0\rightarrow 4)$

Nous reconsidérons l'exemple donné dans l'équation 2.24 au chapitre 2. La probabilité *a posteriori* du symbole $b=1$ est donnée par l'expression :

$$\Pr\left(d^i=1|u^i,v_1^i\right)=\sum_{(s',s)/d_i(s',s)=1}\alpha^i\left(s'\right)\beta^{i+1}\left(s\right)\gamma^i\left(s',s\right) \tag{3.13}$$

Un multiplexeur à 8 entrées et un vecteur de 3 bits aléatoires sont nécessaires pour réaliser ce calcul. Le module *DEC* compare ensuite ces probabilités *a posteriori* pour ne garder que la probabilité la plus grande. La probabilité la plus grande indique que le symbole correspondant est la décision dure du symbole binaire transmis d^i. La sortie du module *DEC* correspond à l'estimation dure du symbole binaire $\widehat{d^i}$.

3.3.3.2 Architecture du module dans le domaine exponentiel

Le module *DEC* dans le domaine exponentiel exige la suppression des multiplexeurs à entrées multiples. Cette suppression implique une association d'un module exponentiel à chaque probabilité *a posteriori* de branche. Une porte logique AND à 8 entrées est indispensable afin de traiter cette conversion. Alors, la probabilité *a posteriori* $\Pr\left(d^i=1|u^i,v_1^i\right)$ de l'expression 3.13 est réécrite dans le domaine exponentiel comme suivant :

$$E\Pr\left(d^i=1|u^i,v_1^i\right)=\prod_{(s',s)/d^i(s',s)=1}\exp\left(-\alpha^i\left(s'\right).\gamma^i\left(s',s\right).\beta^{i+1}\left(s\right)\right) \tag{3.14}$$

Le diagramme du module *DEC* dans le domaine exponentiel est illustré dans la figure 3.12. Le module multiplieur produit les huit probabilités *a posteriori* de branche. L'architecture du module multiplieur a été détaillée au chapitre 2. Chaque probabilité est ensuite désignée par un module exponentiel, alors 8 modules exponentiels pour notre code. Afin de regrouper ces huit probabilités, une porte logique AND à 8 entrées est requise.

$\Pr(0)$	$\Pr(1)$	$E\Pr(0)$	$E\Pr(1)$	X	Y	compteur
0	0	1	1	0	0	=
1	0	0	1	1	0	++
0	1	1	0	0	1	- -
1	1	0	0	0	0	=

TABLE 3.1 – Tableau de vérité pour l'estimation du symbole codé.

Comme déjà expliqué au chapitre 2, la prise de décision du symbole codé est réalisée par un compteur up/down saturé de *m* bits. En effet, afin de s'adapter au décodage conventionnel, nous devons transformer les probabilités *a posteriori* dans le domaine exponentiel au domaine conventionnel en utilisant des modules logarithmiques. Cependant, ce processus de conversion peut être évité dû à la caractéristique du module $\exp(-x)$ lorsqu'il a besoin d'une seule porte NOT. Si la

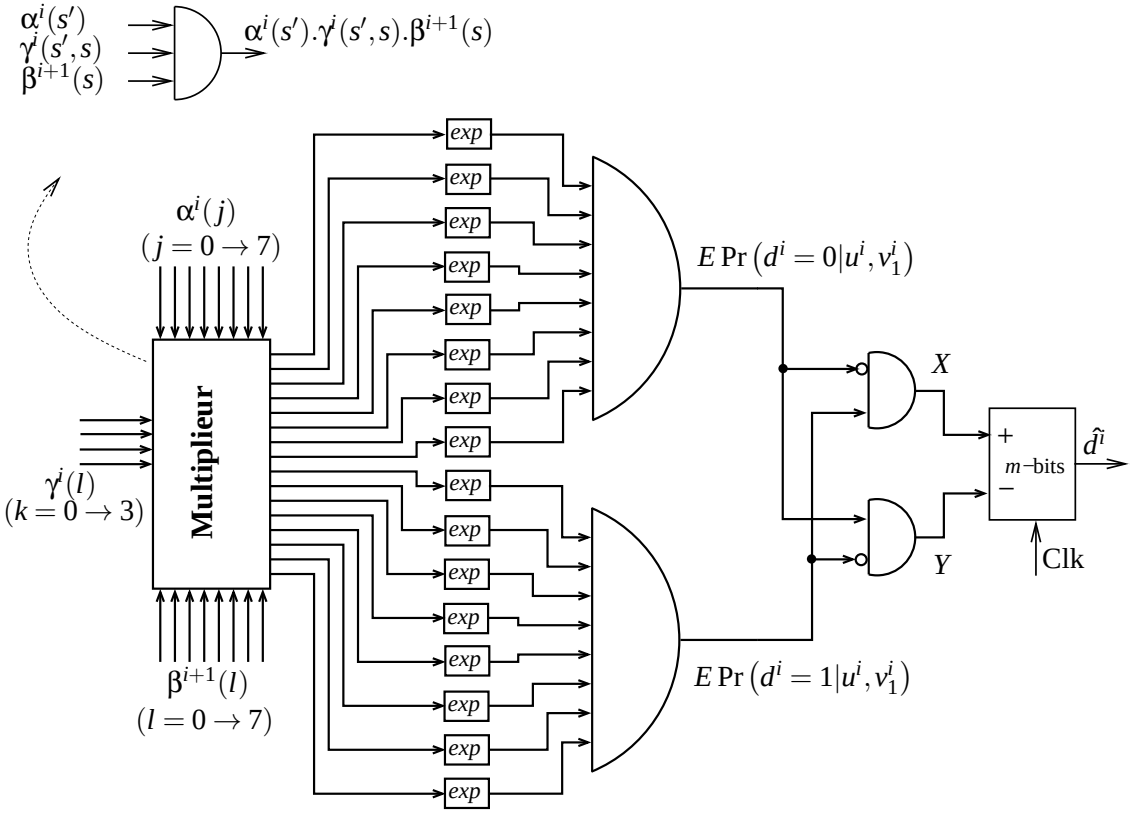


FIGURE 3.12 – Diagramme des blocs du module *DEC* dans le domaine exponentiel.

probabilité x s'évolue et devient plus grande, la probabilité $\exp(-x)$ se baisse et devient alors plus faible. De plus, à l'aide de nos observations sur des simulations comportementales en C++, l'approximation du premier ordre du module exponentiel est suffisante pour estimer la probabilité *a posteriori*. Par conséquent, la comparaison des probabilités *a posteriori* exponentielles est suffisante pour trouver la décision dure du symbole codé. La complexité du circuit est alors réduite par la suppression des modules logarithmiques. Ce choix d'implémentation du module *DEC* dans le domaine exponentiel a été bien validé par les performances de décodage qui seront présentées dans la section suivante.

Comme montré au tableau 3.1, lorsque le bit représentant la probabilité $\Pr(0)$ qui correspond au symbole estimé $d^i = 0$ est égal à '1' et celui représentant la probabilité $\Pr(1)$ qui correspond au symbole estimé $d^i = 1$ est égal à '0', le compteur est incrémenté d'une unité. Cette incrémentation a lieu si le compteur n'a pas atteint à sa limite maximum $(+2^{m-1}-1)$. A l'inverse, si le bit représentant la probabilité $\Pr(0)$ est égal à '0' et celui représentant la probabilité $\Pr(1)$ est égal à '1', le compteur décroît une unité jusqu'à sa limite minimum (-2^{m-1}). Dans les autres cas, le contenu du compteur n'évolue pas.

Lorsque le nombre maximum de cycles DC_{max} est atteint, le décodeur arrête le processus de décodage. Le signe de la valeur dans le compteur est considéré pour la décision du symbole le plus vraisemblable. Si le bit de poids fort MSB est égal à '1', le symbole émis est $\hat{d^i} = 0$. Sinon, $\hat{d^i} = 1$.

3.4 Performance de correction d'erreurs de décodeurs stochastiques exponentiels

Dans cette section, l'avantage de l'approche exponentielle sera mis en évidence au niveau des performances de décodage stochastique pour des codes convolutifs et des turbocodes. Ensuite, nous comparons la complexité de l'architecture d'un décodeur stochastique conventionnel et celle d'un décodeur stochastique exponentiel. Enfin, nous terminons cette section par une estimation du débit d'un turbo-décodeur stochastique dans le domaine exponentiel.

Afin d'obtenir une bonne comparaison, les symboles reçus à partir du canal sont quantifiées sous le format ($p = 7$, $q = 4$). La taille des générateurs aléatoires (ou la sortie des mémoires ROM) est de $W = 7$ bits. Les compteurs du module *DEC* sont des compteurs up/down saturés sur 3 bits. La solution *NDS* et la technique des mémoires EM de longueur 32 bits sont utilisées afin de supprimer les problèmes de corrélation. Notre étude se déroule via une transmission BPSK sur un canal AWGN. Comme justifié ci-dessus, les ordres des modules exponentiels et logarithmiques ($u_s = 1$, $u_l = 1$) sont suffisants pour assurer un bon compromis exactitude complexité.

3.4.1 Performance du décodage convolutif stochastique exponentiel APP

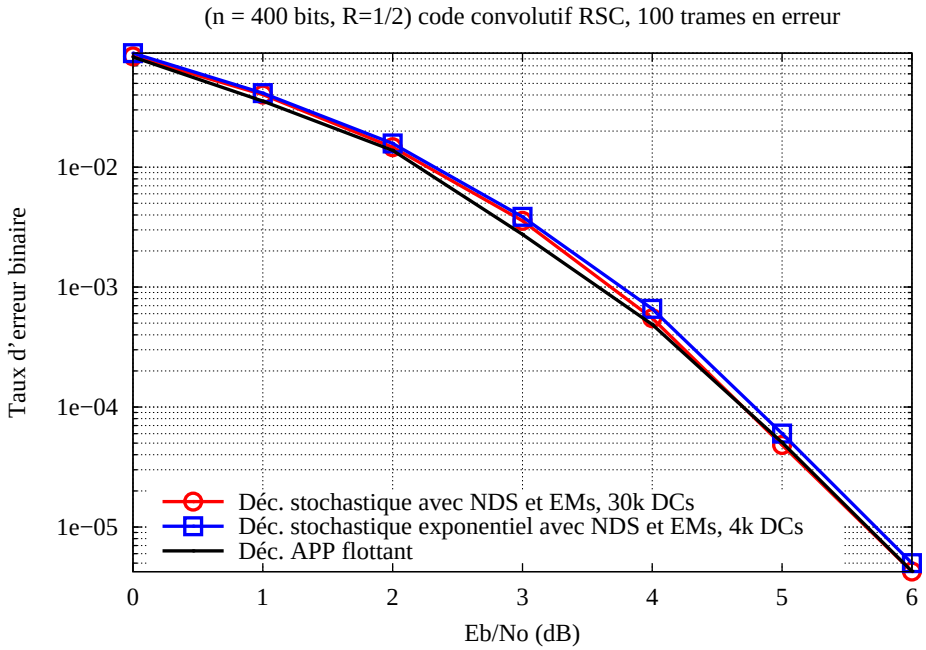


FIGURE 3.13 – Performance du décodage stochastique exponentiel pour un code convolutif RSC (k=200, $R = 1/2$).

La figure 3.13 nous donne les performances de décodage d'un code convolutif RSC de longueur $k = 200$ et de rendement $R = 1/2$ en termes de taux d'erreur binaire - TEB. Ces performances sont obtenues par des simulations fonctionnelles de description en langage C++ pour trois différentes architectures. La courbe rouge décrit les performances d'un décodage stochas-

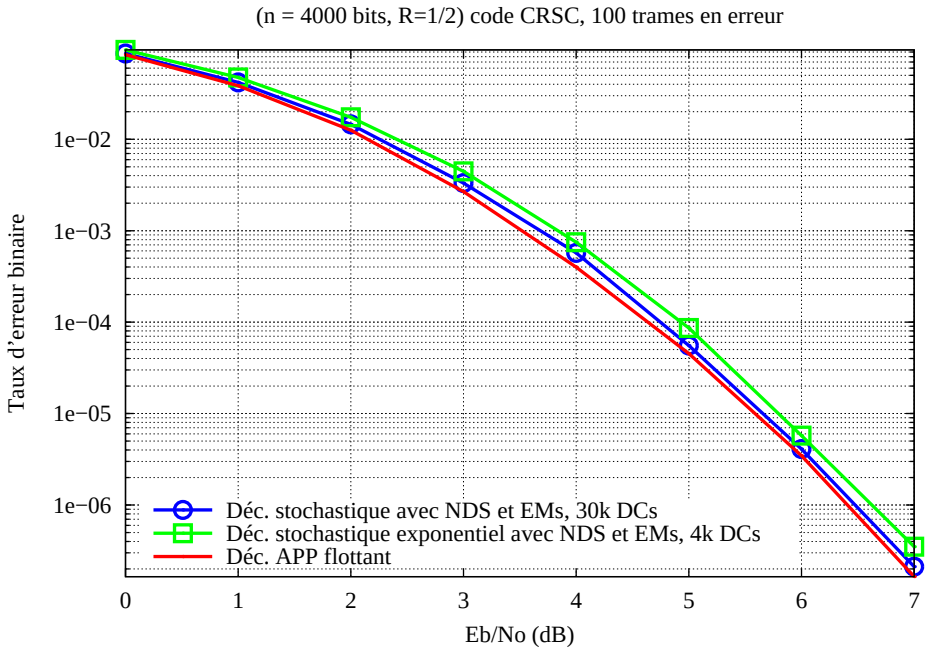


FIGURE 3.14 – Performance du décodage stochastique exponentiel pour un code convolutif RSC (k=2000, $R = 1/2$).

tique conventionnel. La courbe bleue représente les performances de notre décodage stochastique exponentiel. La courbe noire est le résultat d'un décodage en virgule flottante qui exploite l'algorithme APP de référence. Pour un code convolutif, le facteur $\psi(\sigma)$ de *NDS* est retenu et égal à 1.0 pour toutes les valeurs de σ. Le coefficient $\Omega = \sigma_0^2 = 1,0$.

Nous pouvons noter que l'approche exponentielle apporte une performance de décodage quasi-similaire par rapport à celle du décodage stochastique conventionnel et à celle du décodage en virgule flottante. Cependant, le nombre de cycle de décodage maximum DC_{max} est significativement réduit de 30k DCs (décodage stochastique conventionnel) à 4k DCs (décodage stochastique exponentiel). Cela signifie que l'approche exponentielle fournit une meilleure vitesse de convergence en garantissant une performance comparable à celle d'un décodage conventionnel.

La figure 3.14 nous montre également les performances de décodage d'un code convolutif de taille pratique (k = 2000 bit, R=$1/2$). Nous observons que le couple d'ordres ($u_s = 1$, $u_l = 1$) est suffisant, à savoir une perte de performance : 0,2dB par rapport au décodage en virgule flottante. Ainsi, le décodeur exponentiel nécessite seulement $DC_{max} = 4k$ pour atteindre cette performance. Par comparaison, un décodeur stochastique conventionnel exige, quant à lui, 30k DCs. Un gain de 80% est donc obtenu sur la vitesse de convergence.

3.4.2 Performance de turbo décodage stochastique exponentiel

La figure 3.15 décrit les performances de décodage par des simulations fonctionnelles de description en langage C++ pour un turbocode de taille $k = 200$ et de rendement $R = 1/3$. Le coefficient *NDS* de décodage stochastique des turbocodes est choisi sur la base de la meilleure

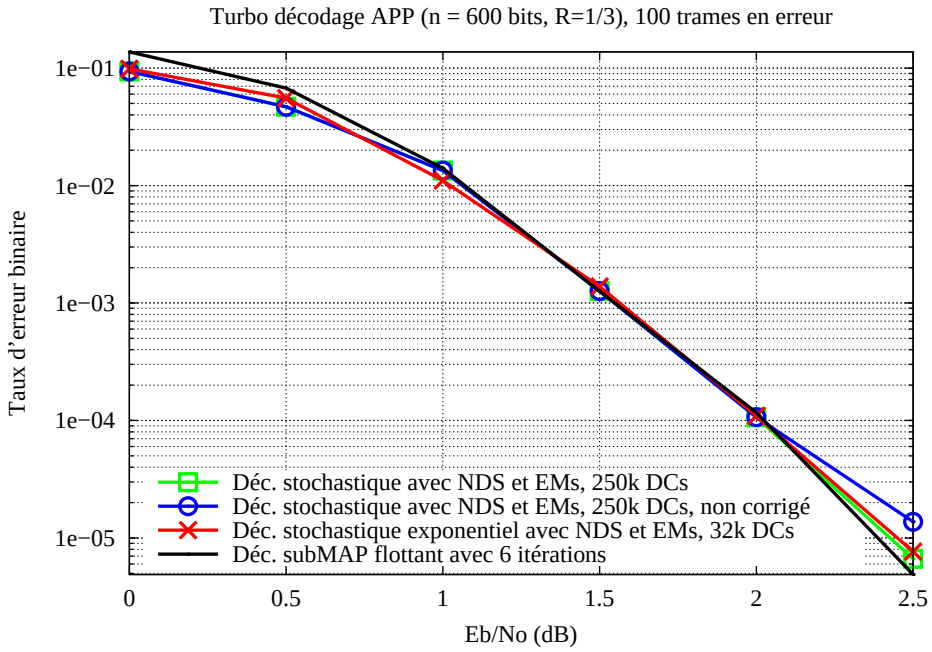


FIGURE 3.15 – Performance du décodage stochastique exponentiel pour des turbocodes convolutifs (k=200, $R = 1/3$).

performance de décodage. Dans le cadre d'une transmission BPSK sur le canal AWGN, le coefficient Ω est égal à $\sigma_0^2 = 1.5$. Le *facteur de correction* $\psi(\sigma)$ doit varier en fonction de différentes valeurs de RSB pour que les probabilités reçues à partir du canal s'éloignent des probabilités 0 et 1 à grand RSB. $\psi(\sigma)$ = 1,0 pour RSB = (0.0dB $\rightarrow$ 2.0dB) et $\psi(\sigma) = 1,2$ pour un grand RSB (2.5dB).

Les résultats du décodage stochastique sont comparés avec ceux d'un décodeur SubMAP flottant de 6 itérations qui est le plus souvent retenu pour une implémentation matérielle de turbo-décodeur en pratique. Nous constatons que le décodeur stochastique exponentiel (la courbe rouge) atteint une performance quasi-similaire à celles d'un décodeur stochastique *version corrigée* (la courbe bleue) et d'un décodeur SubMAP flottant (la courbe noire). Cependant, l'exploitation du domaine exponentiel permet de diminuer efficacement le nombre de cycle maximum DC_{max} : de 250k DCs (décodage stochastique conventionnel) à 32k DCs (décodage stochastique exponentiel), soit un gain de 87%. Néanmoins, ce résultat très encourageant est encore insuffisant pour obtenir un grand débit de décodage. C'est pourquoi, d'autres solutions doivent être proposées afin de réduire le nombre de cycles DCs.

3.5 Complexité matérielle d'un décodeur stochastique exponentiel APP

La complexité matérielle d'un décodeur stochastique exponentiel APP est exprimée dans le tableau 3.2. Ce tableau détaille l'estimation de complexité des modules élémentaires d'un turbo-

décodeur stochastique à 8 états dans le domaine exponentiel. Cette complexité est illustrée en termes d'éléments logiques : des bascules, des portes logiques ainsi que du compteur et des bits aléatoires.

Module	Ressources matérielles élémentaires								Bits aléatoires	
	NAND2	AND2	OR2	XOR2	MUX2	MUX8	compteur	D-FF	7 bits	1 bit
Γ		42	14						2	
A / B	52		8	8	16			264		40
Ext	32		2	2	4			66		10
Dec	24	16					1			
Total	160	58	32	18	36		1	592	2	90

TABLE 3.2 – Complexité d'une section d'un turbo-décodeur stochastique binaire à 8 états avec addition dans domaine exponentiel.

Cette complexité est considérée pour le cas du couple d'ordres des modules exponentiels et logarithmiques ($u_e = 1$, $u_l = 1$). Par conséquent, chaque module exponentiel et logarithmique est modélisé par une seule porte logique NOT. La complexité de chacune des portes NAND7 au sein des modules A et B est approximativement considérée comme $7/2$ de celle d'une porte logique NAND2, et la complexité de chaque porte NAND3 est approximativement considérée comme $3/2$ de celle d'une porte logique NAND2. Chacune des portes logiques AND8 dans le module Ext est composée de 7 portes AND2.

En comparaison avec la complexité d'un turbo-décodeur stochastique conventionnel, le coût additionnel des opérations addition dans le domaine exponentiel est raisonnable. Seules 160 portes logiques NAND2 sont ajoutées pour remplacer les 18 MUX8 :1 dans le domaine conventionnel. Néanmoins, l'approche exponentielle réduit le nombre de portes logiques AND2 de 188 à 58, le nombre de MUX2 :1 de 52 à 36, le nombre de bits aléatoires (de type 1 bit) de 214 à 90 en comparaison avec la version du décodeur stochastique conventionnel. Cela signifie que l'exploitation le décodage dans le domaine exponentiel nous permet de diminuer considérablement la complexité matérielle.

Nous avons également fait une comparaison de la complexité d'un turbo-décodeur stochastique exponentiel APP avec celle d'un décodeur conventionnel SubMAP en virgule fixe présenté par G.Montorsi dans [90]. Comme déjà indiqué dans chapitre 2, cette comparaison est pertinente si et seulement si elle est considérée pour une unité de traitement d'une section de treillis. C'est pourquoi, on considère que chaque LUT (Look-Up Table) est assignée à une unité de ressource matérielle élémentaire comme celles données dans le tableau 3.2. Dans ce cas, 633 et 304 LUT sont respectivement nécessaires pour la version SubMAP en virgule fixe et la version stochastique exponentielle. En outre, le coût d'une section du décodeur stochastique exponentiel en termes de bascules est raisonnable, car il nécessite seulement 592 bascules. Quant à lui, le décodeur SubMAP requiert 1398 bascules. Nous constatons donc qu'un décodage stochastique dans le domaine exponentiel a une complexité comparable à celle d'une version SubMAP. Néanmoins, il faut rappeler encore que la complexité des turbo-décodeurs stochastiques en termes de LUTs et

de bascules n'a pas encore tenu compte la complexité des bits aléatoires.

3.6 Estimation du débit d'un turbo-décodeur stochastique exponentiel

Afin d'évaluer le débit d'un turbo-décodeur stochastique exponentiel, nous reprenons l'expression donnée au chapitre 2 :

$$D_s = f_{clk} \cdot \frac{k}{DC_{max}} \quad \text{[Mbits/s]} \tag{3.15}$$

Dans l'architecture d'un turbo-décodeur stochastique exponentiel (k=200, $R = 1/3$), le nombre de cycles de décodage maximum est égal à 32k DC. Ce décodeur reçoit en parallèle les symboles démodulés et fournit tous les symboles décodés. Nous faisons l'hypothèse réaliste d'un fréquence sur une cible ASIC pouvant atteindre 500MHz. Alors, le débit de ce décodeur est :

$$D_s = 500.\frac{200}{32.000} = 3,125 \quad \text{[Mbits/s]} \tag{3.16}$$

Nous constatons que le débit d'un turbo-décodeur stochastique exponentiel est plus grand que celui d'un décodeur stochastique conventionnel (0,4 Mbps). Ce résultat s'explique par le fait que le décodeur exponentiel exige moins de cycles qu'un décodeur conventionnel. Cela signifie un turbo-décodeur stochastique exponentiel est non seulement compétitif en termes de complexité matérielle, mais aussi en termes de débit. Néanmoins, ce débit reste encore de l'ordre du Mbits/s. Par conséquent, d'autres solutions doivent être étudiées pour réduire le nombre de cycles de décodage et donc augmenter le débit de décodage. Les sections suivantes nous amène à une solution complémentaire, qui permet de diminuer significativement le nombre de cycles : la solution *multi-flux*.

3.7 Décodage stochastique en parallèle ou *multi-flux*

L'approche de décodage stochastique en parallèle a été implémentée avec succès pour décoder des codes LDPC par S. Sharifi Tehrani durant ses travaux de thèse [16]. Cette approche est associée à des techniques de mémoires EM afin de résoudre le problème de corrélation. Ainsi, l'auteur a réussi à construire un décodeur stochastique atteignant un débit de l'ordre du Gbps. Cette section va aborder une nouvelle approche de décodage stochastique - l'approche *multi-flux* qui est reliée à un autre type de mémoire. Cette technique a été introduite par M. Arzel *et al.* [93] pour décoder des codes Cortex. Les auteurs ont montré que cette nouvelle technique était naturellement robuste pour supprimer le problème de corrélation. L'application de cette approche dans la construction des modules d'un turbo-décodeur stochastique est détaillée. Finalement, nous estimons le débit de décodage d'un turbo-décodeur stochastique multi-flux en fonction du nombre de flux.

3.7.1 Coût des techniques de mémoires en termes d'efficacité architecturale

L'approche de décodage stochastique est souvent associée à la technique de mémoires comme des EMs [76]. Chaque mémoire EM est une structure complexe qui associe une bascule JK avec

un registre à décalage afin de résoudre le problème de corrélation. Cependant, pour réduire efficacement le problème de blocage, la profondeur de chaque EM - registre à décalage - doit être suffisamment large. Typiquement, la taille de chaque EM est généralement de 32 (ou 64) bascules [74]. Dès lors, pour une sélection aléatoire d'un bit regénératif à partir de chaque EM, il faut 5 bits (ou 6 bits) aléatoires d'adresse. La technique de mémoires EM est coûteuse lorsqu'un grand nombre de EM est requis. Les mémoires TFM [88] et MTFM [4] qui sont plus efficaces que des mémoires EM ont été proposées pour remplacer les mémoires EM. Un décodeur basé sur des mémoires TFM a des performances de décodage similaires mais requiert seulement 40% de la complexité sur ASIC par rapport à un décodeur basé sur des mémoires EM [88]. Les résultats d'implémentation sur ASIC ont démontré que la technique TFM était prometteuse pour un décodage en parallèle à très faible complexité. Néanmoins, chaque mémoire TFM nécessite encore des registres à décalage de 8 bits, des générateurs aléatoires, des comparateurs et des additionneurs.

Les mémoires EM et TFM appliquées au décodage des codes LDPC sont initialisées à des probabilités qui sont partiellement extraites à partir des sorties du canal [16]. Cette étape d'initialisation permet aux décodeurs stochastiques de codes LDPC de converger rapidement, et donc, d'obtenir un meilleur débit de décodage. Néanmoins, pour le décodage stochastique de turbo-codes, des probabilités récurrentes ou des probabilités extrinsèques sortantes ne sont pas disponibles dès le départ pour ce décodeur. C'est pourquoi, les mémoires EM associées doivent être équiprobablement initialisées. De plus, un turbo-décodeur stochastique possède en pratique un grand nombre de probabilités récurrentes et de probabilités extrinsèques. Alors, le décodeur stochastique exige un grand nombre de DCs pour sortir l'équiprobabilité. Par conséquent, il ralentit significativement la convergence du décodeur et réduit le débit de décodage stochastique. Pour résoudre ce problème, M. Arzel [93] a proposé une nouvelle architecture de mémoire afin de remplacer les mémoires EM : l'approche *multi-flux*. L'auteur a prouvé que cette nouvelle approche de mémoire était robuste à tout problème de corrélation même pour des architectures possédant un grand nombre d'états inconnus.

3.7.2 Architecture *multi-flux*

L'architecture originale de division de deux flux stochastiques basée sur une mémoire 32-bit EM est illustrée dans la figure 3.16. Chaque probabilité d'entrée p_i est représentée par un flux stochastique s_i. Elle est conjointement traitée avec d'autres probabilités par les modules A, B, C ou une mémoire EM pour éviter le problème de corrélation.

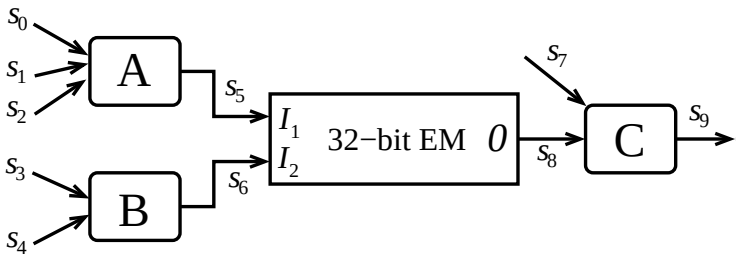


FIGURE 3.16 – Architecture basée sur une mémoire EM traitant des probabilités représentées par des flux stochastiques.

A chaque instant, lorsqu'une corrélation arrive, la mémoire EM fournit aléatoirement un bit à partir d'un jeu des bits régénératifs. Cependant, il est possible de générer un bit régénératif à partir d'autres flux stochastiques indépendants qui représentent la même probabilité. Pour que les flux stochastiques concurrents soient décorrélés, le nombre de flux concurrents doit être suffisamment grand. Cette nouvelle approche est différente des techniques de mémoires EM ou TFM car elle permet d'utiliser un bit régénératif courant au lieu d'un ancien bit stocké dans des mémoires EM et TFM. Ainsi, des états inconnus peuvent être régénérés.

Le bit régénératif doit être aléatoirement choisi à partir de ces flux concurrents de la même manière que pour une mémoire EM. La nouvelle structure doit donc posséder un mécanisme de sélection de type aléatoire des bits, ce qui introduit la diversité dans les flux stochastiques concurrents. Ainsi, l'architecture basée sur des mémoires EM peut être remplacée par une autre architecture, dite *multi-flux*, comme décrite figure 3.17(a). Dans cette nouvelle architecture, tous les flux et les éléments logiques sont dupliqués ρ fois ($\rho > 2$) et la sélection aléatoire des bits est manipulée par un mélangeur (*shuffler* en anglais).

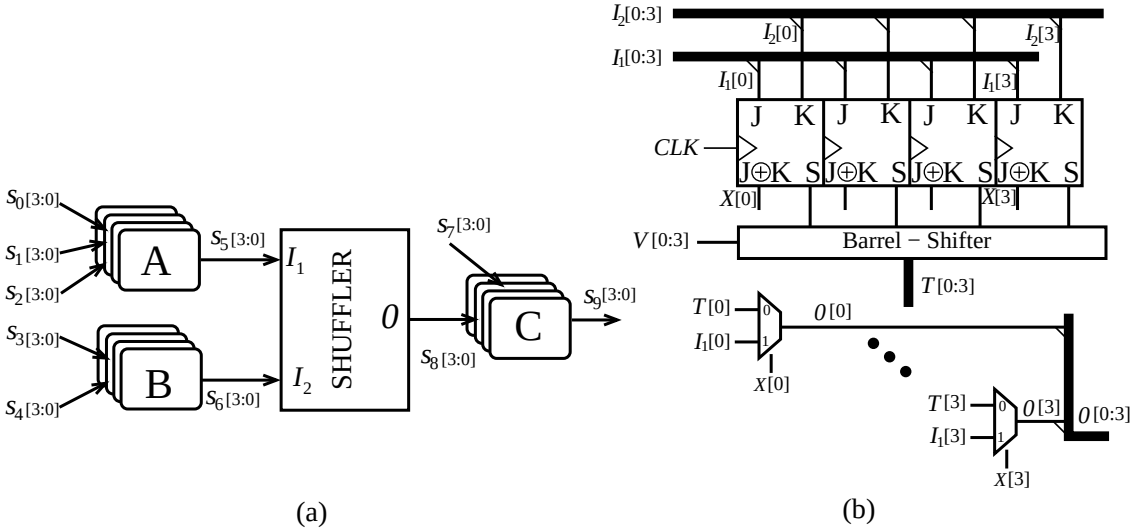


FIGURE 3.17 – Architecture avec probabilités représentées par 4 flux stochastiques traités par des modules logiques en parallèle à travers un mélangeur [93]. (a) Architecture multi-flux. (b) 4 flux stochastiques représentant une même probabilité traités par un mélangeur.

Ce mélangeur est composé de ρ bascules JK, d'un "*barrel-shifter*" de ρ-bits entrée-sortie et de ρ multiplexeurs ($\rho = 4$ en figure 3.17). Les bascules JK (figure 3.18) effectuent des normalisations stochastiques et fournissent les flux $V[0 : \rho - 1]$. Le "*barrel-shifter*" est purement combinatoire, ce qui permet de diversifier spatialement les entrées $V[0 : \rho - 1]$ à chaque cycle et de produire à la sortie des bits mélangés $T[0 : \rho - 1]$. Cela correspond à la propriété de sélection presque aléatoire d'un bit à partir d'une mémoire EM. Les multiplexeurs permettent, quant à eux, de fournir des bits en fonction de la corrélation. Lorsqu'une division stochastique dans la $i^{ème}$ bascule JK produit une corrélation ($J_i \oplus K_i = U[i] = X[i] = 0$), le bit $T[i]$ est choisi comme le résultat de la division. En revanche, lorsque $J_i \oplus K_i = U[i] = X[i] = 1$, la valeur de J_i est considérée comme le résultat. La mémoire MEM de l'architecture JK correspond à une seule bascule où sa valeur - un bit

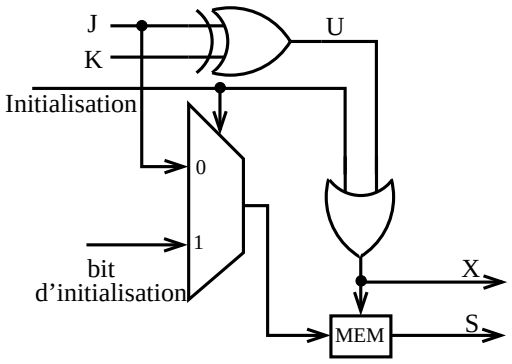


FIGURE 3.18 – Architecture de module JK dans le domaine *multi-flux*.

régénératif - est obtenue à tout instant. Cette caractéristique permet d'éviter les états inconnus.

Le *barrel-shifter* peut être réalisé comme un *crossbar* qui est un type de permutation spatiale programmable. L'implémentation d'un *crossbar* dépend de chaque type de cible : ASIC ou FPGA. L'exemple de la réalisation d'architecture d'un *crossbar* sur une cible ASIC est décrit dans [94]. Le lecteur peut voir l'extension de conception de différentes architectures de *crossbar* dans [95] qui sont utilisées dans des interrupteurs du système téléphonique.

La règle de mélange des bits dans notre *crossbar* peut être choisie de manière déterministe et simple afin d'éviter le coût des bits aléatoires d'adresse additionnel comme c'est le cas dans les mémoires EM. Dans notre cas, un décalage circulaire des bits d'une position à droite est considéré pour tous les différents mélangeurs. Par conséquent, un compteur qui incrémente d'une unité à chaque cycle d'horloge permet le décalage circulaire des bits. Le coût matériel d'implémentation de notre *crossbar* sur un circuit FPGA sera introduit dans la section 3.9.3.

Lorsqu'une probabilité est représentée par ρ flux stochastiques (ou sous une autre définition : ρ vecteurs stochastiques), alors le nombre de DC_{max} est divisé par ρ pour obtenir une précision similaire. Le débit correspondant augmente de ρ fois. Dès lors, cette nouvelle architecture peut atteindre un débit similaire à des architectures de facteur de parallélisme ρ, comme indiqué dans [71]. Dans [93], en utilisant une architecture de 16-flux, l'auteur a démontré que cette approche permettait de produire 90% des bits régénératifs pour un faible nombre de cycles d'exécution. Par comparaison, une architecture de mémoire EM nécessite 10 fois plus de cycles pour fournir ces 90% bits régénératifs. Dès lors, cette approche améliore significativement la convergence du décodeur stochastique.

Néanmoins, la règle basique de décalage circulaire des bits d'une position à droite au sein de chaque *barrel-shifter* n'est pas complètement aléatoire. Des simulations fonctionnelles (présentées en sections suivantes) montrent qu'elle introduit des corrélations dans un environnement de décodage des turbocodes. C'est pourquoi, nous proposons une solution qui consiste à utiliser des entrelaceurs simples après des *barrel-shifters*. Ces entrelaceurs purement combinatoires permettent de diffuser les bits sortants des "*barrel-shifter*s" dans les différents flux de sortie de mélangeurs et alors, de réduire la corrélation issue d'une basique loi de décalage d'un *barrel-shifter*. Grâce à ces entrelaceurs, les performances du décodage stochastique multi-flux de turbocodes sont significativement améliorées.

3.8 Architecture des modules d'un turbo-décodeur stochastique multi-flux

3.8.1 Modèle d'un turbo-décodeur stochastique multi-flux

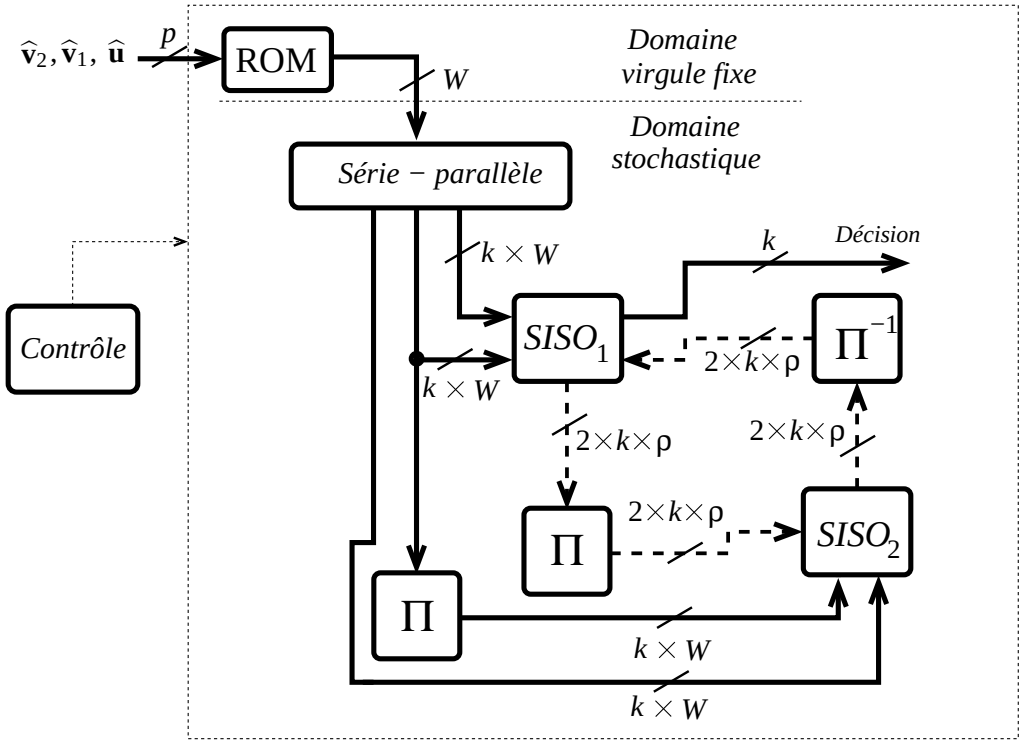


FIGURE 3.19 – Architecture d'un turbo-décodeur stochastique multi-flux.

L'architecture d'un turbo-décodeur stochastique multi-flux est décrite dans la figure 3.19. Chaque décodeur élémentaire SISO correspond à un décodeur convolutif stochastique *multi-flux* APP. Ses entrées sont des probabilités issues du module *série-parallèle*. Chaque décodeur SISO multi-flux se compose de 5 modules constituants : Γ, A/B, Ext, DEC. L'architecture générale d'un décodeur stochastique SISO est similaire à celle de la figure 2.2 du chapitre 2, seul l'échange de messages entre les modules s'effectue dans le domaine *multi-flux*. Le processus de décodage multi-flux a lieu au sein de chaque décodeur SISO *multi-flux*. Chaque probabilité d'entrée est convertie en ρ flux stochastiques qui se propagent parallèlement pour évaluer des métriques de branche (module Γ), des métriques récurrentes *aller/retour* (module A/B), des probabilités extrinsèques (module *Ext*) et des probabilités *a posteriori* (module *DEC*). L'échange des messages entre les deux décodeurs SISO *multi-flux* se fait par des bus de $2 \times k \times \rho$ bits qui représentent ρ bus de k flux stochastiques de 2 symboles possibles du code simple-binaire. Dans les sections suivantes, nous allons détailler différents modules de chaque section en treillis d'un décodeur stochastique multi-flux SISO.

3.8.2 Module Γ

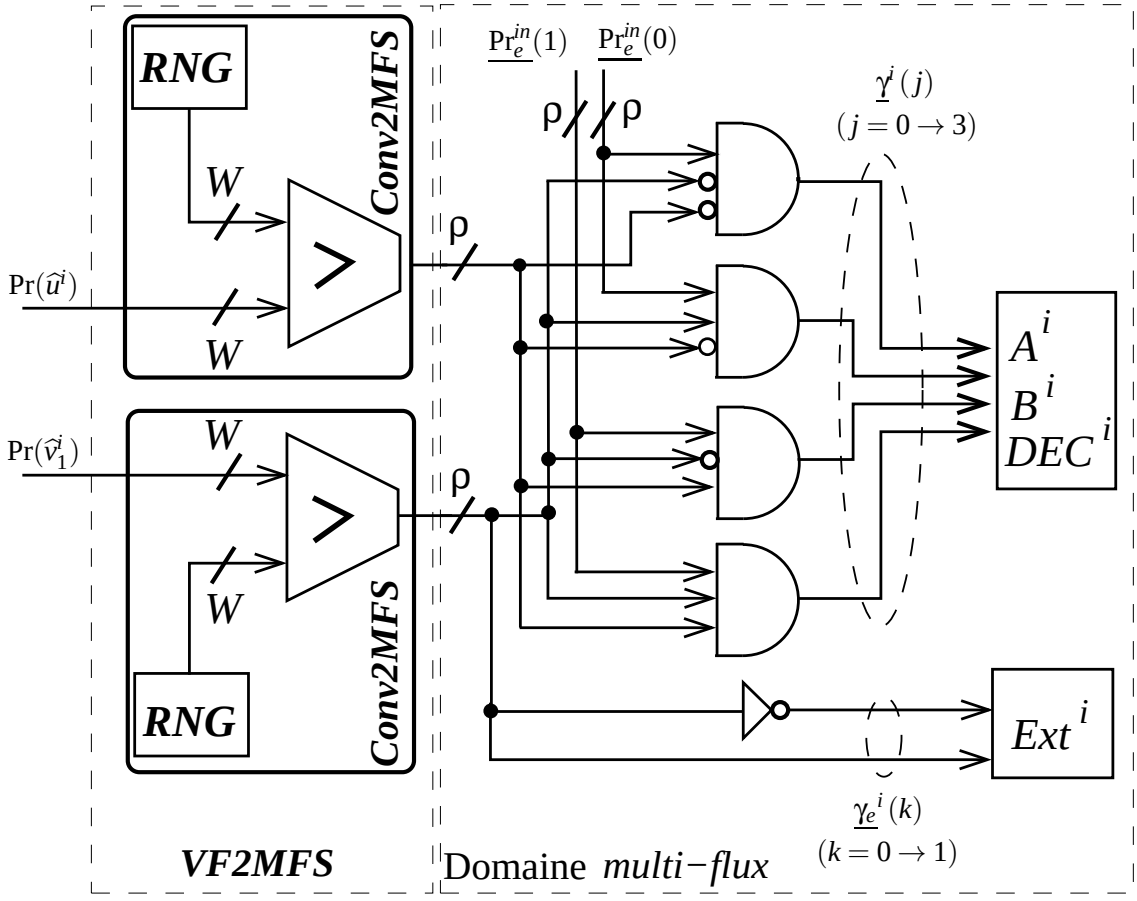


FIGURE 3.20 – Diagramme en blocs du module Γ.

L'architecture du module Γ d'un décodeur stochastique *multi-flux* est détaillée dans la figure 3.20. Les entrées de ce module sont l'ensemble des probabilités représentées sur W bits correspondant au symbole émis d^i et à la redondance y_1^i fournies par le module *série-parallèle*. La sortie du module Γ est constituée par ρ vecteurs de flux stochastiques $\underline{\gamma}^i(j)(j=0\rightarrow 3)$ utilisés pour évaluer des métriques récurrentes au sein du module A/B, des probabilités *a posteriori* au sein du module *DEC*. Une autre sortie du module Γ est constituée par ρ vecteurs de flux stochastiques $\underline{\gamma_e}^i(k)(k=0\rightarrow 1)$ utilisés pour calculer des flux stochastiques extrinsèques au module *Ext*.

La conversion des probabilités en multiples flux stochastiques se déroule dans le bloc *virgule fixe - multiples flux stochastiques* - **VF2MFS**. Cet élément comprend un convertisseur ***Conv2MFS*** associé à la probabilité Pr($\widehat{u}^i$) et un convertisseur ***Conv2MFS*** associé à la probabilité Pr($\widehat{v}_1^i$). Chaque convertisseur est composé de ρ générateurs de séquences (pseudo-)aléatoires ***RNG*** et de ρ comparateurs afin de convertir en parallèle une même probabilité d'entrée en ρ vecteurs de flux stochastiques. La sortie de chaque convertisseur est un bus de ρ bits. Ces vecteurs de flux stochastiques sont transmis au domaine *multi-flux* pour le calcul des métriques de branche.

Chaque porte logique AND2 de ρ bits a deux entrées de ρ bits et une sortie de ρ bits. Elle est construite par ρ portes logiques conventionnelles AND2. Chaque porte logique AND2 conven-

tionnelle comprend deux entrées de 1 bit et une sortie de 1 bit. Ce principe s'applique à tous les autres éléments logiques, telles que des portes XOR, MUX2 :1, MUX8 :1, OR,... et des bascules JK. La connection entre deux éléments logiques du domaine *multi-flux* est assurée par un bus de ρ bits traités en parallèle par des portes logiques conventionnelles.

Pour ce type de décodeur, ρ bits aléatoires sont générés en parallèle à chaque cycle DC. La complexité du module ***VF2MFS*** est donc plus de ρ fois celle du module Γ conventionnel équivalent. Dans ce cas, le nombre de générateurs aléatoires de $W = 7$ bits, de comparateurs et de portes logiques AND3 est multiplié par ρ.

3.8.3 Modules A/B

Ces deux modules calculent récursivement les métriques récurrentes *aller/retour*. L'entrée du module A à l'instant i correspond aux ρ vecteurs de flux stochastiques $\underline{\gamma}^i(j)(j = 0 \rightarrow 3)$ fournis par le module Γ et aux ρ vecteurs de flux stochastiques $\underline{\alpha}^i(k)(k = 0 \rightarrow 7)$ adjacentes à l'instant précédent. La sortie de ce module est constituée par les ρ vecteurs de flux stochastiques $\underline{\alpha}^{i+1}(l)(l = 0 \rightarrow 7)$ qui sont utilisés par le module *Ext* et par le module *DEC*. La connection entre les portes logiques et les bascules est réalisée par des bus de ρ bits.

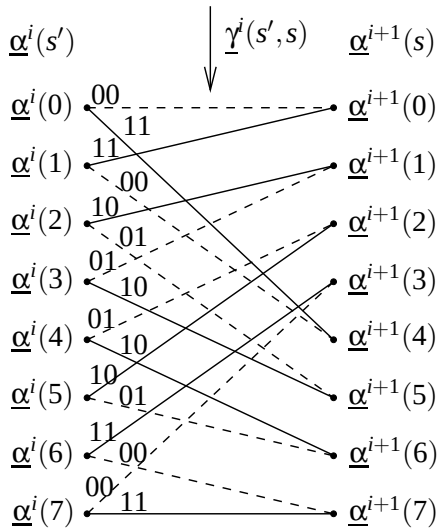


FIGURE 3.21 – Représentation d'une section de treillis.

La figure 3.22 montre l'architecture du module A par un traitement *multi-flux* pour le calcul de ρ métriques récurrentes $\underline{\alpha}^i(1)$. Comme expliqué dans le chapitre 2, nous décomposons ce module en deux sous-modules. Le premier sous-module (figure 3.22(a)) calcule en parallèle les ρ nouvelles métriques $\underline{\widehat{\alpha}}^{i+1}(1)$. Ce calcul est obtenu en se basant sur une section de treillis comme le montre la figure 3.21. Ce sous-module est similaire à celui présenté dans le chapitre 2. Seul la connection entre les portes logiques diffère et correspond à un bus de taille ρ. Ce module requiert ρ bits aléatoires pour additionner en parallèle les métriques récurrentes. La complexité du premier sous-module est ρ fois celle d'un traitement conventionnel. Le second sous-module effectue la normalisation de ces nouvelles métriques récurrentes. Au sein de ce sous-module, les

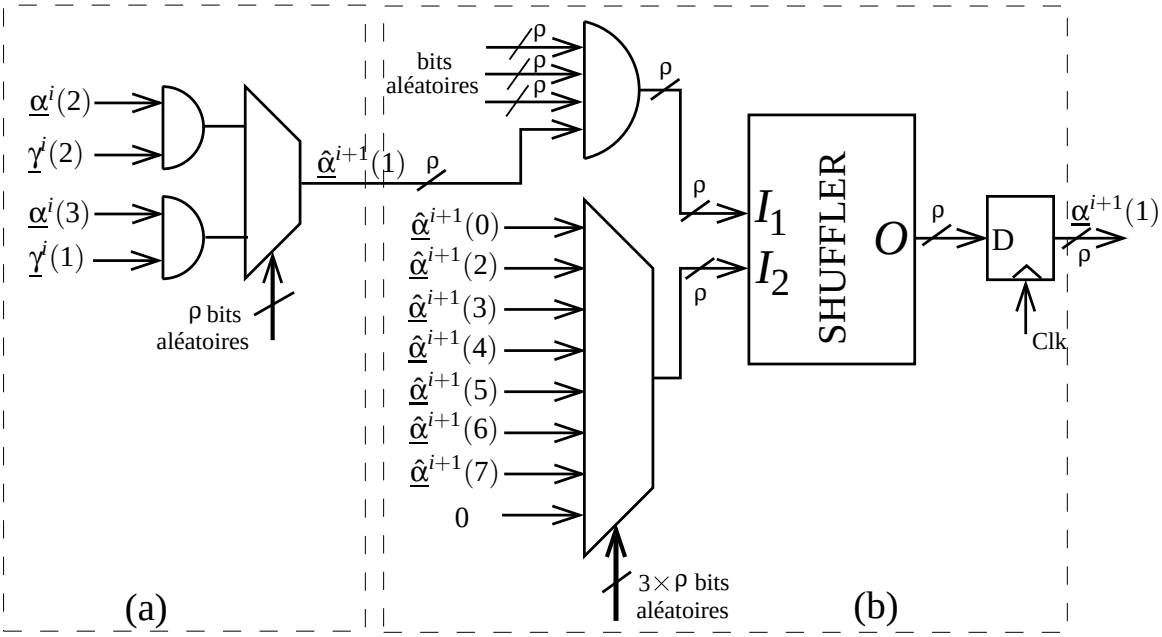


FIGURE 3.22 – Schéma de blocs du module *A* pour un traitement *multi-flux* du calcul de $\underline{\alpha}^{i+1}(1)$. (a) Calcul de nouvelles métriques récurrentes *aller*. (b) Normalisation de nouvelles métriques récurrentes *aller*.

ρ mémoires EM et les ρ vecteurs de 5 bits sont remplacés par un mélangeur de profondeur ρ. Chaque mélangeur est associé à un état de treillis et permet de décorréler les flux stochastiques par une normalisation. Par conséquent, chaque section de treillis a besoin de 8 mélangeurs de profondeur ρ bits. Ce module est répété afin de calculer toutes les métriques récurrentes *aller* et *retour*.

3.8.4 Module *Ext*

La figure 3.23 présente l'architecture du module *Ext* pour un traitement *multi-flux*. L'entrée du module *Ext* comprend ρ vecteurs de flux stochastiques $\underline{\alpha}^i(j)(j=0\rightarrow 7)$, ρ vecteurs de flux stochastiques $\underline{\beta}^{i+1}(k)(k=0\rightarrow 7)$ et ρ vecteurs de flux stochastiques $\underline{\gamma_e}^i(l)(l=0\rightarrow 1)$. Ce module fournit ρ vecteurs de flux stochastiques $\underline{\Pr_e^{out}}(m)(m=0\rightarrow 1)$ qui correspondent à deux probabilités extrinsèques. Il se décompose en deux sous-modules : calcul de nouvelles probabilités extrinsèques (figure 3.23(a)) et normalisation de nouvelles probabilités extrinsèques 3.23(b)). Le premier sous-module est similaire à celui d'un décodeur SISO conventionnel (figure 3.23(a)). il nécessite ρ MUX8 :1 et ρ vecteurs de 3 bits aléatoires. Il fournit en parallèle ρ flux stochastiques correspondant à ρ probabilités extrinsèques $\underline{\Pr_e^{out}}(0)$ et ρ flux stochastiques extrinsèques $\underline{\Pr_e^{out}}(1)$. Le second sous-module effectue la normalisation des flux stochastiques extrinsèques. Au cours de cette étape, deux mélangeurs de taille ρ bits sont requis. Chacun est associé à un flux stochastique extrinsèque. Par ce biais, les mémoires EM et les vecteurs de 5 bits aléatoires d'adresse sont inutiles.

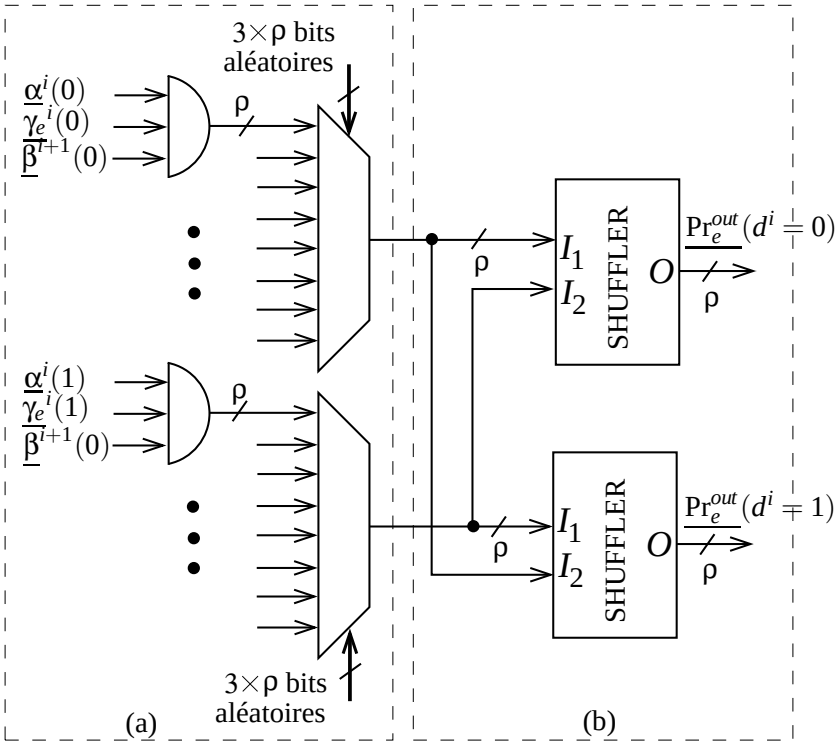


FIGURE 3.23 – Schéma de blocs du module *Ext* pour un traitement *multi-flux*. (a) Calcul des nouvelles probabilités extrinsèques. (b) Normalisation des nouvelles probabilités extrinsèques.

3.8.5 Module *DEC*

Le rôle du module *DEC* est d'évaluer les probabilités *a posteriori* et de produire le symbole décodé. Il nécessite ρ vecteurs de flux stochastiques $\underline{\alpha}^i(j)(j=0\rightarrow 7)$ et ρ vecteurs de flux stochastiques $\underline{\beta}^i(k)(k=0\rightarrow 7)$ fournis respectivement par les modules *A* et *B*, et ρ vecteurs de flux stochastique $\underline{\gamma}^i(l)(l=0\rightarrow 3)$ produits par le module Γ. L'architecture du module *DEC multi-flux* est similaire à celle d'un traitement conventionnel et est décrite dans la figure 3.24. Ce module traite en parallèle ρ bits. La complexité du module *DEC* pour un traitement *multi-flux* s'avère alors ρ fois plus élevée que celle du module *DEC* classique, en termes d'opérations logiques, de bits aléatoires et de compteurs.

En outre, ce module *DEC* contient ρ compteurs up/down saturés fournissant en parallèle ρ bits de signe qui représentent également les ρ symboles simples binaires décodés correspondants. Afin d'améliorer la prise de décision d'un décodeur multi-flux, nous proposons d'utiliser un compteur **Z** de *z* bits. Ce compteur est initialisé pour contenir la valeur 0. Lorsque le nombre de cycles DC_{max} est atteint, le décodage s'achève. ρ bits de signe sortants de ρ compteurs up/down saturés sont récupérés en parallèle par le compteur **Z**. Ce compteur accumule tous les bits de signe. Lorsqu'un bit de signe est égal à "1", le compteur **Z** est incrémenté d'une unité. Il décroît d'une unité lorsqu'un bit de signe est égal à "0". Le nombre de bits *z* de ce compteur est choisi pour s'assurer que $\rho \leqslant 2^{z-1}$. Finalement, le compteur **Z** vérifie la valeur du bit de signe pour déterminer le symbole le plus vraisemblable. Si le bit de poids fort (en anglais *Most Significant Bit* - MSB) est égal à "0", le symbole émis est $d^i = 1$, sinon, $d^i = 0$.

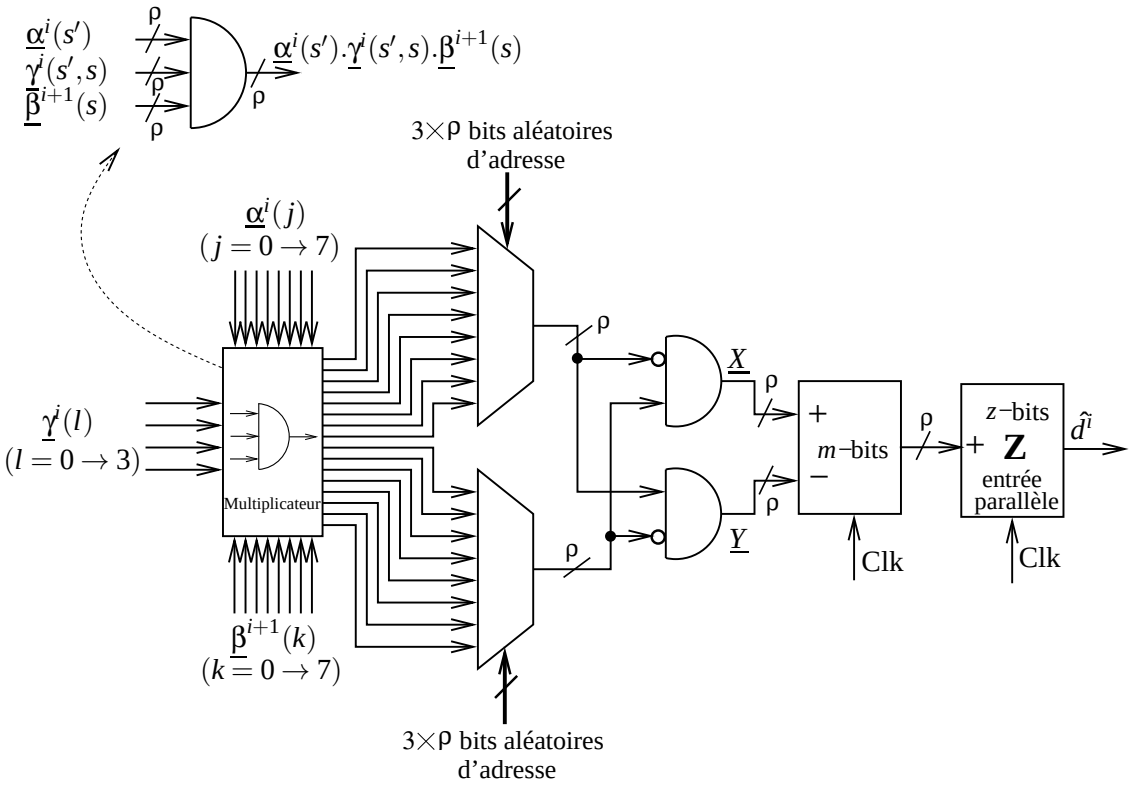


FIGURE 3.24 – Diagramme de blocs du module *DEC* en domaine *multi-flux*.

3.9 Performance de décodeurs stochastiques multi-flux

Cette section présente les performances d'un décodeur convolutif stochastique multi-flux APP et celles d'un turbo-décodeur stochastique multi-flux. Dans un premier temps, des performances en termes de TEB sont présentées. Puis, l'estimation de la complexité matérielle d'un décodeur stochastique multi-flux est donnée. Enfin, une évaluation de débit potentiel d'un turbo-décodeur stochastique multi-flux est fournie. Afin de valider notre architecture et de la comparer avec d'autres architectures de décodeurs stochastiques ou conventionnels, certains paramètres sont retenus. Le couple de bits de quantification des symboles reçus à partir du canal suit le format ($p=7$, $q=4$). La longueur des générateurs de séquences aléatoires est de taille $W=7$ bits. Les compteurs du modules *DEC* sont des compteurs up/down saturés sur 3 bits. Le coefficient *NDS* et des mémoires EMs de longueur de 32 bits sont réutilisés pour comparer notre architecture multi-flux. La transmission correspond à une modulation BPSK sur un canal AWGN. Nous considérons un décodeur stochastique multi-flux avec 32 flux de décodage en parallèle.

3.9.1 Performances d'un décodeur convolutif stochastique multi-flux APP

La figure 3.25 présente les performances de décodage stochastique multi-flux d'un code convolutif (k=200, $R=1/2$) en termes de TEB à partir des simulations fonctionnelles. La courbe bleue correspond au cas d'utilisation des entrelaceurs an sein des mélangeurs. La courbe verte

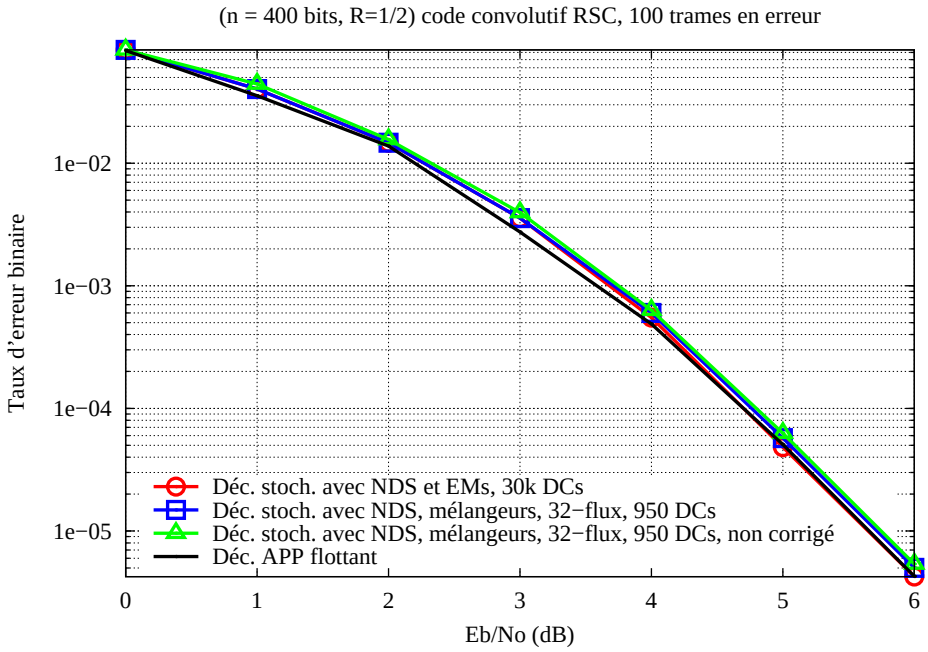


FIGURE 3.25 – Performance du décodage stochastique multi-flux pour un code convolutif RSC (k=200, $R = 1/2$).

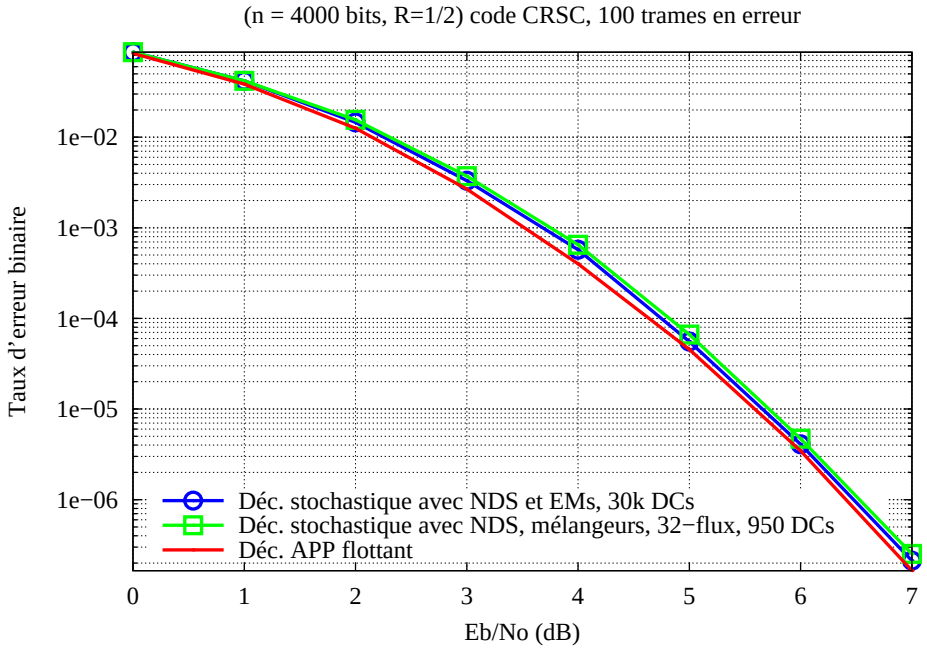


FIGURE 3.26 – Performance du décodage stochastique multi-flux pour des codes convolutifs RSC (k=2000, $R = 1/2$).

représente les performances sans entrelaceur. Les résultats de performance de ces deux décodeurs sont comparés avec ceux d'un décodeur stochastique conventionnel et d'un décodeur exploitant

l'algorithme MAP de référence en virgule flottante. Pour des codes convolutifs, le facteur $\psi(\sigma)$ de *NDS* est toujours égal à 1,0 pour toutes les valeurs σ. Le coefficient $\Omega = \sigma_0^2 = 1,0$.

Le décodeur stochastique multi-flux correspondant à la courbe verte obtient des performances proches de celles d'un décodage stochastique conventionnel et d'un décodage en virgule flottante. En fait, des entrelaceurs ajoutés dans des mélangeurs ne changent pas les performances de décodage pour 32-flux. La courbe verte et bleue sont donc similaires. De plus, l'approche 32-flux permet de gagner en vitesse de convergence. En effet, le décodeur 32-flux nécessite seulement 950 cycles par rapport à 30k DCs d'un décodeur stochastique conventionnel. Cela signifie que notre architecture multi-flux a un gain d'environ 32 fois en termes de débit obtenu en garantissant une performance similaire par rapport à celle d'un décodeur stochastique conventionnel.

La figure 3.26 montre également les performances de décodage stochastique d'un code convolutif de taille plus grande ($k = 2000$, $R = 1/2$). Nous constatons que des 950 cycles DCs sont suffisants pour que le décodeur stochastique 32-flux converge et fournit des performances proches de celles du décodage stochastique de référence. Or, le décodeur stochastique de référence requiert 30k DC pour converger. Cela confirme que l'approche multi-flux donne des résultats prometteurs en termes de vitesse de convergence pour le décodage stochastique des codes convolutifs.

3.9.2 Performance de turbo-décodeurs stochastiques multi-flux

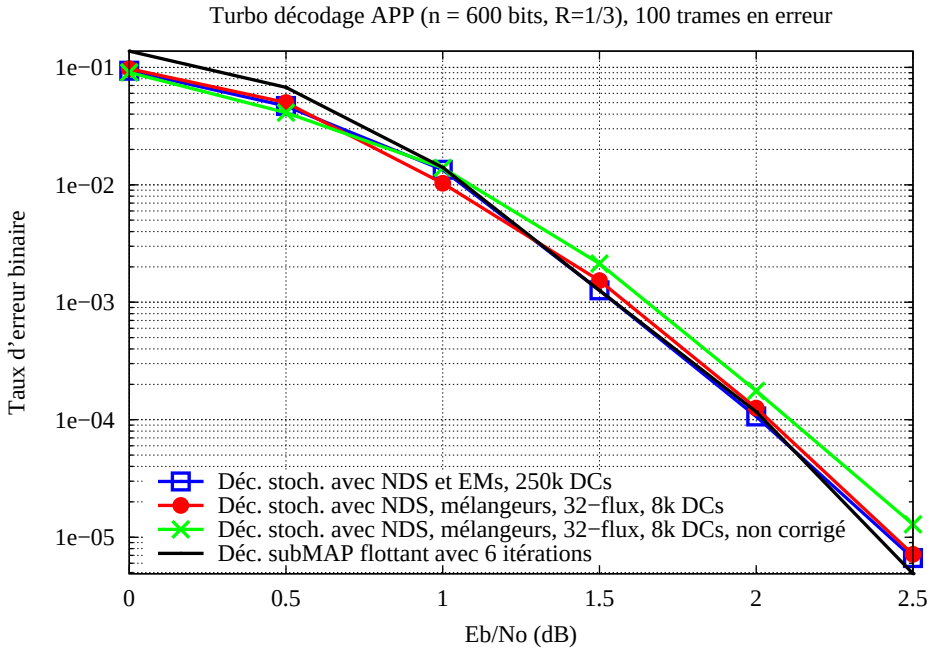


FIGURE 3.27 – Performance du décodage stochastique multi-flux pour des turbocodes convolutifs (k=200, $R = 1/3$).

La figure 3.27 donne les performances de décodage d'un turbocode de taille $k = 200$ et de rendement $R = 1/3$. Les coefficients du facteur de correction *NDS* sont conservés comme montré au chapitre 2. Le coefficient Ω est égal à $\sigma_0^2 = 1,5$. $\psi(\sigma)$ = 1,0 pour RSB = (0.0dB $\rightarrow$ 2.0dB) et

$\psi(\sigma) = 1,2$ pour un grand RSB ($> 2dB$).

Les performances du décodage stochastique multi-flux sont comparées avec celles d'un décodage exploitant l'algorithme SubMAP avec 6 itérations en virgule flottante et avec celles d'un décodage stochastique conventionnel. Des décodeurs 32-flux sont considérés dans notre étude. La courbe verte ,notée *non corrigé*, présente la performance de décodage multi-flux sans entrelaceurs dans les mélangeurs. La courbe rouge correspond à l'ajout d'entrelaceurs au coeur des mélangeurs. Elle démontre l'influence intéressante des entrelaceurs sur les performances de décodage stochastique multi-flux. Par comparaison avec les performances de décodage stochastique conventionnel (soit la courbe bleue), le décodeur stochastique multi-flux exploitant l'architecture de mélangeur proposée par M.Arzel a une légère dégradation d'environ 0,1dB pour un TEB de l'ordre 10^{-5}. Cependant, l'ajout des entrelaceurs améliore les performances de décodage multi-flux. Ils permettent de s'approcher de celles du décodage stochastique conventionnel et d'un décodage en virgule flottante. En outre, l'exploitation de l'approche multi-flux permet de diminuer significativement le nombre de cycles de décodage. Ainsi, ce nombre diminue de 250k DCs pour un décodeur stochastique conventionnel à 8k DCs pour un décodeur stochastique multi-flux. Cela signifie qu'un décodeur stochastique multi-flux est très efficace en termes de cycles de décodage.

3.9.3 Estimation de la complexité d'un décodeur stochastique multi-flux APP

L'augmentation des flux de décodage permet d'améliorer considérablement la vitesse de convergence. Néanmoins, elle rend l'architecture du décodeur plus complexe. Cette section nous donne une estimation de la complexité matérielle d'un décodeur stochastique multi-flux APP en termes de portes logiques élémentaires, de bascules, de bits aléatoires, de compteurs et de "BF - *barrel-shifters*".

Module	Ressources matérielles élémentaires								Bits aléatoires	
	AND2	OR2	XOR2	MUX2	MUX8	compteur	D-FF	BF	7 bits	1 bit
Γ	42.ρ	14.ρ							2.ρ	
A/B	40.ρ	8.ρ	8.ρ	24.ρ	8.ρ		16.ρ	8		56.ρ
Ext	32.ρ	2.ρ	2.ρ	4.ρ	2.ρ		2.ρ	2		6.ρ
Dec	34.ρ+2	1			2.ρ	ρ+1				6.ρ
Total	188.ρ+2	32.ρ+1	18.ρ	52.ρ	20.ρ	ρ+1	34.ρ	18	2.ρ	124.ρ
$\rho = 32$	6018	1025	576	1664	640	33	1088	18	64	3968

TABLE 3.3 – Complexité d'une section d'un décodeur stochastique multi-flux SISO binaire à 8 états avec des mélangeurs.

Le tableau 3.3 résume la complexité d'une section d'un décodeur stochastique multi-flux en fonction du nombre de flux ρ. Chaque "*barrel-shifter*" est sur ρ bits. Chaque module JK d'un mélangeur est composé d'un MUX2 :1, d'une porte logique AND2 et d'une bascule. Ce tableau illustre également la complexité d'un décodeur multi-flux lorsque $\rho = 32$.

En effet, plusieurs solutions sont proposées pour l'implémentation des *barrel-shifters*. Chaque solution donne un coût de complexité correspondante et cette complexité est une fonction du

nombre de ses entrées et de ses sorties. Par synthèse sur FPGA, le coût d'un *barrel-shifter* en termes de LUTs, de bascules et de slices est donné par le tableau 3.4. Dans notre architecture de turbocode, $\rho = 32$ flux est considéré. Alors, le nombre de bascules Flip-Flops et de LUT des *barrel-shifters* pour une section de treillis sont respectivement égaux à 0 et 1728.

Barrel-shifter	LUTs	Flip-Flops	Slices
$\rho = 8$ flux	18	0	6
$\rho = 16$ flux	32	0	10
$\rho = 32$ flux	96	0	31

TABLE 3.4 – Complexité d'un *barrel-shifter* en fonction du nombre de flux implémenté.

Une comparaison directe de la complexité d'un décodeur multi-flux APP avec ρ décodeurs stochastiques conventionnels entièrement parallèles est possible. En parallélisant le décodeur stochastique par un facteur de ρ, la complexité en termes d'éléments logiques, de compteurs est donc multipliée de ρ. En revanche, nous voyons que l'insertion des mélangeurs permet de réduire significativement la complexité matérielle. En termes de bascules, le nombre pour une section peut être réduit de 592*ρ à 34*ρ lorsque la version des mélangeurs est considérée. Dans ce cas, chaque architecture de mémoire JK ne contient qu'une seule bascule au lieu de 32 bascules par rapport à la version conventionnelle. En termes de bits aléatoires, trop de bits aléatoires doivent être générés pour des architectures basées sur des EMs. En revanche, des mélangeurs n'exigent aucun bit aléatoire. Ainsi, le décodeur ρ−flux nécessite seulement 124*ρ bits par rapport à 214*ρ bits de ρ décodeurs stochastiques conventionnels parallèles. Cependant, l'utilisation des mélangeurs nécessite 18 "*barrel-shifters*" de ρ bits. Dès lors, l'approche de décodage multi-flux est un solution architecturale prometteuse en termes de performance atteinte et de complexité matérielle comparable aux caractéristiques d'un décodage entièrement parallèle.

3.9.4 Estimation du débit d'un turbo-décodeur stochastique multi-flux

Comme les solutions proposées précédemment, le débit d'un turbo-décodeur stochastique multi-flux s'exprime :

$$D_s = f_{clk} . \frac{k}{DC_{max}} \quad \text{[Mbits/s]} \tag{3.17}$$

Dans une architecture d'un turbo-décodeur stochastique multi-flux (k=200, $R = 1/3$), le nombre de cycles de décodage maximum est égal à 8k DC. Dans cette architecture, la réception des symboles à partir du module *série-parallèle* et la sortie des symboles décodés s'effectuent en parallèle. Supposons que la fréquence de fonctionnement de ce décodeur est de 500Mhz, alors, le débit de ce décodeur est :

$$D_s = 500 . \frac{200}{8.000} = 12,5 \quad \text{[Mbits/s]} \tag{3.18}$$

Par rapport aux autres techniques, l'approche multi-flux permet d'obtenir un meilleur débit de décodage. Ce résultat est obtenu car le décodeur multi-flux nécessite moins de cycles par rapport à une approche stochastique conventionnelle. Pour une même probabilité, 32 bits d'entrée sont

générés pendant un seul DC au lieu de générer 1 bit pendant 32 DCs. Ceci résulte en une division du nombre de cycles par un facteur de 32. Dès lors, ce décodeur 32-flux fournit un débit 32 fois plus élevé, pour une augmentation des ressources matérielles inférieure à un rapport 32.

3.10 Contribution des deux propositions pour la conception d'un turbo-décodeur à haut débit

Cette section va présenter une contribution à partir des deux approches (*exponentiel* et *multi-flux*) pour la conception d'un nouveau décodeur stochastique : décodeur *stochastique exponentiel multi-flux*. La caractérisation de l'architecture intégrant ces deux propositions s'effectue par ses performances de décodage, sa complexité matérielle et le débit atteint. On considère que ce décodeur possède $\rho = 32$ flux de décodage en parallèle.

3.10.1 Performance d'un décodeur convolutif stochastique exponentiel multi-flux APP

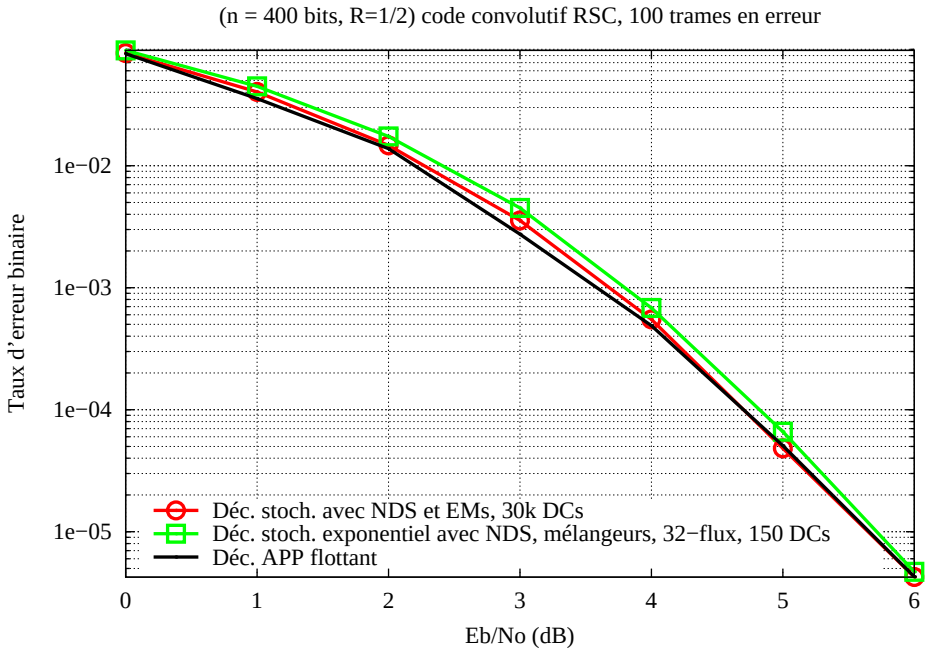


FIGURE 3.28 – Performance de décodage stochastique exponentiel multi-flux pour un code convolutif RSC (k=200, $R = 1/2$).

La figure 3.28 décrit les performances de décodage d'un code convolutif (k=200, $R = 1/2$). La courbe verte correspond aux performances d'un décodage stochastique exponentiel multi-flux. Elles sont similaires à celles d'un décodage stochastique conventionnel (la courbe rouge) ou à celles d'un décodage en virgule flottante de référence (la courbe noire) pour un TEB = 10^{-5}. Par ailleurs, l'association de ces deux approches permet d'obtenir un grand gain en termes de cycles de décodage. En effet, le décodeur stochastique exponentiel multi-flux nécessite seulement 150 DCs

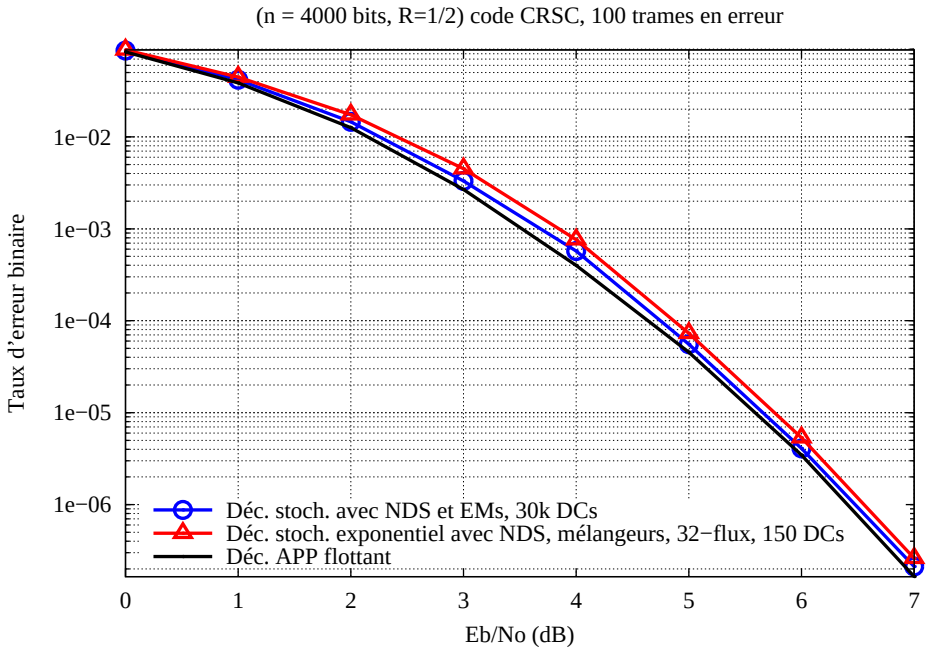


FIGURE 3.29 – Performance de décodage stochastique exponentiel multi-flux pour un code convolutif RSC (k=2000, $R = 1/2$).

en comparaison des 30k DCs d'un décodeur stochastique conventionnel. La figure 3.29 montre également les résultats pour le décodage d'un code convolutif (k=2000, $R = 1/2$). Dans ce cas, le décodeur stochastique exponentiel multi-flux requiert seulement 150 DCs et a une dégradation de seulement 0,2dB à $TEB = 10^{-6}$ par rapport à celle du décodage en virgule flottante de référence.

3.10.2 Performance d'un turbo-décodeur stochastique exponentiel multi-flux

La figure 3.30 illustre les performances de décodage d'un décodeur stochastique exponentiel multi-flux. Elles sont comparées à celles d'un décodage stochastique conventionnel (courbe bleue) et à celles d'un décodeur SubMAP en virgule flottante de 6 itérations (courbe verte). Nous constatons que le turbo-décodeur stochastique exponentiel multi-flux proposé atteint des performances proches de celles d'un décodage stochastique conventionnel et de celles d'un décodage SubMAP. Il est à noter que le décodeur stochastique exponentiel multi-flux requiert seulement 1k DCs pour obtenir ces performances. Rappelons que le décodeur stochastique conventionnel nécessite 250k DCs. Ce résultat est obtenu par un parallélisme de décodage exponentiel de 32 flux. Dans ce cas, le nombre de cycles DCs est divisé par un facteur 32 par rapport au décodeur exponentiel avec une augmentation de complexité par un facteur $\rho = 32$.

3.10.3 Complexité d'un décodeur stochastique exponentiel multi-flux

Le tableau 3.5 détaille la complexité d'une section d'un décodeur stochastique exponentiel 32-flux APP en termes d'éléments logiques. Cette complexité est à comparer avec celle d'un décodeur composé de 32 décodeurs stochastiques conventionnels entièrement parallèles où la complexité

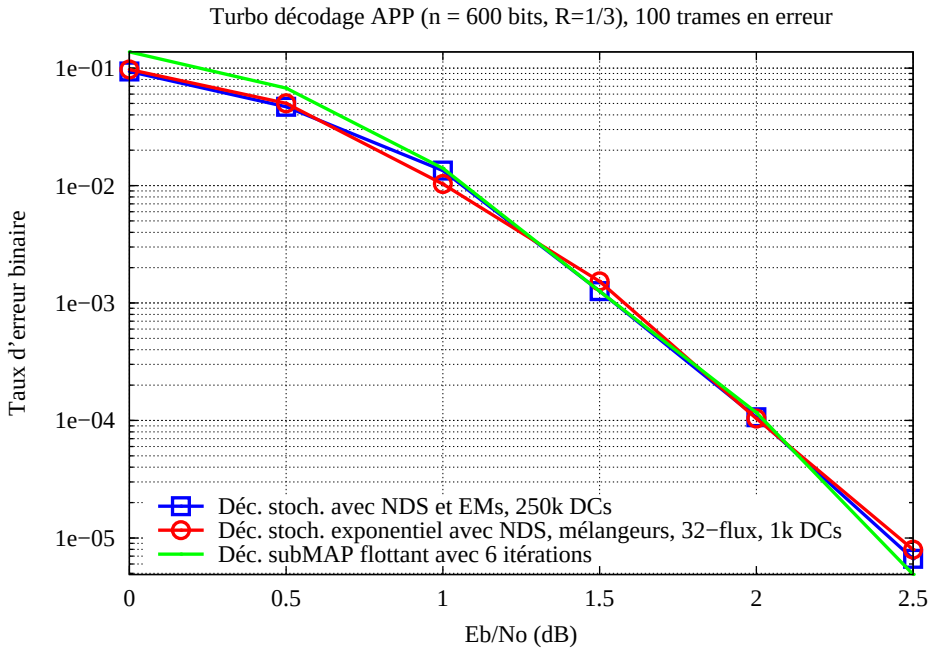


FIGURE 3.30 – Performance de décodage stochastique exponentiel multi-flux pour un turbocode convolutif (k=200, $R = 1/3$).

Module	Ressources matérielles élémentaires								
	NAND2	AND2	OR2	XOR2	MUX2	MUX8	compteur	D-FF	BF
Γ		42.ρ	14.ρ						
A/B	52.ρ		8.ρ	8.ρ	8.ρ			16.ρ	8
Ext	32.ρ		2.ρ	2.ρ	2.ρ			2.ρ	2
Dec	24.ρ	16					ρ+1		
Total	160.ρ	42.ρ+16	32.ρ	18.ρ	18.ρ		ρ+1	34.ρ	18
$\rho = 32$	5120	1360	1024	576	576		33	1088	18

TABLE 3.5 – Complexité d'une section d'un décodeur stochastique exponentiel 32-flux SISO binaire de 8 états en termes d'éléments logiques.

de chaque décodeur conventionnel est décrite dans le tableau 2.1 du chapitre 2. Pour le faire, la comparaison repose sur des implémentations sur FPGA et la complexité d'un décodeur stochastique exponentiel 32-flux est normalisée par un facteur $\rho = 32$. Nous considérons que chaque LUT est associée à une unité de ressource matérielle élémentaire comme celles données dans le tableau 3.5. Par conséquent, 271 LUT sont nécessaires pour implémenter un flux d'une section du décodeur stochastique exponentiel 32-flux. Ce nombre de LUTs est moins que celui d'un décodeur stochastique conventionnel lorsqu'il requiert 313 LUTs par section. De plus, la complexité en termes de bascules par section est fortement réduite de 592 pour la version conventionnelle à 34 pour la version stochastique exponentielle 32-flux.

Module	Bits aléatoires	
	7 bits	1 bit
Γ	$2*\rho$	0
A/B		0
Ext		0
Dec		0
Total	2*ρ	0
$\rho = 32$	64	0

TABLE 3.6 – Complexité d'une section d'un décodeur stochastique multi-flux SISO binaire de 8 états en termes de bits aléatoires.

Le tableau 3.6 résume le nombre de bits aléatoires nécessaires pour chaque module d'une section d'un décodeur stochastique exponentiel 32-flux. Ce nombre de bits aléatoires est normalisé sur 32 flux concurrents. Dans ce cas, un flux stochastique d'une section requiert seulement 2 vecteurs de 7 bits afin de produire les flux stochastiques représentant des probabilités *a priori*. En revanche, un décodeur stochastique conventionnel nécessite également des vecteurs d'un bit aléatoire pour les MUX2 :1 et MUX8 :1. En comparaison avec 228 bits aléatoires d'une section d'un décodeur stochastique conventionnel, nous constatons que le décodeur stochastique exponentiel 32-flux exige très faible nombre de bits aléatoires que 32 décodeurs stochastiques conventionnels entièrement parallèles. En d'autres termes, un décodeur stochastique exponentiel 32-flux possède une complexité raisonnable.

3.10.4 Estimation du débit d'un turbo-décodeur stochastique exponentiel multi-flux

Nous avons montré que le décodeur stochastique exponentiel multi-flux permettait de réduire le nombre de cycles, de 250k DCs à 1k DCs, pour décoder un turbocode (k=200, $R = 1/3$) en garantissant une bonne performance de décodage. Le débit de décodage d'un décodeur stochastique exponentiel multi-flux correspondant est donné :

$$D_s = f_{clk} . \frac{k}{1000} \quad \text{[Mbits/s]} \tag{3.19}$$

Nous supposons que ce décodeur possède une fréquence de fonctionnement de 500Mhz. Alors, l'estimation de débit de décodage exponentiel multi-flux est :

$$D_s = 500 . \frac{200}{1000} = 100 \quad \text{[Mbits/s]} \tag{3.20}$$

Dans ce cas, nous obtenons un débit de l'ordre de 100Mbps. Ce débit est bien supérieur à celui d'un décodage stochastique conventionnel (0,4Mbps) dans les mêmes conditions. Dès lors, un turbo-décodeur stochastique exponentiel multi-flux fournit non seulement une bonne performance mais aussi un grand débit de décodage. A ce stade, nous pouvons augmenter la longueur de la séquence d'information ($k > 200$), par exemple, k = 2000 bits pour augmenter d'autant le débit. Par conséquent, nous pouvons atteindre un débit de l'ordre du Gbps sur un circuit ASIC.

3.11 Conclusion

Ce chapitre a introduit deux nouvelles approches pour augmenter le débit d'un turbo-décodeur stochastique. La première approche implique le décodage stochastique dans le domaine exponentiel. Cette technique a pour objectif d'éliminer les opérations addition stochastique à plusieurs entrées qui ralentissent significativement la convergence d'un turbo-décodeur stochastique. En utilisant des conversions exponentielles et logarithmiques, cette technique a permis d'améliorer efficacement la convergence du décodeur. Par ailleurs, l'architecture de turbo-décodeur dans le domaine exponentiel a un coût matériel compétitif par rapport à une architecture de turbo-décodeur SubMAP en virgule fixe.

Turbocode ($k = 200, R = 1/3$)	Performance	Complexité			Nb de cycles	Débit (Mbps)
		LUT	FF	bits aléatoires		
SubMAP flottant 6 Itérations	Ciblée	633	1398	0	2,4k	8
Stochastique Conventionnel	Similaire	313	592	228	250k	0,4
Stochastique Exponentiel	Similaire	304	592	104	32k	3,125
Stochastique 32-flux	Similaire	11651	1088	4096	8k	12,5
Stochastique Exp. 32-flux	Similaire	10384	1088	448	1k	100

TABLE 3.7 – Evolution de différentes architectures de turbo-décodeur en termes de performance, de complexité, de nombre de cycles et de débit.

La seconde approche, *multi-flux*, proposée par M. Arzel a été étudiée. Nous avons montré que cette technique pouvait être appliquée avec succès pour le décodage stochastique de turbocodes. En représentant une même probabilité par ρ flux stochastiques en parallèle, un décodeur multi-flux permet une montée en débit de ρ fois celle d'un décodeur conventionnel sans dégradation de performance. Néanmoins, l'utilisation de plusieurs flux en parallèle augmente également la complexité matérielle, mais dans un rapport inférieur à ρ.

Une contribution associant ces deux approches est proposée pour la construction d' un décodeur stochastique exponentiel multi-flux. Les caractéristiques de ce décodeur sont données en termes de performance de décodage, de complexité et de débit. Par des simulations fonctionnelles sur un turbocode (k=200,$R = 1/3$), nous démontrons que ce décodeur stochastique exponentiel multi-flux peut réduire fortement le nombre de cycles de décodage, de 250k DCs à 1k DCs. Dès lors, le débit de décodage correspondant s'accroît significativement.

Le tableau 3.7 décrit la comparaison récapitulative de différentes architectures de turbo-décodeur stochastique en termes de performance, de la complexité et du débit. Dans cette comparaison, la complexité de chaque *barrel-shifter* de $\rho = 32$ bits dans les versions multi-flux est exprimée par 96 LUTs (*c.f* le tableau 3.4). Nous pouvons noter que l'application de deux ap-

proches multi-flux et exponentielles permet d'améliorer considérablement le débit de décodage. Cela justifie que l'architecture d'un décodeur stochastique exponentiel multi-flux est une solution pertinente pour l'implémentation numérique d'un turbo-décodeur à haut débit.

CHAPITRE 4

Prototypage d'un turbo-décodeur stochastique

Sommaire

4.1 Introduction

Nous rappelons que l'objectif de cette thèse est l'étude et la mise en œuvre d'un turbo-décodeur stochastique. Nous avons détaillé dans le chapitre 2 l'étude algorithmique d'un turbo-décodeur stochastique. Puis, le troisième chapitre fût consacré à l'étude architecturale qui a abouti à la construction d'un turbo-décodeur stochastique. Dans ce chapitre, nous présentons l'implémentation d'un turbo-décodeur stochastique exponentiel multi-flux ainsi que ses performances. En effet, cette réalisation sur circuit FPGA permet l'évalution et la validation des algorithmes et architectures. La validation du décodeur sur un circuit a donc nécessité, en plus du récepteur, le développement d'un émetteur et d'un émulateur de canal constituant ainsi une chaîne de transmission numérique complète. L'objectif principal de ce chapitre est de décrire la plate-forme intégrant la chaîne de transmission afin de valider le turbo-décodeur stochastique et d'en estimer la complexité, la fréquence de fonctionnement et enfin le débit de décodage.

Dans un premier temps, nous allons présenter l'environnement d'implémentation et la plate-forme FPGA dédiés à notre étude. Ensuite, l'implantation des différents modules de la chaîne sur cette plate-forme est décrite. Pour y parvenir, nous allons choisir un cas d'étude adéquat afin de démontrer la faisabilité de l'architecture. Pour ce cas, nous présentons, les performances du décodeur en termes de TEB, simulées et mesurées sur la plate-forme du prototypage du turbo-décodeur stochastique. Afin d'aboutir à cet objectif, un autre prototypage correspondant à un décodeur convolutif stochastique sera également introduit dans ce chapitre. Les résultats de synthèse obtenus permettent alors d'estimer le débit de décodage associée à la complexité matérielle.

4.2 Métriques associées à l'environnement de prototypage

Dans cette section, nous introduisons brièvement l'environnement d'implémentation et de validation du prototypage.

4.2.1 Objectif du prototypage

4.2.1.1 Fonctionnalité

La recherche dans le domaine des futurs réseaux de communication sans fil accorde un intérêt particulier aux systèmes numériques exploitant des turbocodes. Des études théoriques ont démontré que l'approche stochastique pouvait apporter des résultats prometteurs à la fois en termes de performance et de débit de décodage des turbocodes. Néanmoins, une fois l'architecture du décodeur stochastique validée au travers de tests exhaustifs sur une description en langage C++, des plate-formes matérielles sont indispensables pour les valider dans des contextes réalistes à savoir :

- Canaux réels.
- Contraintes d'implémentation : limitations des ressources matérielles disponibles (unité de calcul et de mémoire) et des débits des interfaces de communications.
- Problèmes de quantification dus à la conversion analogique-numérique et à la représentation en virgule fixe.

Dans ce contexte, le développement de plate-formes de prototypage pour les turbo-décodeurs stochastiques doit permettre de répondre aux questions suivantes :

- Comment évaluer les performances des algorithmes de décodage stochastique pour les turbocodes dans un environnement réaliste ?
- Comment appréhender la complexité de l'architecture proposée ?
- Comment évaluer le débit de décodage de l'architecture ?

La combinaison de ces demandes implique la construction d'un prototype qui atteint un haut débit et une bonne performance de décodage.

4.2.1.2 Performance

L'intégration d'un turbo-décodeur stochastique doit assurer une bonne performance, ou au minimum, une perte raisonnable de performance de décodage par rapport à celle du modèle flottant de référence décrit en langage C++. Concernant la mise en œuvre du décodeur, cette intégration est réalisée à l'aide d'outil de CAO (Conception Assistée par Ordinateur). Le turbo-décodeur stochastique est constitué par différents modules élémentaires (Γ, A/B, Ext, DEC) et ses entrées-sorties sont tous des bus d'information binaire. Des tests de vérification bit à bit entre le modèle C++ et la description sur une plate-forme sont donc possibles. Dans le modèle de référence en C++, le fonctionnement et l'échange d'information entre les modules ne peuvent fournir des informations temporelles de manière statistique. Cela signifie que le temps de propagation est le même pour tous les modules. En revanche, ce n'est pas le cas pour une transmission réelle sur une plate-forme matérielle. Dans ces conditions, les processus de transmission des deux modèles

fournissent des valeurs intermédiaires différentes bien que les décisions finales obtenues soient les mêmes. En conséquence, les simulations ne permettent pas de vérifier pleinement la validité de l'architecture. Pour aboutir à un prototype correct, le test a été réalisé sur plusieurs trames obtenues dans des conditions différentes : trame sans bruit (obtenue à partir d'un $\frac{E_b}{N_0}$ très grand) pour une validation sommaire, trames avec beaucoup d'erreurs ($\frac{E_b}{N_0}$ autour de 0,5 dB) pour vérifier que les erreurs sont proches aux même endroits dans les deux décodeurs. Les simulations du modèle comportemental nous ont permis d'avoir une référence pour le prototype. Néanmoins, il faut noter que seule une trace de taux d'erreur permet de valider le modèle de plate-forme.

4.2.1.3 Débit

L'évaluation des débits est réalisée sur la trame choisie selon le critère de complexité de surface. Par ailleurs, les résultats sont obtenus de manière à avoir des performances de correction, pour chaque configuration, similaires à celles d'une architecture utilisant des virgules flottantes et 6 itérations. Pour cela, le nombre de cycles de décodage est fixé selon la taille de la trame d'information à coder. De plus, les facteurs de correction sont choisis en se basant sur les meilleures performances asymptotiques (voir chapitre 2). Le débit d'information utile est calculé en fonction de la fréquence de fonctionnement f_{clk}, du nombre de cycle de décodage nécessaire DC_{max} et également du nombre de bits d'information k par l'équation 4.1 :

$$D_s = f_{clk} \cdot \frac{k}{DC_{max}} \quad \text{[Mbits/s]} \tag{4.1}$$

4.2.2 Plate-forme

Avant de valider le système par prototypage, il convient de préciser clairement les interfaces entre ce qui va être exécuté sur la plate-forme et ce qui va être exécuté sur le PC hôte. Deux prototypes correspondent à deux différents types de codes - codes convolutifs et turbocodes - seront présentés. La figure 4.1 montre l'environnement d'implémentation pour déterminer les performances de deux décodeurs stochastiques, ainsi que les débits de décodage de turbocode. D_s désigne le débit de décodage, R le rendement du code ($R = 1/2$ pour le code convolutif et $R = 1/3$ pour le turbocode). Et $(p = 7)$ la quantification utilisée sur l'entrée du décodeur stochastique.

L'interface numéro (1) de l'environnement permet de décomposer le prototypage en deux différents sous-environnements. Du côté de l'environnement PC hôte, cette interface produit les paramètres de configuration de notre système grâce à des communications avec le PC hôte. Du côté de l'environnement plate-forme, ce sont des composants qui seront mis en œuvre sur la plate-forme visée : un générateur des bits d'information, un codeur turbo/convolutif, un émulateur de canal, un coeur de décodeur turbo/convolutif stochastique ainsi qu'un détecteur d'erreur. Dans le cadre de la thèse, nous nous focalisons sur le débit de décodage des turbocodes stochastiques, qui est présenté à l'interface numéro (2) de la figure 4.1.

Le PC hôte envoie d'abord les paramètres de configuration génériques au système implémenté sur la plate-forme. En fonction d'un rapport signal sur bruit, l'émulateur de canal ajoute un bruit AWGN (*Additive White Gaussian Noise*) correspondant aux messages générés par le générateur des bits d'information. Dans notre étude, nous avons considéré un mapping de type BPSK

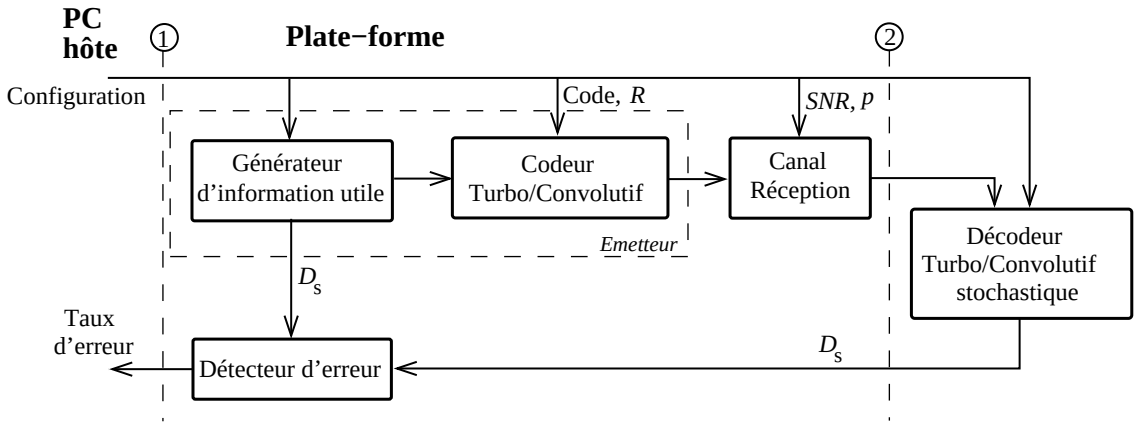


FIGURE 4.1 – Schéma du système de prototypage incluant les interfaces.

(*Binary Phase Shift Keying*). Des conversions binaire-symbole et symbole-binaire sont donc présentes avant et après l'émulateur de canal. Puis, le turbo-décodeur stochastique fournit le résultat de décodage à partir des valeurs bruitées. Enfin, le taux d'erreur binaire calculé à partir du résultat de décodage et le débit sont affichés en temps réel sur l'écran du PC hôte. Ces résultats visualisés sur l'écran permettent de vérifier le débit de la plate-forme et de valider les performances asymptotiques des turbocodes qui y sont programmés.

Afin d'atteindre les débits les plus élevés possibles, il aurait fallu implémenter notre architecture de turbo décodage sur un circuit électronique de type ASIC. Ce travaille aurait permis de nous comparer avec les autres résultats disponibles dans la littérature en termes de débit ou d'efficacité architecturale. Cependant, l'implémentation sur un circuit ASIC est très coûteuse au niveau du temps de conception et la fabrication du circuit ASIC est très complexe. Dans un premier temps, nous avons choisi de l'implémenter sur un circuit FPGA. Cette approche permet d'obtenir rapidement un résultat sur le débit de décodage.

La plate-forme de prototypage utilisée est la carte DN9000K10PCI conçue par la société DiniGroup. Le schéma bloc de la carte est présenté dans la figure 4.2. Cette carte comprend 6 FPGAs Virtex-5 LX330 de la société Xilinx. Les 6 FPGA sont interconnectés par l'intermédiaire de liaisons directes configurables en mode single-end ou différentiel. Ils sont également reliés par un bus principal (*main bus*) pour des opérations de configuration ou pour des transmissions de données. Le bus principal est accessible par un contrôleur PCI permettant de lire ou d'écrire des données sur les FPGA. Des sockets sont également disponibles pour transmettre de grandes quantités de données de mémoire de type DDR (jusqu'à 24Go) connectée aux différents FPGA. La carte dispose de trois interfaces pour établir une connection avec un PC hôte :

– une interface USB, qui permet un accès direct aux six FPGAs par l'intermédiaire d'un bus principal (*main bus*) avec un débit plutôt lent de 80Mbps en lecture (de la carte vers le PC hôte) et 32Mbps en écriture.

– une interface PCI, qui offre un débit maximum de 700Mbps en lecture et 350Mbps en écriture vers un seul des FPGAs de la carte.

– deux interfaces Ethernet, qui offrent chacune un débit de 1Gbps entre le PC hôte et un

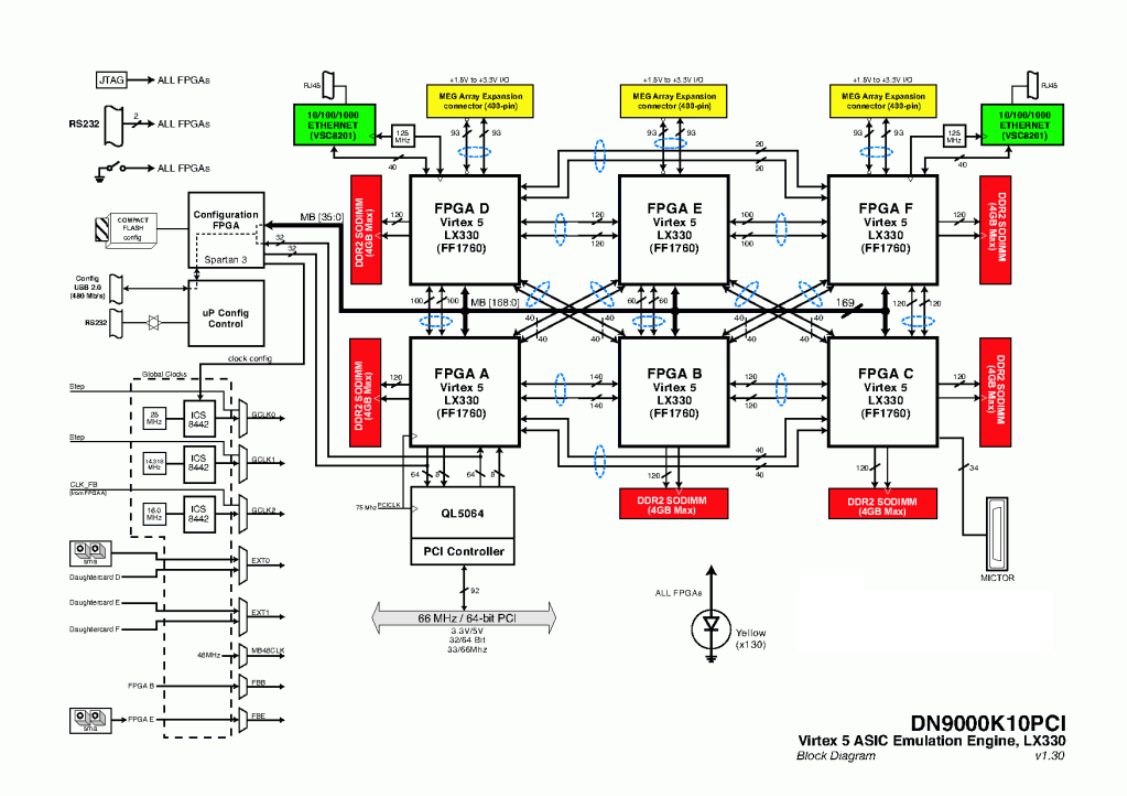


FIGURE 4.2 – Carte de prototypage DN9000K10PCI de la société DINIGROUP.

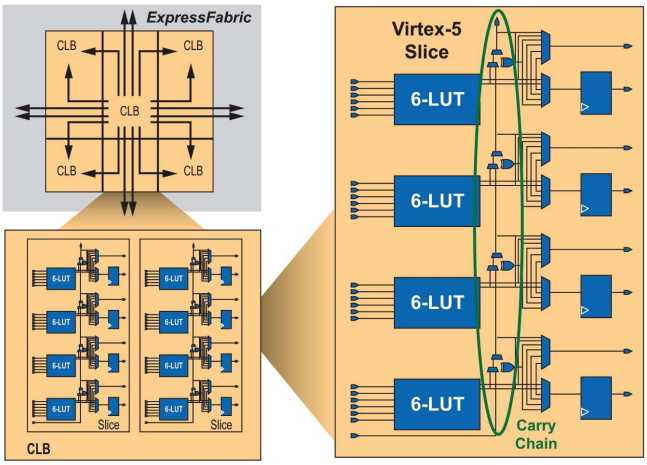


FIGURE 4.3 – Architecture d'un *slice* dans le circuit FPGA Virtex5 LX330.

FPGA, à condition d'implanter dans le FPGA la pile réseau complète du protocole Ethernet.

Chaque FPGA, gravé en technologie 65nm, dispose de 51840 "slices". Chaque *slice* contient 4 LUTs à 6 entrées et 4 bascules FlipFlops. L'architecture d'un *slice* est détaillée en figure 4.3. Le terme LX signifie qu'il est particulièrement optimisé en termes de ressources logiques et de mémoires embarquées. Chaque bloc reconfigurable CLB est constitué de 2 *slices* possédant chacun 4 LUT et 6 entrées et 4 bascules. En plus du grand nombre de blocs reconfigurables, ce circuit

dispose de mémoires embarquées (BRAM) - 288 blocs de RAM de 36Kbits (soit environ 10Mbits de RAM) - et également de blocs DSP48 -192 blocs DSP48 - pour optimiser l'utilisation des LUT. Le Virtex-5 LX330 intègre également un bloc de gestion des horloges qui permet une répartition d'horloges parfaite au sein du circuit.

Dans le cadre de cette thèse, la mise en œuvre du système est considérée à l'aide des environnements de CAO de Xilinx (ISE, XST etc). Pour implémenter une architecture sur le circuit FPGA, le fichier *bitstream*, généré à l'issue du flot de conception, est chargé dans le circuit FPGA via une interface USB à partir du PC hôte. Plus d'informations sont disponibles dans le document technique de la carte [96].

4.3 Description de la chaîne de transmission numérique

Dans cette section, nous présentons d'abord l'architecture du générateur des bits d'information. Ensuite, la partie émettrice de la chaîne de transmissions est détaillée au niveau architectural. Puis, un émulateur matériel de canal de transmission de type BABG - Bruit Additif Blanc Gaussien (AWGN -*Additive White Gaussian Noise*) est introduit. Des synthèses logiques préliminaires, effectuées avec l'outil XST, ont permis d'estimer la complexité des différents modules et sous-modules constituant la chaîne de transmission complète.

4.3.1 Générateur des données binaires

Le générateur des données binaires est conçu à partir d'un LFSR (*Linear Feedback Shift Register*). Les LFSR forment une famille de générateurs aléatoires uniformes. Un LFSR est un registre à décalage à rétroaction linéaire qui permet d'engendrer une suite de longueur finie d'éléments binaires. La suite engendrée par un LFSR possède de bonnes propriétés statistiques lorsque les coefficients de rétroaction du LFSR sont bien choisis. Ces coefficients sont représentés par un polynôme caractéristique $P(X)$ qui détermine la longueur de la séquence pseudo-aléatoire. La figure 4.4 détaille la structure du LFSR que nous avons implémentée. Il s'agit d'un registre à décalage de 32 bits. La récursivité du registre est définie à l'aide du polynôme suivant :

$$P(X) = X^0 + X^1 + X^{21} + X^{31} \tag{4.2}$$

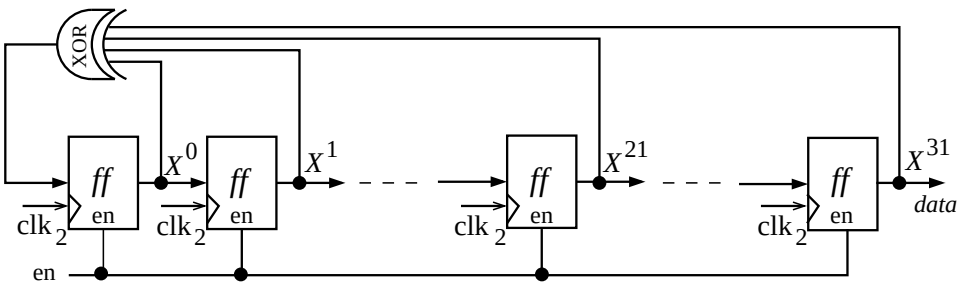


FIGURE 4.4 – Architecture du générateur pseudo-aléatoire.

Ainsi, un certain nombre de bascules FlipFlop et une porte logique XOR sont suffisants pour l'implémentation matérielle d'un tel générateur. Le nombre de bascules utilisé est la fonction

du polynôme et une porte logique XOR est utilisée pour la rétroaction. Un signal *en* permet d'activer ou de suspendre la génération des trames binaires à la sortie. En effet, la valeur du registre a une périodicité qui dépend du polynôme générateur. C'est pourquoi, un LFSR ne génère pas une séquence de données aléatoires. mais pseudo-aléatoire. Si la période est suffisamment grande, nous considérons que les données générées sont aléatoires. Dans notre cas, le polynôme correspond à une séquence pseudo-aléatoire de longueur $4,3 * 10^9$. La donnée de sortie du dernier registre est considérée comme la sortie binaire du générateur aléatoire. Le fonctionnement du générateur est synchronisé par la fréquence d'horloge clk_2.

4.3.2 Architecture de l'émetteur

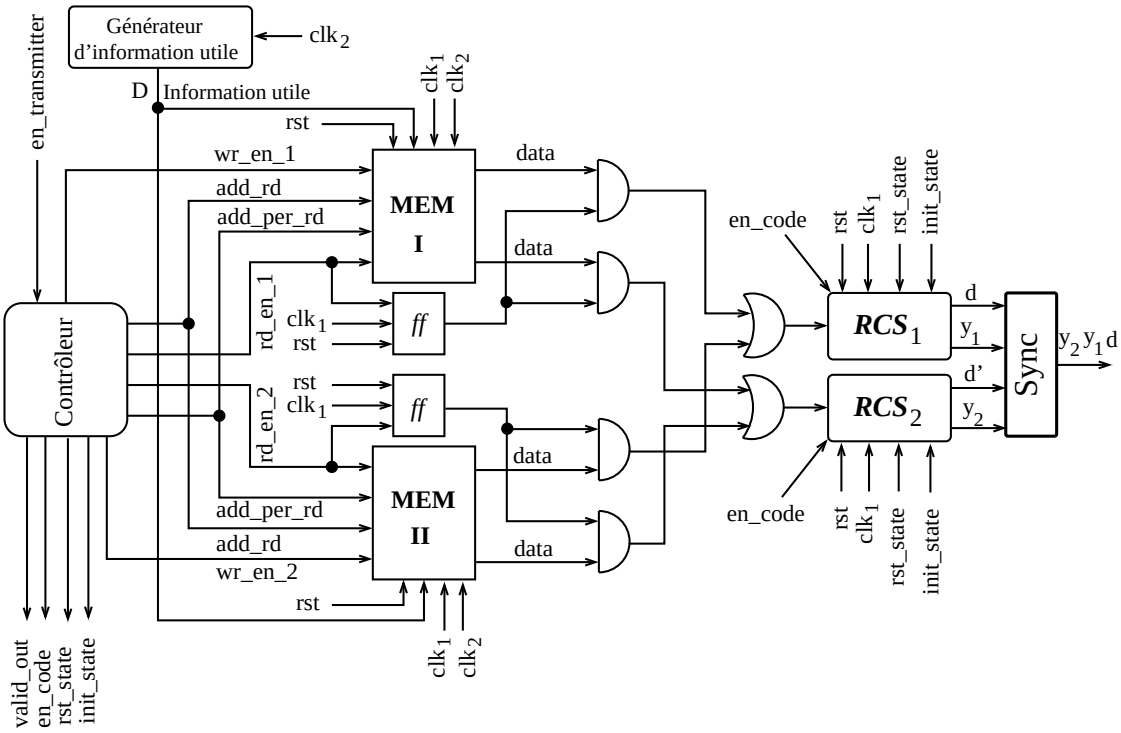


FIGURE 4.5 – Architecture de l'émetteur.

L'architecture globale de la partie émettrice correspondant à un turbo-décodeur dans notre cas est détaillée dans la figure 4.5. Les données numériques sont générées de manière séquentielle, pseudo-aléatoire et indépendante par le générateur de séquences pseudo-aléatoires définies à la sous-section précédente. L'horloge associée à la fréquence de génération est notée clk_2. Les données produites sont ensuite stockées dans les deux mémoires dédiées, notées **MEM I** et **MEM II**. Ces deux mémoires **MEM** ont un fonctionnement dit *ping-pong*. Lorsque la mémoire **MEM I** reçoit des données numériques, l'autre mémoire **MEM II** va fournir les données au turbo-codeur. Cela signifie que le codage des données et la génération de données se produisent en parallèle. La taille de chaque mémoire est égale à la longueur de la trame d'information générée à coder. Ainsi, la taille de chaque mémoire est égale à *k* bits. Chaque mémoire dispose de deux accès, dédiés à la lecture et l'écriture de données, qui sont cadencées par deux horloges distinctes clk_1 et clk_2. Les

opérations de lecture et d'écriture sont effectuées en parallèle. Une bascule est associée à la sortie de chaque mémoire **MEM** afin de synchroniser les sorties de mémoire. Quant à l'entrelaceur associé au codeur, il est équivalent à celui du turbo-décodeur, c'est-à-dire qu'il est remplacé par un vecteur contenant des adresses entrelacées, dont la taille est égale à k bits.

Les signaux de contrôle sont délivrés par un contrôleur (*c.f*, la figure 4.5). Les lectures des mémoires **MEM** sont respectivement contrôlées par les signaux *rd_en_1* et *rd_en_2*. De plus, les données sont respectivement lues à l'aide des adresses dans l'ordre naturel et entrelacées *add_rd* et *add_per_rd*. Pour produire un mot de code avec un rendement $1/3$, deux codeurs convolutifs systématiques récursifs, notés RCS_1 et RCS_2 sont concaténés en parallèle. L'architecture de l'émetteur correspondant au codeur convolutif est similaire à celle présentée dans la figure 4.5, dans laquelle le codeur RCS_2 sera éteint, et le rendement du code est égal à $R = 1/2$. L'architecture détaillée de ces deux codeurs a déjà été introduite dans le chapitre 1. Le signal *en_code* indique aux deux codeurs le début du processus de codage. Nous avons utilisé la technique de fermeture circulaire du treillis. Lorsque le codeur élémentaire RCS_1 est circulaire, alors il a besoin de deux signaux d'initialisation d'état : *rst_state* et *init_state*. Le processus de codage se déroule selon deux étapes successives. Le signal *rst_state* permet de commencer le codage d'une trame à partir d'un état "tout à zéro". Après avoir trouvé l'état circulaire, le signal *init_state* permet d'initialiser le codage à partir de cet état circulaire. Afin d'assurer la continuité d'écriture et de lecture des

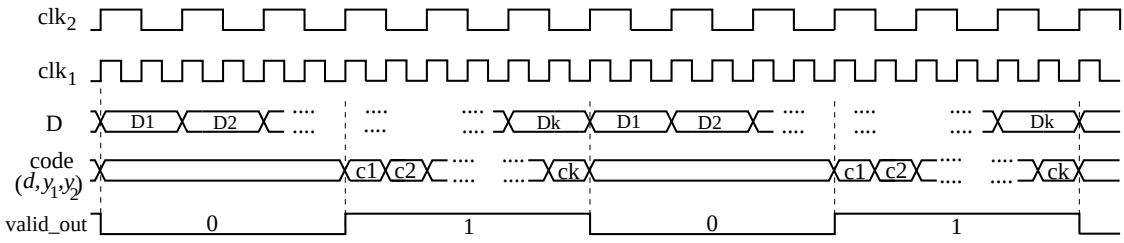


FIGURE 4.6 – Chronogramme de la génération du mot de code dans la partie émettrice.

données au sein des deux mémoires **MEM**, l'horloge clk_1 fonctionne deux fois plus rapidement que l'horloge clk_2. Cela correspond à la situation où le temps d'écriture d'une trame dans une mémoire **MEM** est équivalent à deux fois celui du codage de la trame sortie de l'autre mémoire **MEM** (*c.f* la figure 4.6). Nous obtenons alors un mot de code (d, y_1, y_2) à l'issue du codage. Par conséquent, l'émetteur ne produit pas de données codées pendant $k * clk_1$ cycles de l'horloge clk_1, ce qui correspond à l'étape de codage à partir d'état "tout à zéro". Un bloc **Sync** qui contient seulement des bascules FlipFlops permet de synchroniser les sorties de l'émetteur afin que les symboles codés soient successivement envoyés au canal selon l'ordre : (premier : systématique, deuxième : redondance 1, dernier : redondance 2). Le signal *valid_out* indique au récepteur la disponibilité des données à la sortie du codeur.

4.3.2.1 Contrôleur de la partie émission

Ce module permet d'organiser les échanges des données au sein de l'émetteur comme illustré dans la figure 4.7. Le contrôleur est de type machine d'états et comprend 5 états différents :

- Init : Etat de départ.
- Circulation state : Etat temporel de traitement afin de trouver l'état circulaire.
- Init coding : Etat d'initialisation du codage à partir de l'état circulaire.
- Coding : Etat de codage circulaire des données.
- Wait new frame : Etat temporel d'attente d'une nouvelle trame à coder.

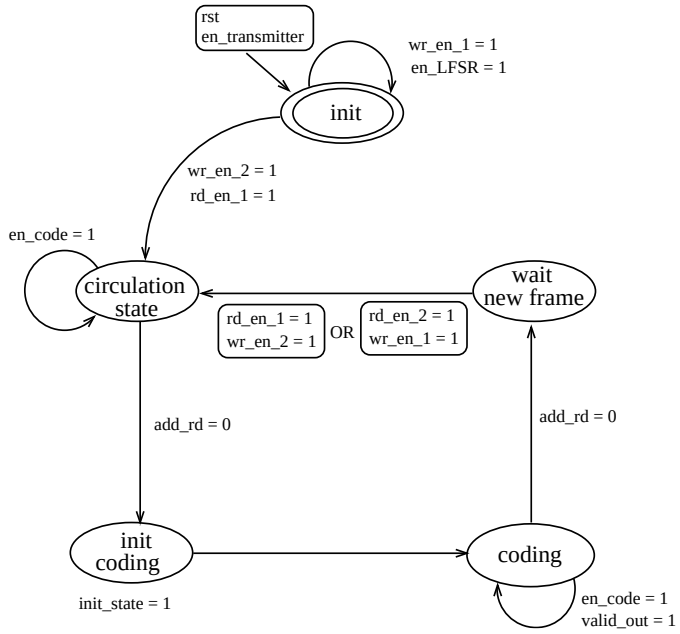


FIGURE 4.7 – Contrôleur de la partie émission.

Le signal de remise à zéro asynchrone *rst* initialise le contrôleur dans le premier état. Après un certain nombre de cycles d'horloge clk_2, le signal *en_transmitter* qui est délivré par le contrôleur général de la chaîne de transmission est activé pour demander l'exécution de l'émetteur. Lorsque le signal *wr_en_1* est actif, la génération d'une trame d'information et l'écriture des données dans la première mémoire **MEM I** se produisent. Lorsque la mémoire **MEM I** est remplie et le signal *wr_en_2* est actif, l'émetteur passe alors à l'état temporel "circulation state". Au cours de cet état, les codeurs convolutifs sont activés par le signal *en_code*. Dès lors, le processus de codage de la trame débute afin de trouver l'état circulaire. A chaque cycle d'horloge clk_2, un bit d'information et un bit entrelacé sont lus et sont ensuite codés par les deux codeurs *RCS*. Le codage est fini lorsqu'il n'y a plus de bit à lire dans la mémoire **MEM** (le signal *add_rd* est actif à 0). Dans ce cas, l'émetteur passe à un autre état temporel, qui permet d'initialiser les états des deux codeurs *RSC* aux états circulaires correspondant. Lorsque l'émetteur a terminé la transition vers l'état circulaire, il se place dans l'état "coding". Dans cet état, les deux codeurs convolutifs effectuent le codage circulaire (le signal *en_code* est à 1) et fournissent les symboles codés à l'émulateur du canal. Le signal *valid_out*, lorsqu'il est à 1, indique que les symboles codés sont disponibles à la sortie de l'émetteur. Le codage circulaire se termine lorsque les symboles de la mémoire **MEM** (le signal *add_rd* est égal à 0) sont tous lus. L'émetteur se met alors dans un état d'attente jusqu'à l'activation du signal d'entrée soit *wr_en_1* ou soit *wr_en_2* indiquant la génération d'une nouvelle trame.

4.3.2.2 Synthèse logique de la partie émettrice

Virtex-5	FlipFlop	LUT	BRAM	DSP	f_{max} (MHz)
Valeur	172	182	0	0	420

TABLE 4.1 – Complexité de synthèse de l'émetteur pour une cible FPGA.

Pour estimer la complexité matérielle d'une architecture donnée sur une cible FPGA, nous nous intéressons aux ressources nécessaires, à savoir : FlipFlop, LUT (Lookup Table), BRAM (Bloc RAM) et bloc DSP. Nous donnons alors les nombres respectifs de ressources nécessaires à partir des rapports fournis par l'outil de synthèse XST de la société Xilinx. Le tableau 4.1 récapitule les résultats de synthèse pour la partie émettrice implémentée. La fréquence maximale estimée f_{max} est également donnée. Une fréquence peut être fixée dans le fichier de contrainte défini par l'utilisateur. Notons que l'émetteur requiert très peu de ressources matérielles : moins de (1%) du taux d'occupation du FPGA Virtex-5 LX330 en termes de bascules FlipFlops, et en termes de LUTs. Aucune mémoire de type BRAM n'est requise. La fréquence de fonctionnement pour le générateur de données pseudo-aléatoires est fixée en se basant sur la fréquence maximale de fonctionnement du module récepteur qui sera donnée aux sections suivantes.

4.4 L'émulateur de canal de transmission

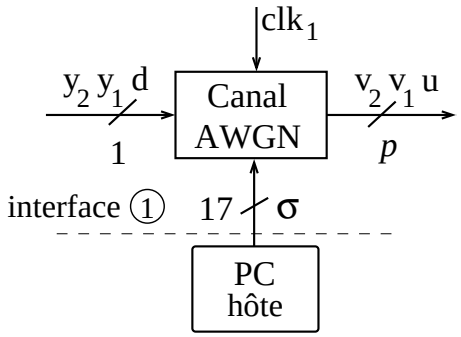


FIGURE 4.8 – Interface d'un canal AWGN - *Additive White Gaussian Noise*.

Après codage d'une trame, les symboles codés sont transmis à travers un émulateur de canal de transmission. Il est donc nécessaire d'implémenter l'émulateur du canal retenu sur le circuit FPGA. La figure 4.8 présente l'interface de l'émulateur de canal BABG - Bruit Additif Blanc Gaussien (AWGN - *Additive White Gaussian Noise*) implémenté. La déviation du bruit σ est pré-calculée selon la puissance souhaitée, c'est-à-dire le rapport signal sur bruit, et fournie par le PC hôte à travers l'interface numéro 1 (*c.f* la figure 4.1). Chaque valeur de σ est codée sur 17 bits dont 16 bits dédiés à sa partie fractionnaire. La valeur de σ correspond à des RSBs allant de 0 dB à 15 dB avec un pas de 0,25 dB. Les valeurs bruitées u, v_1, v_2 sont représentées sur 24 bits pour assurer une meilleure précision. Elles sont respectivement obtenues à partir des entrées d, y_1,

y_2, issues de l'émetteur. Au cours de la transmission, un émulateur de canal ajoute un bruit blanc gaussien aux messages codés. Dans la section suivante nous présentons le principe de génération du bruit selon la méthode de Wallace et le résultat de synthèse de l'émulateur de canal.

Génération du bruit gaussien suivant la méthode de Wallace

Considérons le bruit comme une variable aléatoire **B** caractérisée par sa variance σ^2 et une espérance nulle. Nous obtenons alors une distribution gaussienne $\mathcal{N}(0,\sigma)$ de la variable **B**. Ce modèle de canal est basé sur la méthode de Wallace [97]. Il a été étendu pour le système MIMO par Oscar David SANCHEZ GONZALEZ durant son travail de stage de fin d'étude au département Electronique de Telecom Bretagne. Ce travail a été publié dans [98]. Cette méthode est utilisée pour générer une variable aléatoire gaussienne ayant une distribution normale $\mathcal{N}(0,1)$.

L'intérêt d'utilisation de la méthode de Wallace, présentée dans [97], par rapport à d'autres méthodes et en particulier la méthode de Box-Muller présentée dans [99], est qu'elle ne requiert aucune évaluation des fonctions transcendantales non-linéaires comme $(\sqrt{x}, log(x), sin(x))$. Tandis que la méthode de Box-Muller est l'une des méthodes les plus utilisées pour générer des distributions non uniformes. Cependant, des expérimentations utilisant le générateur de bruit gaussien selon la méthode Box-Muller ont montrées des dégradations de performances de correction ayant pour origine les imperfections du générateur [100]. Par rapport à la méthode de Box-Muller, la méthode de Wallace possède donc de bonnes propriétés avec une complexité de mise en œuvre plus faible [99, 100]. A partir d'un ensemble d'échantillons d'une distribution gaussienne, la méthode de Wallace permet de générer une nouvelle séquence d'échantillons ayant également une distribution gaussienne en utilisant une formulation de type [100] :

$$x_{n+1} = \frac{1}{2}(x_{n-a} + x_{n-b} + x_{n-c} + x_{n-d}) \quad d > c > b > a \geqslant 0 \tag{4.3}$$

A chaque étape, **K** variables sont retenues à partir des échantillons initiaux et **K** nouvelles variables sont générées par une transformation linéaire. De plus, les futurs échantillons seront, quant à eux, sélectionnés à partir des nouveaux échantillons générés. Dong-U Lee et *al.* ont également proposé une implémentation matérielle de l'algorithme de Wallace [100]. Cette implémentation était basée sur les idées suggérées par Wallace [97], un ensemble de $4 * 256 = 1024$ échantillons gaussiens stockés dans une RAM à double accès, de telle sorte que l'ensemble puisse être mis à jour. Ce traitement permet de se prémunir de la corrélation possible en recourant à un générateur uniforme de T_URNG pour adresser les échantillons retenus. La variable gaussienne générée est quantifiée sur 24 bits comme dans [97].

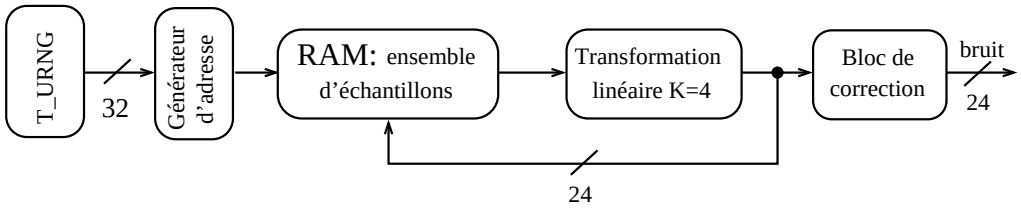


FIGURE 4.9 – Architecture du générateur de bruit gaussien selon la méthode de Wallace.

Afin de vérifier la normalité des échantillons produits, deux tests statistiques sur un grand nombre d'échantillons ont été appliqués. Les tests sont le test du χ^2 et le test de A-D comme expliqué dans [101]. Ces deux tests sont utilisés pour valider la fonction de distribution. Le test A-D (Anderson-Darling) assure que les valeurs éloignées de la valeur moyenne (les valeurs aux queues de la distribution gaussienne) sont bien distribuées. Le test χ^2 est un test de non-corrélation entre deux variables. Il est utilisé pour vérifier la distribution des valeurs autour de la valeur moyenne. Lorsque les échantillons sont validés par ces deux tests, la variable aléatoire correspondante est normalement distribuée dans tout son intervalle. Ces deux tests considèrent une statistique de test pour calculer une valeur, notée p-*value* [101]. Si la valeur p-*value* obtenue à partir d'un des tests est supérieure à 0,05, alors les données sont supposées d'être normalement distribuées. Le résultat d'implémentation des deux méthodes Wallace et Box-Muller est donné dans [100]. Ces deux méthodes passent les tests statistiques avec succès sur un grand nombre d'échantillons de bruit générés. Néanmoins, la méthode de Box-Muller requiert plus de complexité et elle est moins rapide que celle de Wallace. Par conséquent, dans le cadre de cette thèse, nous considérons le canal utilisant la méthode de Wallace.

Pour obtenir un canal AWGN - *Additive White Gaussian Noise*, il est nécessaire de multiplier la déviation du bruit σ (figure 4.8) selon la puissance souhaitée avec la variable aléatoire ayant une distribution normale $\mathcal{N}(0,1)$. Puis, le bruit est ajouté aux messages codés à transmettre. Les symboles transmis par l'émulateur de canal sont ensuite tronqués avant d'être traités par le turbo-décodeur. Dans notre cas, les données u, v_1, v_2 sont saturées et quantifiées sur 7 bits sous le format $(p = 7, q = 4)$.

Virtex-5	FlipFlop	LUT	BRAM	DSP48	f_{max} (MHz)
Valeur	576	1323	1	7	66,18

TABLE 4.2 – Complexité de synthèse de l'émulateur de canal.

Les résultats de synthèse logique pour notre émulateur de canal selon la méthode de Wallace est donné dans le tableau 4.2. Cet émulateur n'utilise que 576 des FlipFlops, 1323 LUTs sur un Virtex-5 LX330. Un bloc de RAM et 7 blocs de DSP48 sont nécessaires. Enfin, l'émulateur du canal peut fonctionner à une fréquence allant jusqu'à $f_{max} = 66,18$ Mhz.

4.5 Décodeur stochastique pour turbocodes et codes convolutifs

Avant de détailler l'architecture du turbo-décodeur stochastique, quelques paramètres algorithmiques utilisés pour le prototypage sont à considérer :

- Canal BABG - Bruit Additif Blanc Gaussien (AWGN - *Additive White Gaussian Noise*).
- Modulation de type BPSK.
- Code convolutif systématique récursif circulaire (CRSC) à 8 états et rendement $1/2$ introduit au chapitre 1.
- Turbocode convolutif de rendement $1/3$.

- Entrelaceur $S-$Random introduit dans [37].
- Format ($p = 7, q = 4$) de quantification du symbole reçu à l'entrée du turbo-décodeur.
- Profondeur du générateur des séquences aléatoires $W = 7$ bits.
- Coefficients de correction de performance des turbocodes $\Omega = \sigma_0^2 = 1,5$ et $\psi(\sigma) = 1,0$.
- Ordre des modules exponentiels et logarithmiques ($u_e = 1, u_l = 1$).
- Comme justifié au chapitre précédent, le nombre de LUTs estimé pour implémenter une section de treillis d'un décodeur convolutif stochastique exponentiel 32-flux est trop grand (8672 LUTs). Sachant que le nombre maximum de LUTs disponible sur la carte Virtex-5 LX330 est d'environ 200.000 LUTs, il est impossible d'implémenter la version exponentielle 32-flux sur un circuit FPGA Virtex-5 LX330. Par conséquent, afin de tester le prototype, nous avons choisi une architecture disposant seulement de $\rho = 8$ flux de décodage en parallèle. De plus, nous avons sélectionné une taille de $k = 50$ bits pour la séquence d'information utile. Ces choix ne dépassent pas 70% du nombre de ressources maximum de Virtex-5 LX330 ce qui permet de les mettre en œuvre avec succès sur la carte.

4.5.1 Architecture globale du turbo-décodeur stochastique

Les symboles, après avoir été bruités, arrivent au dernier élément de la chaîne de transmission : le turbo-décodeur stochastique exponentiel multi-flux. L'architecture du turbo-décodeur stochastique est détaillée dans la figure 4.10. Ce turbo-décodeur est décomposé en deux blocs principaux : les deux décodeurs stochastiques $SISO_1$ et $SISO_2$ qui assurent l'échange des flux stochastiques extrinsèques au cours d'un processus itératif par l'intermédiaire d'un entrelaceur Π et d'un désentrelaceur Π^{-1}. L'architecture globale du décodeur convolutif stochastique est similaire à celle du turbo-décodeur stochastique présentée dans la figure 4.10, dans laquelle le décodeur $SISO_2$ ainsi que les entrelaceurs/désentrelaceurs sont éteints. L'architecture de chaque décodeur SISO a été présentée au chapitre précédent. Comme ces deux décodeurs ont besoin de valeurs aléatoires pour convertir des probabilités *a priori* en flux stochastiques, un module générant des séquences aléatoires **RNG** est indispensable. Le décodeur est doté d'un contrôleur qui permet de gérer l'ordonnancement des différentes tâches du décodeur.

Un signal REQ indique au récepteur, lorsqu'il est actif à 1, que l'émetteur est prêt à transmettre des nouvelles données. Une fois que ces nouvelles données sont transmises sur le canal et que le signal ACK est actif à 1, le contrôleur permet de démarrer la réception séquentielle des données à décoder. Pour ce faire, le contrôleur active le pré-traitement de la conversion des symboles bruités quantifiés en probabilités *a priori* représentées sur $W = 7$ bits par le module **ROM**. Pour optimiser l'efficacité du récepteur, une architecture de décodage composée de deux mémoires, notées **MEM I** et **MEM II**, est proposée. Ces deux mémoires à entrée en série, et sortie en parallèle possèdent un principe de fonctionnement de ping-pong. Le décodeur charge des symboles à partir d'une mémoire tandis que l'autre mémoire est successivement remplie par de nouveaux symboles. La taille de chaque mémoire doit être suffisante pour sauvegarder une trame. Lorsque les probabilités sont écrites dans une mémoire **MEM**, le décodeur lit les probabilités reçues dans l'autre mémoire **MEM** pour décoder. Lorsque le décodeur finit le décodage d'une trame, la mémoire correspondante est alors libérée pour recevoir une nouvelle trame. L'intérêt est de permettre un décodage

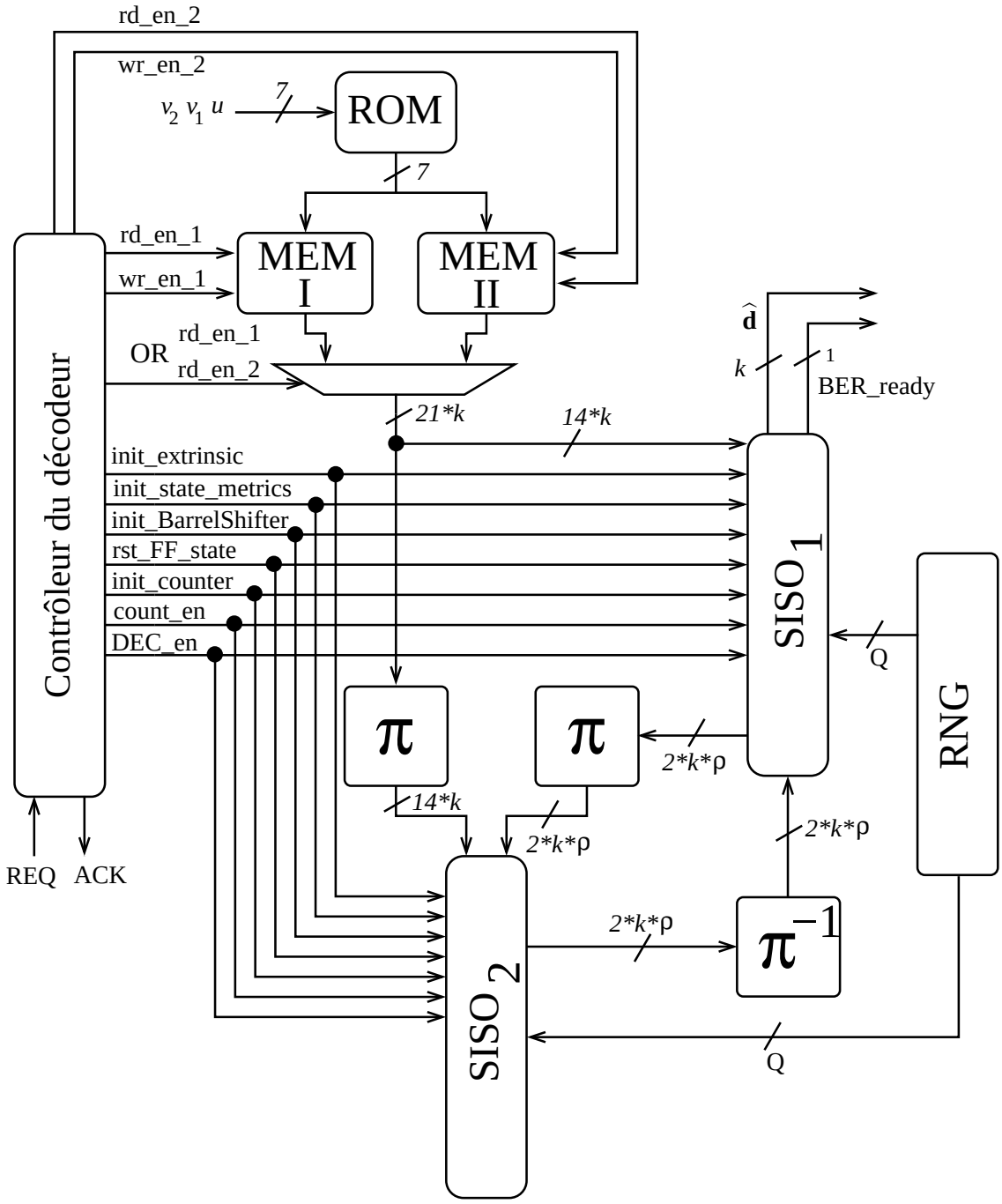


FIGURE 4.10 – Architecture du turbo-décodeur stochastique.

sans des continuités. L'écriture et la lecture des données dans une mémoire sont respectivement gérées par deux signaux *rd_en* et *wr_en* fournis par le contrôleur. Les indices 1 et 2 correspondent respectivement aux deux mémoires **I** et **II**. Les données sont écrites en série dans la mémoire, et elles sont lues en parallèle. Une fois une mémoire remplie, un signal de lecture (soit *rd_en*_1 soit *rd_en*_2) permet au décodeur de lire la trame reçue dans la mémoire correspondante. Cette phase enclenche le processus de décodage itératif de la trame en parallèle au sein de deux décodeurs sto-

chastiques SISO. Comme le décodeur stochastique ne possède aucune information de décodage au départ, deux signaux *init_extrinsic* et *int_state_metrics* permettent d'initialiser des métriques récurrentes aller/retour et des informations extrinsèques à des valeurs équiprobables. Le signal *init_BarrelShifter* sert à initialiser le fonctionnement des *barrel-shifters*. L'écriture et la lecture dans les entrelaceurs et dans le désentrelaceur se font en parallèle, simultanément au décodage. Un compteur *up/down* saturé est initialisé par le signal $int\_counter = 1$ et fonctionne lorsque le signal $cout\_en = 1$. A chaque coup d'horloge, le compteur indique le nombre de cycles courant permettant de continuer ou d'arrêter le processus de décodage itératif. Le contrôleur génère également un signal *DEC_en* pour indiquer au décodeur d'achever le comptage des bits représentant les probabilités *a posteriori*. Le signal de sortie *BER_ready* indique que la trame décodée est maintenant disponible à sa sortie, ce qui permet de démarrer le processus de détection des erreurs dans la trame et de calculer le TEB. L'architecture de deux décodeurs stochastiques exponentiels multi-flux SISO a été introduite dans le chapitre 3. Dans les deux sous-sections suivantes, nous allons détailler deux sous-modules du turbo-décodeur stochastique : le contrôleur du turbo-décodeur et le générateur des valeurs aléatoires **RNG**.

4.5.2 Contrôleur de la partie réception

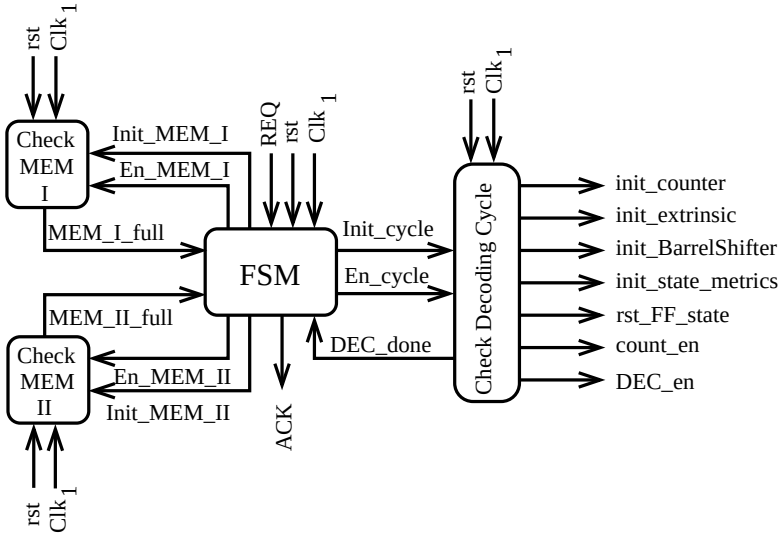


FIGURE 4.11 – Architecture du contrôleur du turbo-décodeur stochastique.

La figure 4.11 présente l'architecture du contrôleur de turbo-décodeur stochastique et la figure 4.12 décrit sa machine d'état. Le contrôleur comprend une machine d'état **FSM**, deux contrôleurs de mémoire - **Check MEM I** et **Check MEM I** - et un contrôleur du nombre de cycles de décodage. La machine d'états est composée de 7 états qui sont indexés par les numéros de 1 à 7. Au début, le décodeur est dans l'état d'attente (état 1) et prêt à recevoir des symboles bruités ($ACK = 1$). Lorsque l'émetteur et le canal ont terminé leur convergence, le signal REQ est actif à 1, le décodeur passe donc à l'état de réception des données dans la mémoire **MEM I** (état 2). Le signal MEM_I_full indique, lorsqu'il est à 1, que la mémoire **MEM I** est remplie par des

probabilités *a priori* et le décodeur passe à l'état numéro 3. A ce stade, le décodeur effectue le décodage sur les données provenant de la mémoire **MEM I** et sauvegarde des probabilités *a priori* dans la mémoire **MEM II** en parallèle. Le signal *DEC_done* indique, quand il est actif à 1, que le décodeur a terminé le décodage d'une trame. Considérons que N_d est le nombre de cycles nécessaires pour recevoir une trame de donnée (N_d étant la taille de la trame de données). Dans ce cas, le décodeur vérifie deux situations :

– Si $N_d > DC_{max}$: le décodeur prend la décision et il est alors dans l'état numéro 5. Au cours de cet état, il continue à sauvegarder des données dans la mémoire **MEM II**.

– Si $N_d < DC_{max}$: le décodeur termine la réception des données de la mémoire **MEM II**, et il continue le décodage jusqu'au nombre maximum de cycles DC_{max} (l'état numéro 4).

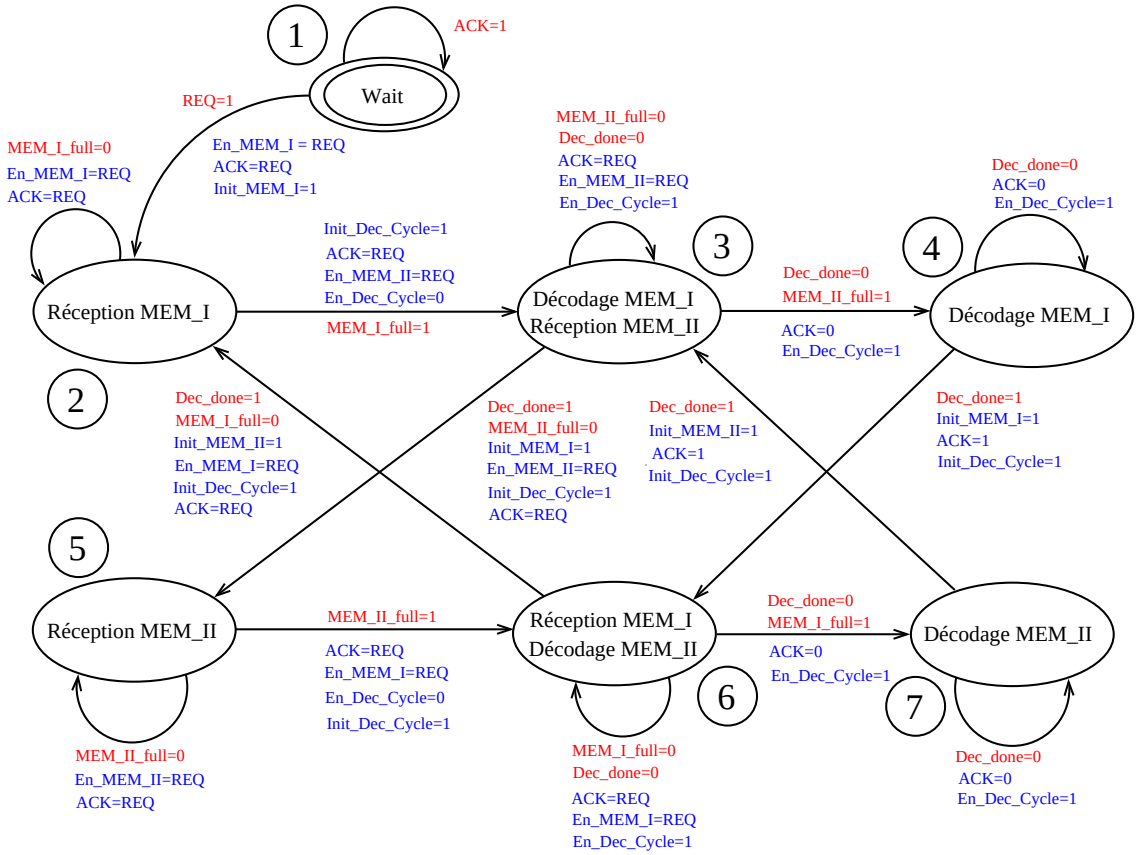


FIGURE 4.12 – Ordonnancement des différentes tâches du turbo-décodeur stochastique.

Lorsque le décodeur se trouve à l'état numéro 4 et que le nombre maximum de cycles DC_{max} est obtenu, le décodeur se place dans l'état numéro 6. Dans cet état, il traite les probabilités de la mémoire **MEM II**, et en même temps, il reçoit la nouvelle trame dans la mémoire **MEM I**. Les signaux sortant du contrôleur de cycles de décodage (*c.f* la figure 4.11) sont les entrées de chaque décodeur SISO. Ils sont activés en fonction du nombre de cycles de décodage prédéfinis.

Le processus réception - décodage est alternativement appliqué aux deux mémoires et il se termine lorsque le nombre de bits erronés ou le nombre de trames erronées est atteint. Il est

également à noter que le turbo-décodeur est cadencé par la même horloge que le canal et que celle du turbo-codage circulaire clk_1.

4.5.3 Générateurs de séquences aléatoires - RNG

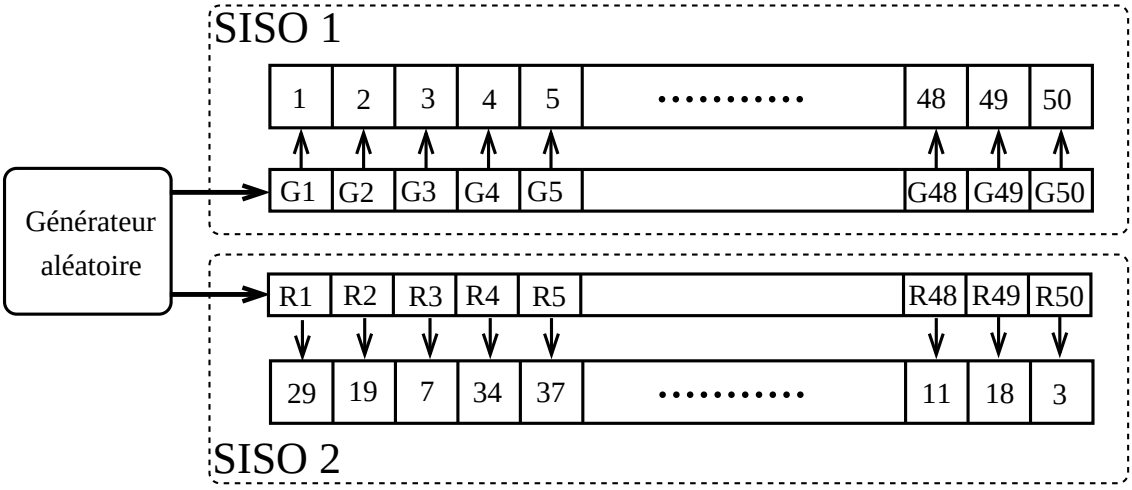


FIGURE 4.13 – Deux séquences de bits aléatoires nécessaires pour deux décodeurs élémentaires.

Le turbo-décodeur comprend deux codeurs convolutifs stochastiques. La séquence de probabilités *a priori* d'entrée pour chaque décodeur élémentaire est la même mais aux différents ordres : naturel et entrelacé. La figure 4.13 décrit l'organisation des probabilités *a priori* au sein de chaque décodeur SISO. Dans le décodeur $SISO_1$, les probabilités sont traitées dans l'ordre naturel : de 1 à 50. Dans le décodeur $SISO_2$, en fonction de l'entrelaceur *S*-Random, les probabilités de la position 29 sont associées à la première section, celles de la position 19 sont associées à la deuxième section,... Chaque décodeur requiert une séquence de valeurs aléatoires pour convertir ces probabilités en flux stochastiques : la section *i* du premier décodeur nécessite une séquence de bits aléatoires *Gi* et celle du second décodeur est associée à une séquence de bits aléatoires *Ri* (*c.f* la figure 4.13). Dans ce cas, le nombre de bits aléatoires d'un turbo-décodeur est deux fois plus grand que celui d'un décodeur élémentaire.

Dans le chapitre précédent, nous avons expliqué que le modèle d'un décodeur stochastique exponentiel multi-flux SISO nécessitait des bits aléatoires pour une section, afin de convertir des probabilités *a priori* correspondant à deux symboles systématiques et de redondances (*c.f* le tableau 3.6 du chapitre 3). La figure 4.14 montre une répartition d'utilisation des bits aléatoires au coeur de chaque décodeur SISO. Chaque colonne correspond à une section de treillis et indique le nombre de flux implémentés pour une section ($\rho = 8$). Chaque ligne correspond au nombre de sections ($k = 50$). Chaque colonne *i* requiert 8 vecteurs de $W = 7$ bits aléatoires rs_{ij} avec $j = (1->8)$ pour convertir la même probabilité sys_i en 8 flux stochastiques et 8 vecteurs de $W = 7$ bits rd_{il} avec $l = (1->8)$ pour convertir la même probabilité red_i en 8 flux stochastiques. Avec $(k = 50)$ sections différentes d'un turbo-décodeur, chacune comprend $\rho = 8$ flux stochastiques, alors, le nombre de bits aléatoires nécessaire est égal à (2*7*8*50=22.400) bits. Si nous supposons que chaque bit aléatoire est construit par un registre à décalage de 10 bascules. Le nombre de bascules

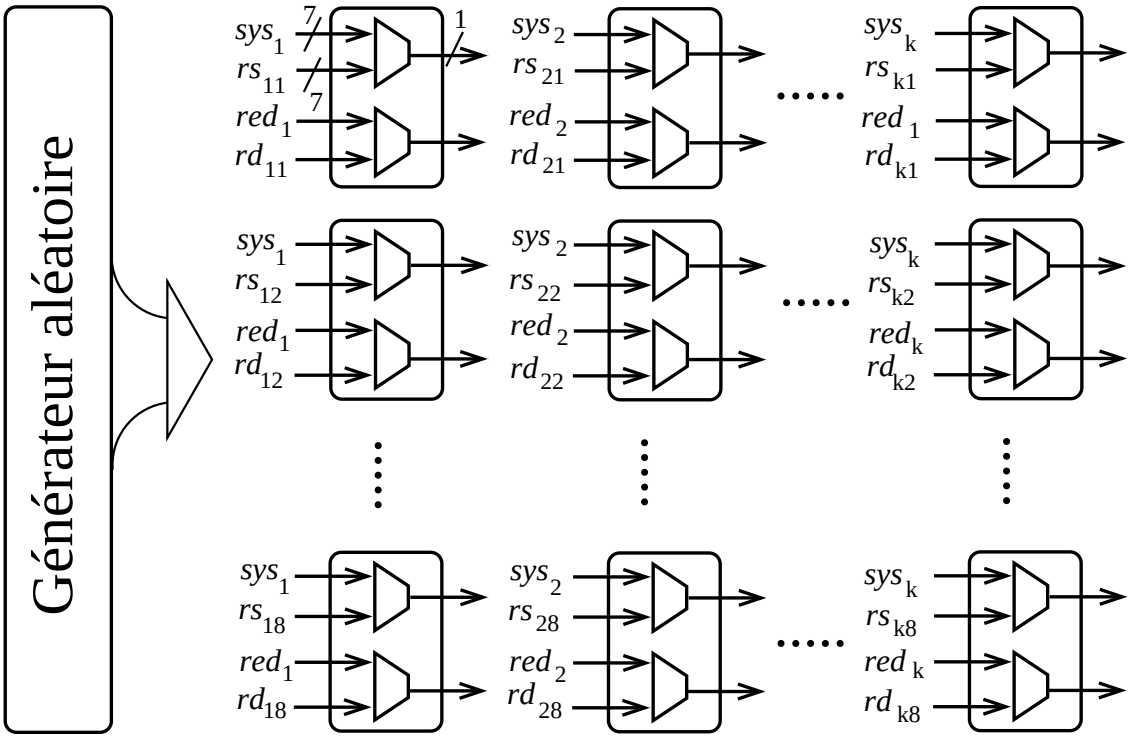


FIGURE 4.14 – Répartition des séquences aléatoires au sein d'un décodeur SISO.

requis est alors égal à 224.000 bascules. Ce chiffre dépasse la ressource maximale disponible sur un circuit Virtex-5 LX330. Ce problème est critique lorsque la longueur de la séquence d'information devient grande. Dès lors, des techniques sont nécessaires pour réduire le nombre de bits aléatoires requis.

4.5.3.1 Partage de bits aléatoires entre les deux décodeurs SISO

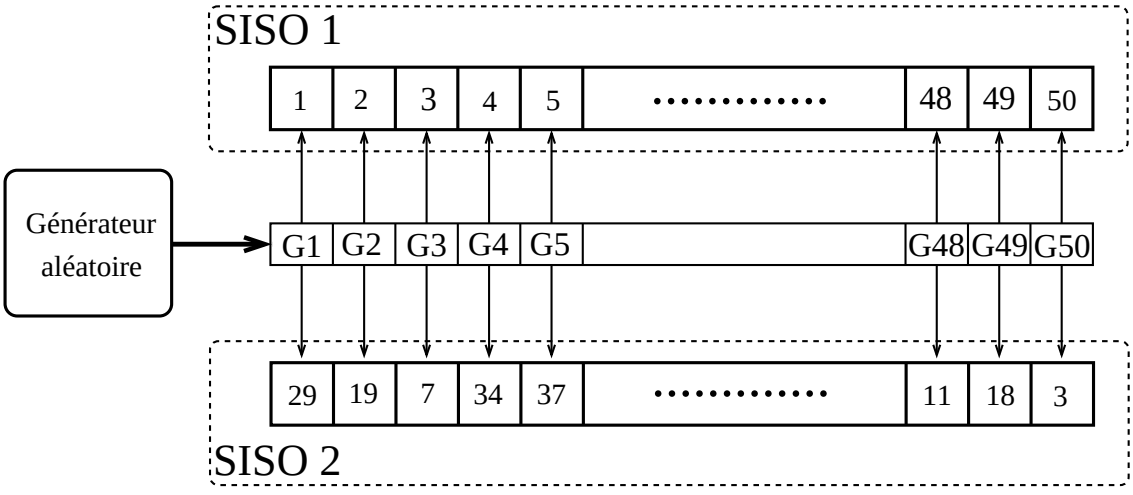


FIGURE 4.15 – Une même séquence de bits aléatoires associée aux deux décodeurs élémentaires.

Comme indiqué dans la figure 4.13, la première section du $SISO_1$ contient des probabilités, notées probabilités 1, et est alimentée par la séquence de bits aléatoires $G1$. En parallèle, après entrelacement, la première section du $SISO_2$ reçoit des probabilités, notées probabilités 29, et est alimentée par la séquence de bits aléatoires $R1$. Dans une séquence d'information, ces deux probabilités sont à des positions différentes, et ne présentent pas d'inter-corrélation. Par conséquent, au lieu de générer deux séquences différentes de bits aléatoires $G1$ et $R1$, nous pouvons réduire le nombre de séquences en utilisant une même séquence aléatoire comme montré dans la figure 4.15. Cela permet alors de réduire d'un facteur deux le nombre de bits aléatoires. En conséquence, le nombre de bits aléatoires total est maintenant égal à 11.200 bits. Si on suppose que chaque bit est construit à partir d'un registre à décalage de 10 bascules, le nombre de bascules implémentées est donc 112.000. Ce nombre est encore trop élevé. Cela signifie qu'une solution complémentaire doit être envisagée afin de diminuer le nombre de bits aléatoires.

4.5.3.2 Partage de bits aléatoires entre les différentes sections

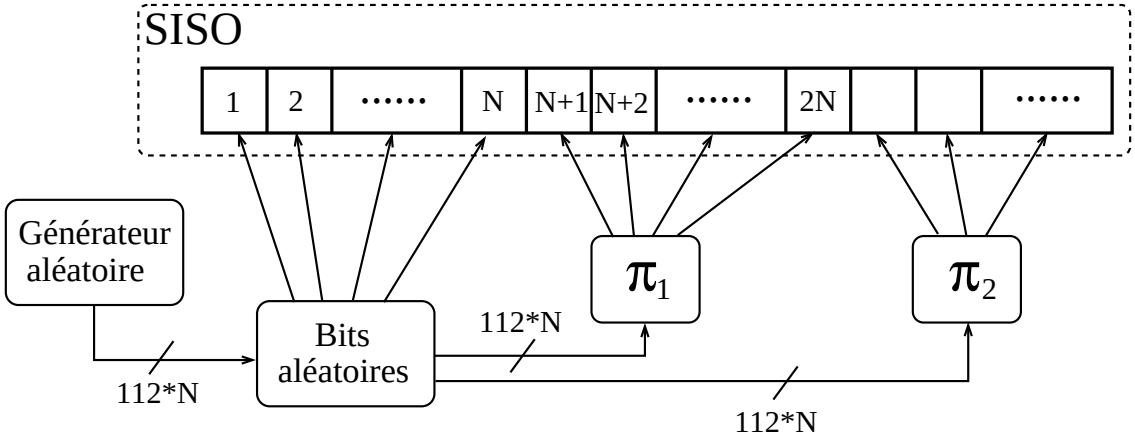


FIGURE 4.16 – Partage des bits aléatoires au sein d'un décodeur SISO.

Dans le but de réduire le nombre de bits aléatoires, des architectures efficaces comme celle décrite dans [16] doivent être considérées pour générer les bits aléatoires. Dans notre architecture, nous proposons de générer un jeu de bits aléatoires pour un certain nombre de sections prédéfinies N $(N < 50)$. Ensuite, ce jeu de bits sera réutilisé par d'autres sections. Chaque section nécessite $2*7*8 = 112$ bits aléatoires, le nombre de bits générés pour N sections est alors de $112*N$. Afin de réduire la complexité de l'architecture du générateur aléatoire, nous choisissons de répéter ces bits aléatoires selon une règle déterministe et simple. Ce nombre de bits est d'abord assigné à N sections successives selon l'ordre $1 \rightarrow N$. Puis, il est respectivement assigné à N sections suivantes selon l'ordre $(N+1) \rightarrow 2*N$,... Cette distribution doit considérer les conditions suivantes :

- Le nombre de sections N choisi doit être suffisamment grand pour assurer une décorrélation entre les flux stochastiques sortant des différentes sections, par exemple, les sections (1, N+1, 2*N+1, ...) ou les sections (2, N+2, 2*N+2,...).
- Dans une section, les bits aléatoires sont requis pour générer des flux stochastiques représentant une même probabilité correspondant au symbole systématique et de redondance.

Alors, les bits aléatoires dans ce jeu ($112 * N$ bits aléatoires) doivent être décorrélés entre eux.

– Ces bits aléatoires associés à d'autres sections peuvent alors être tirés afin de décorréler les différents flux stochastiques sortants.

Par conséquent, nous proposons une architecture distribuée de génération des bits aléatoires qui permet de générer les bits aléatoires pour $N = 8$ sections différentes. Basé sur notre connaissance des simulations fonctionnelles, $N = 8$ est suffisamment grand pour ne pas avoir de corrélation entre les sections. Avant de les utiliser pour d'autres sections, des entrelaceurs sont indispensables Π_i ($i = 1, 2, ...$) pour mélanger ces bits aléatoires. Cette solution est illustrée dans la figure 4.16. Les entrelaceurs Π_i considérés dans notre prototypage sont de type entrelaceur aléatoire [37]. Ainsi, le nombre de bits aléatoires total généré par notre architecture est égal à 896 bits.

Les bits aléatoires peuvent être générés à partir de plusieurs solutions. Ils peuvent être construits, par exemple, à partir des registres à décalage qui sont présentés dans la section 4.3.1. Ils peuvent également être construits par une autre approche : un véritable générateur de valeurs aléatoires (en anglais *True Random Number Generator - TRNG*). Ce type de générateur fonctionne en se basant sur une source de bruit physique de composant (bruit thermique, décomposition radioactive,...). Il est largement considéré dans la littérature car il requiert moins de ressources matérielles et de consommation d'énergie que les registres à décalage conventionnels [102]. Une autre *TRNG* a été développé au Département Electronique de Telecom Bretagne et nous envisageons de l'utiliser dans une prochaine architecture.

Dans notre étude, nous considérons les bits aléatoires construits à partir des registres à décalage. Le nombre de bascules de chaque registre est choisi en se basant sur la longueur maximale de la séquence pseudo-aléatoire qui dépend du polynôme implémenté. En outre, les résultats d'implémentation de notre architecture sur un circuit FPGA nous montre que 500 cycles de décodage sont suffisants pour obtenir une bonne performance de décodage. Par conséquent, nous avons choisi des registres à décalage comprenant 12 bascules représentés par le polynôme $P(X) = X^0 + X^3 + X^5 + X^{11}$. Ce type de générateur fournit une séquence pseudo-aléatoire de longueur 4095.

4.5.4 Synthèse logique sur circuit FPGA

Virtex-5 LX330	RNG	SISO	Entrelaceur	Turbo-décodeur
Slice LUTs	4218 (2%)	40531 (20%)	-	101708 (49%)
Slice FlipFlops	10359 (5%)	18787 (9%)	1600 (1%)	52641 (25%)
BRAM	0	0	0	0
DSP48	0	0	0	0
f_{max}(MHz)	-	197	-	133

TABLE 4.3 – Résultats de synthèse logique pour le turbo-décodeur stochastique.

Le tableau 4.3 décrit les résultats de synthèse logique pour les différents blocs constituant le turbo-décodeur stochastique : RNG, entrelaceurs/désentrelaceur, décodeur SISO et turbo-décodeur. Une estimation de la fréquence maximale (f_{max}) à l'issue de la synthèse a été également faite pour ces blocs ce qui permet d'estimer la fréquence de fonctionnement de l'architecture du turbo-décodeur stochastique.

L'entrelaceur, ainsi que le désentrelaceur, utilise seulement des bascules et ce nombre de bascules est égal à celui de probabilités extrinsèques sortant de chaque SISO : chaque section possède 8 flux en parallèle et chaque flux produit 2 flux stochastiques représentant deux probabilités extrinsèques possibles, comme nous l'avons déjà justifié au chapitre 3.

Au niveau de la répartition de la complexité au sein du turbo-décodeur stochastique, un décodeur SISO représente environ 40% de la complexité en termes de LUTs et 36% en termes de FlipFlops. Au total, le turbo-décodeur stochastique exponentiel 8-flux utilise 49% des LUTs et 25% des FlipFlops disponibles sur un Virtex-5 LX330. Aucun bloc de BRAM et de DSP48 n'est requis par le turbo-décodeur stochastique. De plus, par une comparaison de la complexité entre le décodeur stochastique SISO et le turbo-décodeur stochastique montrée dans le tableau 4.3 du chapitre 4, nous constatons que les deux modules *Ext* du turbo-décodeur représentent une grande complexité de ressources disponibles du circuit Virtex-5 LX330 : 9% en termes de LUTs et 6% en termes de FlipFlops.

Nous pouvons noter que le turbo-décodeur requiert plus des LUTs que des bascules, ce qui a été bien justifié par le tableau de complexité 3.7 du chapitre 3. D'ailleurs, le chapitre 3 a également démontré que l'architecture multi-flux du turbo-décodeur stochastique permettait une augmentation du débit de ρ fois, elle augmentait aussi la complexité d'un facteur ρ. Nous obtenons un meilleur débit, malheureusement, nous perdons la complexité. Cela explique que notre prototype de turbo-décodeur dans ce cas utilise un grand nombre de surfaces, 49% de ressources du circuit FPGA Virtex-5 en termes de LUTs.

En outre, le décodeur stochastique SISO peut fonctionner jusqu'à une fréquence de $f_{max} = 197$ MHz, et le turbo-décodeur stochastique peut atteindre une fréquence maximale de $f_{max} = 133$ MHz. Ces deux types de décodeur se disposent de différentes fréquences maximales car le turbo-décodeur exige plus de surface, alors, le chemin critique issu du routage devient problématique. Par une synthèse sur le circuit FPGA Virtex-5 LX330, nous constatons que le routage contribue 88,8% du temps du chemin critique et le calcul logique contribue seulement 11,2% du temps du chemin critique.

4.6 Architecture globale de la chaîne de transmission

4.6.1 Architecture globale

L'architecture globale de la chaîne de transmission est présentée dans la figure 4.17. Le module BER/FER détermine les taux d'erreurs binaires et paquets après un nombre de cycles de décodage maximum (DC_{max}) en comparant la décision issue du turbo-décodeur $\widehat{\mathbf{d}}$ avec les données générées par l'émetteur $\mathbf{d}$. Dans notre cas, nous privilégions la mesure des performances en termes de taux d'erreur binaire (TEB). La valeur du TEB après un nombre de cycles DC_{max}

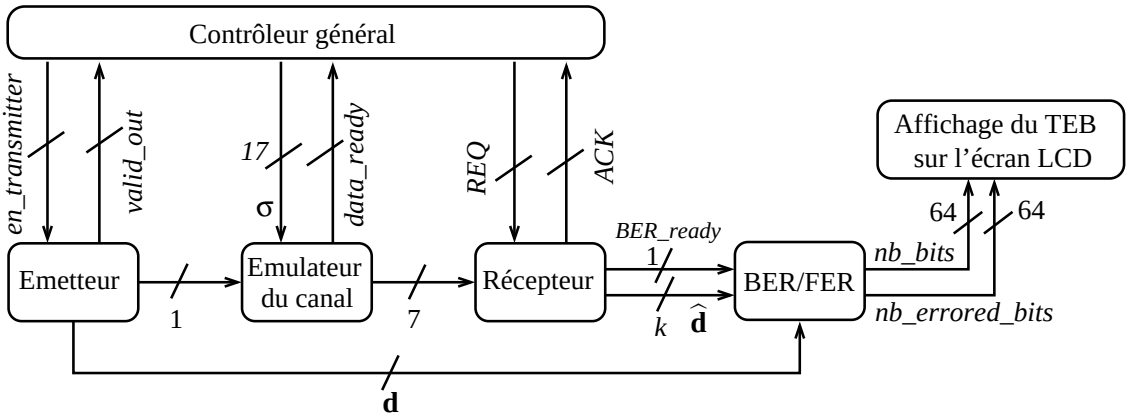


FIGURE 4.17 – Schéma bloc de la chaîne de transmission numérique globale.

est calculée à partir du nombre de bits générés et du nombre de bits erronés. Ces deux nombres peuvent être lus par le PC hôte via le port PCI.

Le contrôleur général du système permet la gestion des tâches entre les trois blocs principaux de la chaîne de transmission : l'émetteur, l'émulateur du canal et le récepteur à savoir soit le décodeur stochastique SISO soit le turbo-décodeur stochastique. Au début, le turbo-décodeur est en état d'attente (*ACK=1*) et est prêt à recevoir de nouvelles données. Le signal *data_ready* est mis à 1 lorsque l'émulateur du canal est prêt à démarrer la génération des variables gaussiennes et génère une valeur de bruit à chaque cycle d'horloge. Le contrôleur demande à l'émetteur de générer une nouvelle trame en mettant le signal *en_transmitter* à 1. Dès que l'émetteur démarre le processus de codage circulaire (*valid_out*=1), le signal *REQ* est activé à 1 et indique au turbo-décodeur de recevoir les symboles transmis par l'émulateur du canal. En même temps que le turbo-décodeur effectue le décodage d'une trame, le signal *ACK* passe à 0, le contrôleur se met alors en attente de la fin du décodage de cette trame. Une fois que le turbo-décodeur a terminé le décodage d'une trame, le signal *ACK* est activé à 1 pour signifier à l'émetteur de transmettre une nouvelle trame.

4.6.2 Synthèse logique sur circuit FPGA

	Emetteur	Canal	SISO	Turbo-décodeur	Chaîne complète
LUTs	182 (0%)	1323 (1%)	40531 (20%)	101708 (49%)	120181 (58%)
FlipFlop	172 (0%)	576 (0%)	18787 (9%)	52641 (25%)	53435 (26%)
BRAM	0	1	0	0	1
DSP48	0	7	0	0	7
f_{max}(MHz)	420	66,18	197	133	66,18

TABLE 4.4 – Résultats de synthèse logique pour la chaîne de transmission numérique.

Le tableau 4.4 récapitule les résultats de synthèse logique pour les différents modules de la

chaîne de transmission numérique. On peut constater que le turbo-décodeur est bien plus coûteux en surface que les autres modules en termes de ressources matérielles. Au niveau de la répartition de la complexité au sein de la chaîne de transmission, le récepteur occupe quasiment 100% de la complexité en termes de LUTs et ainsi qu'en termes de FlipFlops. Il représente également 58% de la complexité de ressources disponibles du circuit Virtex-5 LX330 en termes de LUTs et 26% en termes de FlipFlops.

4.6.3 Performance du prototypage

Le récepteur de la chaîne de transmission numérique illustré dans la figure 4.17 peut être soit un décodeur stochastique SISO soit un turbo-décodeur stochastique. Dans cette sous-section, nous allons présenter les performances de décodage de notre prototype lorsqu'il représente un décodeur stochastique SISO ou un turbo-décodeur stochastique. Les courbes de performances stochastiques sont obtenues en langage C++ après 100 trames erronées, et celles sur les prototypes sont obtenues après 10.000 bits erronés.

4.6.3.1 Performance du décodage des codes convolutifs

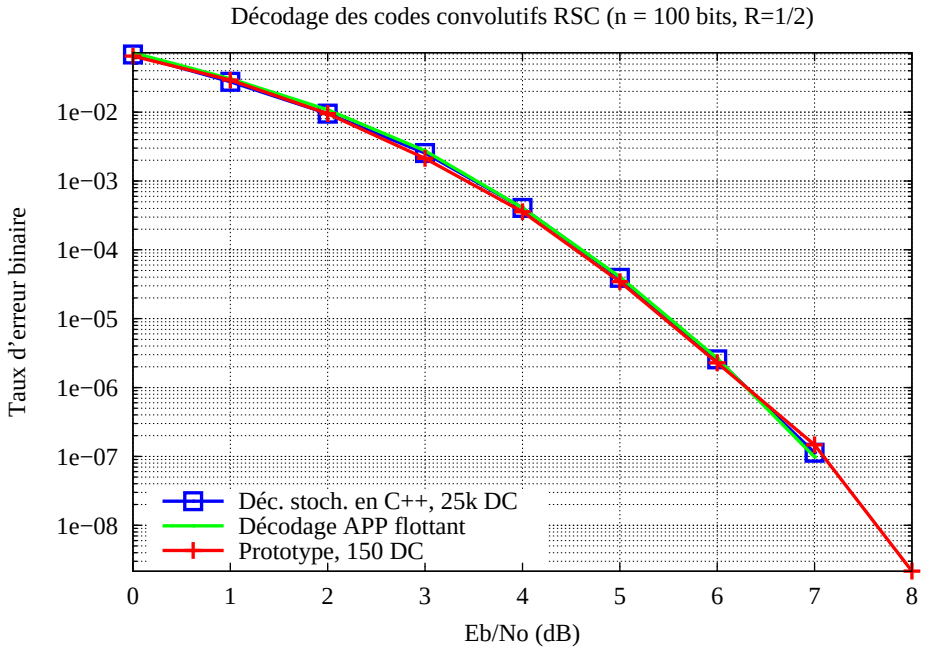


FIGURE 4.18 – Performance du prototype de décodeur stochastique des codes convolutifs (k=50, $R = 1/2$) sur un circuit Virtex-5.

La figure 4.18 présente les performances de décodage des codes convolutifs. Les codes choisis dans cette démonstration sont les codes de taille 50 bits et de rendement $R = 1/2$. Afin de supprimer l'impact de corrélation issu du décodage stochastique introduit au chapitre 1, les coefficients de correction sont respectivement : $\Omega = \sigma_0^2 = 1,0$ et $\psi(\sigma) = 1$ comme détaillé au chapitre 2. La quantification des symboles d'entrée au récepteur est sous le format (p=7,q=4) et la nombre

de bits représentant les probabilités *a priori* est de 7 bits. Le décodeur SISO comprend 8-flux de décodage en parallèle.

Le nombre de cycles DC_{max} égal à 150 DCs est nécessaire afin de montrer que notre prototype peut achever une bonne performance de décodage des codes convolutifs. La courbe rouge correspond à la performance du décodeur stochastique SISO de notre prototype. Cette performance est comparée avec celle d'un décodeur stochastique conventionnel SISO avec 25k DCs obtenu par les simulations comportementales en langage C++ représentée par la courbe bleue et celle d'un décodeur APP flottant représentée par la courbe verte. Nous notons que ces performances sont quasiment similaires.

4.6.3.2 Performance du décodage des turbocodes

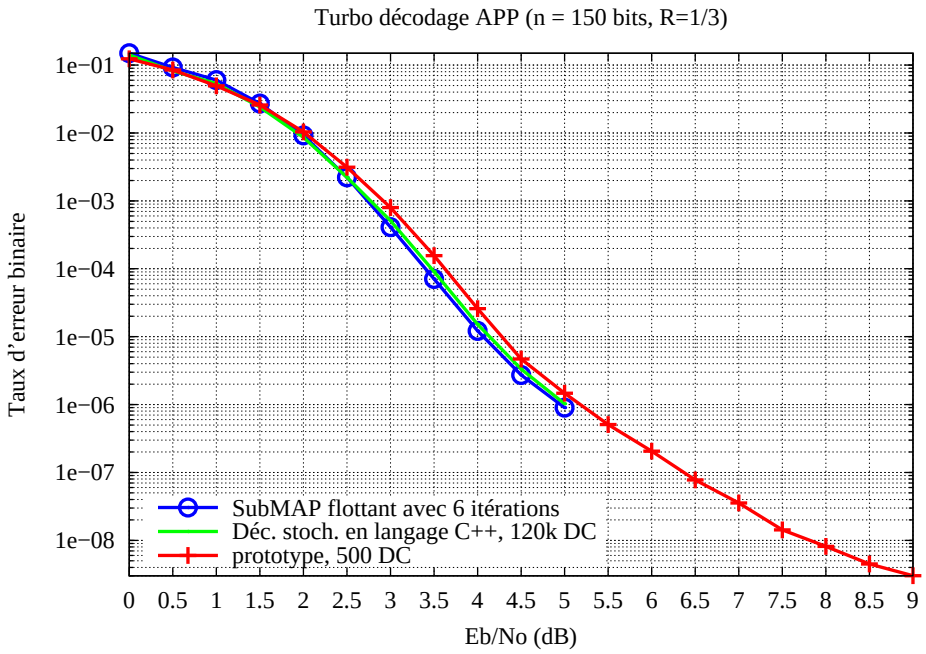


FIGURE 4.19 – Performance du prototype de turbo-décodeur stochastique (k=50, $R = 1/3$) sur un circuit Virtex-5.

La figure 4.19 présente les performances de décodage des turbocodes. La courbe verte représente la performance de décodage stochastique conventionnel avec 120k DCs obtenue par notre architecture en langage C++. La courbe bleue, quant à elle, correspond aux performances obtenues par les simulations fonctionnelles en virgule flottante avec 6 itérations (langage C++). La courbe rouge représente la performance obtenue à l'aide d'un prototype implémenté sur le circuit FPGA. Pour la performance stochastique (la courbe rouge), les données sont quantifiées sous le format (7,4). Le nombre de bits aléatoires retenu pour la conversion en flux stochastique est égal à $W = 7$ bits. L'effet de corrélation durant le décodage stochastique introduit au chapitre 1 est supprimé en choisissant le facteur de correction de la performance : $\Omega = \sigma_0^2 = 1,5$ comme donné au chapitre 2. Le coefficient $\psi(\sigma)$ est choisi en basant sur la bonne performance de décodage.

Pour notre prototype avec la taille de données de 50 bits, ce coefficient $\psi(\sigma) = 1,0$ est suffisant pour atteindre une bonne performance.

La figure 4.19 nous montre que le turbo-décodeur stochastique exponentiel 8-flux nécessite un nombre de cycles $DC_{max} = 500$ pour qu'il puisse effectuer une bonne performance par rapport à la performance fournie par le turbo-décodeur stochastique exponentiel 8-flux de référence sur le langage C++. Par ailleurs, par comparaison avec la performance de référence, à savoir la courbe en virgule flottante, nous obtenons environ 0,17dB de dégradation pour un TEB de l'ordre 10^{-6}.

4.6.4 Débit

Dans cette sous-section, nous allons estimer le débit de décodage maximal de notre prototype pour les deux types de décodeur stochastique différents : le décodeur SISO et le turbo-décodeur.

Avant de donner le débit de décodage de deux décodeurs, nous revoyons la fréquence de fonctionnement de la chaîne globale. Le tableau 4.4 indique également que, parmi ces modules, la fréquence de fonctionnement maximale de l'émulateur du canal est la plus petite, seulement 66,18 MHz. Dès lors, pour que notre chaîne de transmission numérique fonctionne correctement, le récepteur stochastique doit se limiter à une fréquence de fonctionnement maximale de $f_{max} = 66,18$ MHz, bien qu'il lui soit possible de fonctionner à une fréquence maximale, d'après les rapports de l'outil de synthèse ISE de Xilinx, jusqu'à 197 MHz pour un décodeur SISO et jusqu'à 133 MHz pour un turbo-décodeur.

Avec la fréquence maximale $f_{max} = 197$ MHz, le nombre de cycles $DC_{max} = 150$, le prototype pour des codes convolutifs atteint un débit de décodage maximal :

$$D_s = 197.\frac{50}{150} = 65,67 \quad \text{[Mbits/s]} \tag{4.4}$$

Dans le cas du prototype pour des turbocodes, la fréquence maximale $f_{max} = 133$ MHz, le nombre de cycles $DC_{max} = 600$, le débit de décodage maximal obtenu est égal à :

$$D_s = 133.\frac{50}{500} = 13,3 \quad \text{[Mbits/s]} \tag{4.5}$$

Nous constatons que ce débit est assez faible par rapport au débit de l'ordre de Gbits/s de notre objectif. La raison pour ce débit modeste est que nous utilisons pas mal des ressources du circuit FPGA Virtex-5 LX330 par une multiplication de flux de décodage d'un facteur ρ. En regardant sur le résultat de synthèse, le chemin critique prend seulement 1,2 ns pour traverser le calcul logique. Néanmoins, il nécessite 9,62 ns pour traverser le routage. Cela signifie que la majeur du temps de passage du chemin critique est contribuée par le routage (88,8%). Par conséquent, la fréquence maximale de fonctionnent est réduite. Dès lors, si nous considérons un prototype avec moins de complexité, nous pouvons obtenir une meilleure fréquence de fonctionnement en réduisant le routage. Cet espoir peut être obtenu par une réduction du nombre de flux de décodage en parallèle ρ.

Lorsque la fréquence de fonctionnement de la chaîne complète est limitée par la fréquence de fonctionnement de l'émulateur du canal, 66,18 MHz. Avec cette fréquence, le débit d'un décodeur stochastique SISO est égal à :

$$D_s = 66,18.\frac{50}{150} = 22 \quad \text{[Mbits/s]} \tag{4.6}$$

Et le débit du turbo-décodeur stochastique est donné par :

$$D_s = 66,18.\frac{50}{500} = 6,6 \quad \text{[Mbits/s]} \tag{4.7}$$

4.7 Conclusion

Nous rappelons que l'objectif principal de ce chapitre est la validation sur une plate-forme intégrant une chaîne de transmission afin d'en estimer la complexité et le débit des algorithmes et des architectures retenus pour le turbo-décodage stochastique. Dans les deux chapitres précédents, des études théoriques ont démontré l'intérêt d'une telle architecture. Cependant, un développement sur une plate-forme matérielle à base de circuits FGPA est nécessaire pour valider ces études.

Nous avons d'abord présenté le rôle d'un tel prototype ainsi que des demandes en termes de performance, de complexité et de débit. Ensuite, après avoir introduit l'interface de mesure entre le PC hôte et le prototype, nous avons présenté la plate-forme à base de circuit FPGA : la plate-forme DN9000K10PCI. Dans ce chapitre, nous avons détaillé une chaîne de transmission numérique intégrant un émetteur, un émulateur du canal selon la méthode de Wallace, et un turbo-décodeur stochastique. L'architecture de chaque bloc faisant partie de la chaîne de transmission a été détaillée. La chaîne de transmission complète a été implantée sur un seul FPGA Virtex-5 LX330. Au niveau de la synthèse sur cible FPGA, nous nous intéressons en particulier aux ressources nécessaires en termes de FlipFlop, de LUT, de BRAM ainsi que la fréquence de fonctionnement maximale pour chaque module implémenté. Des résultats de synthèse ont démontré la faisabilité de la mise en œuvre de ce turbo-décodeur stochastique en termes de haut débit potentiel. A partir des résultats de synthèse, nous avons noté que :

- L'émetteur requiert très peu de ressources matérielles : 172 FlipFlops et 182 LUTs, qui sont équivalentes à 0,09% du taux d'occupation du FPGA Virtex-5 LX330.
- L'architecture d'un tel émulateur de canal nécessite 1323 LUTs et 576 FlipFlops. En effet, l'émulateur du canal est coûteux en termes de multiplieurs qui sont utilisés par des bloc de correction pour la génération des variables gaussiennes [98].
- Du côté récepteur, le turbo-décodeur stochastique représente quasiment 100% du taux d'occupation en surface de la chaîne complète en termes de LUTs et de FlipFlops. Comme il avait été prévu, le turbo-décodeur stochastique est plus coûteux en surface que les autres modules de la chaîne. Il représente également 49% de la complexité des ressources disponibles du Virtex-5 LX330 en termes de LUTs et 25% en termes de FlipFlops. Pour un code de petite taille (50 bits), cette complexité est grande. Ce résultat est conforme aux prévisions du premier et du deuxième chapitre. Si nous voulons obtenir un nombre minium de cycles de décodage, il faut envisager d'augmenter la complexité matérielle, donc la surface.
- Au total, la chaîne de transmission utilise 58% des LUTs et 26% des FlipFlops de la surface disponible du Virtex-5 LX330.
- Au niveau de la performance du prototype, la mesure d'un TEB a permis de valider les solutions algorithmiques et architecturales retenues pour le turbo-décodeur stochastique.

Une dégradation de 0,17dB est alors introduite à TEB=10^{-6} par rapport à la performance d'un décodeur SubMAP en virgule flottante avec 6 itérations.

– Le décodeur stochastique SISO peut atteindre une fréquence maximale de 197 MHz et le turbo-décodeur stochastique peut fonctionner à une fréquence maximale de 133 MHz. Les débits maximaux correspondant à ces deux décodeurs sont respectivement égaux à 65,67 Mbps et 13,3 Mbps. Néanmoins, en raison de la limite de la fréquence de fonctionnement de l'émulateur de canal, le récepteur dans notre prototypage ne peut alors que fonctionner à une fréquence de 66,18 MHz. Ce qui permet un débit de sortie égal à 22Mbps pour un décodeur stochastique SISO et 6,6 Mbps pour un turbo-décodeur stochastique avec une taille de la trame de données égale à 50 bits et un nombre de cycles de décodage égal à 150 DCs pour un décodeur SISO et 500 DCs pour un turbo-décodeur.

Dans ce chapitre, notre architecture de turbo-décodeur stochastique a permis de valider les études algorithmiques et architecturales introduites dans les chapitres 2 et 3. Elle a fourni une performance proche de la performance de référence en langage C++. En considérant un cas d'étude : un turbo-décodeur stochastique exponentiel multi-flux avec $\rho = 8$ flux et $k = 50$ bits, nous avons démontré que notre architecture permettait une montée de débit. Malheureusement, l'objectif pour une montée de débit engendre une augmentation de la complexité en surface ce qui augmente le problème de routage et alors, réduit la fréquence maximale de fonctionnement.

Néanmoins, les débits obtenus sont encore modestes. Notre architecture du turbo-décodeur stochastique produit une fréquence maximale globale de fonctionnement modeste en comparaison avec l'état de l'art de différentes implémentations de turbo-décodeur sur un circuit FPGA. Elle utilise une grande surface, et ne produit pas un débit de l'ordre du Gbps, mais à nos connaissances, c'est la première architecture de décodage stochastique pour les turbocodes. Une réduction du chemin critique avec une analyse du placement-routage et une réduction de la complexité des traitements d'addition permettront une meilleure efficacité d'architecture pour viser le haut débit. Elle pourrait alors ouvrir une nouvelle direction de recherche dans l'objectif d'atteindre un débit de l'ordre du Gbps.

Conclusion et perspectives

Ce mémoire de thèse porte sur l'étude et l'implémentation d'algorithmes stochastiques dédiés au domaine des communications numériques et en particulier des décodeurs de codes correcteurs d'erreurs. Il s'agit de la première étude réalisée sur le décodage stochastique des turbocodes convolutifs. Le travail a consisté d'une part en une validation algorithmique et d'autre part en une proposition d'architectures innovantes puis de leur implémentation. C'est pourquoi, un état de l'art sur ces deux axes amorce ce mémoire, afin d'exposer au mieux le cadre de nos travaux.

L'objectif principal de cette étude a été de mettre en œuvre une architecture de turbo-décodeur stochastique et de l'implémenter afin d'aboutir à une architecture à haut débit. Le turbo-décodeur stochastique se compose de deux décodeurs SISO séparés par des entrelaceurs/désentrelaceurs. Chaque décodeur SISO est un décodeur convolutif stochastique. Le travail a été décomposé en trois étapes fondamentales : une étude algorithmique, une étude des solutions architecturales et une implémentation de l'architecture sur un circuit FPGA.

L'étude algorithmique (chapitre 2) a permis d'aller de la théorie du décodage stochastique à une construction des différents modules élémentaires d'un turbo-décodeur stochastique de référence. La première étape de cette étude a porté sur l'adéquation algorithme-architecture. Elle a permis de faciliter le passage vers le niveau architectural. Au cours de cette étape, le temps consacré à choisir un nombre de cycles de décodage maximal DC_{max} a été relativement important car il s'agissait de déterminer une valeur DC_{max} n'introduisant pas de dégradation en termes de performance par rapport au système de référence en virgule flottante. Lors de cette étude, une démarche progressive a été fructueuse :

- La proposition d'une architecture générale d'un turbo-décodeur stochastique a été effectuée pour aboutir à une description réalisable au niveau architectural.
- Le principe de conversion en probabilités *a priori* à partir des symboles démodulés reçus à l'entrée du récepteur a été modélisé en virgule fixe avant le traitement dans le domaine stochastique. La phase de quantification des symboles démodulés et des probabilités *a priori* a été réalisée afin de réduire la complexité du module de conversion en probabilités ainsi que la complexité de la conversion en flux stochastiques en garantissant les performances.
- L'algorithme de décodage MAP employant des opérations d'addition, de multiplication et de division a été considéré pour la construction des modules élémentaires du décodeur convolutif stochastique SISO.
- La suppression du problème de corrélation lors de la propagation des probabilités au sein du décodeur stochastique a été appliquée avec succès pour les décodeurs stochastiques SISO puis les turbo-décodeurs stochastiques. Parmi les solutions proposées dans la littérature, l'approche de mise à l'échelle des symboles démodulés *NDS* et l'approche des mémoires de type EMs ont été retenues car elles ont permis d'assurer une bonne performance de décodage.

Cette étude algorithmique a abouti à l'obtention de performance en termes de taux d'erreurs binaires - TEB, similaires à la performance d'un décodeur APP en virgule flottante pour des codes

convolutifs et à celle d'un turbo-décodeur SubMAP en virgule flottante avec 6 itérations. Les premières estimations de la complexité calculatoire et du débit pour un turbo-décodeur stochastique ont également été établies. L'architecture du turbo-décodeur proposée a une complexité compétitive par rapport à celle d'un décodeur SubMAP conventionnel. Cependant, elle a abouti à un faible débit de décodage en raison du grand nombre de cycles DC_{max}. La courbe de performance, la complexité et le débit du turbo-décodeur stochastique proposé ont dès lors été considérés comme référence pour la suite de nos travaux.

L'étude des solutions architecturales avait pour but de diminuer dans un second temps le nombre de cycles DC_{max} afin de favoriser la montée en débit. L'élaboration des différentes solutions architecturales de turbo-décodeur stochastique a permis de déterminer la complexité de la réalisation matérielle et également de donner une évaluation des débits atteignables. Lors de cette étude, nous avons procédé à trois étapes d'exploration :

- Un premier domaine de décodage stochastique a été exploré - le domaine *exponentiel* - dans lequel des opérations de type addition à plusieurs entrées ont été remplacées par d'autres opérations simples. Le choix des ordres des modules exponentiels et logarithmiques a été dicté par la recherche d'un bon compromis complexité/exactitude ou complexité/performance. Cette approche a permis de définir une architecture de décodeur convolutif exponentiel et une architecture de turbo-décodeur stochastique exponentiel qui ont produit des performances similaires à celles des décodeurs stochastiques conventionnels. L'architecture du turbo-décodeur stochastique exponentiel a une complexité plus faible et un meilleur débit par comparaison avec l'architecture du turbo-décodeur stochastique conventionnel.
- Un second domaine de décodage stochastique basé sur l'idée des auteurs présentée dans [93] - le domaine *multi-flux* - a été introduit. En représentant une même probabilité par ρ flux stochastiques en parallèle, un décodeur stochastique multi-flux peut multiplier par ρ le débit en comparaison avec un décodeur stochastique conventionnel sans dégradation de performance. Cette approche est très avantageuse : le décodeur stochastique multi-flux ne nécessite pas des vecteurs de bits aléatoires d'adresse des mémoires, contrairement au décodeur stochastique conventionnel. En revanche, l'augmentation du nombre de flux en parallèle de ρ fois engendre une augmentation d'un facteur ρ de la complexité calculatoire.
- La combinaison des deux solutions a mené à construire une architecture de turbo-décodeur stochastique, dite *turbo-décodeur stochastique exponentiel multi-flux*. Cette architecture non seulement assure une bonne performance de décodage mais aussi réduit significativement le nombre de cycles DC_{max}, produisant un débit prometteur par rapport à celui d'un turbo-décodeur stochastique conventionnel.

L'étape suivante fût l'implémentation sur une plate-forme à base de circuits FPGA Virtex-5 de la société Xilinx de deux architectures correspondant respectivement à deux décodeurs stochastiques exponentiels multi-flux : le décodeur convolutif et le turbo-décodeur. Chaque architecture a été intégrée dans une chaîne de transmission numérique complète comprenant un émetteur et un émulateur de canal sur une plate-forme de prototypage DN9000K10PCI de la société DINI-GROUP. Le prototypage a été conçu pour assurer la généricité au niveau des paramètres suivants :

- La taille de la trame d'information utile k.
- Le format de quantification du symbole reçu à l'entrée du turbo-décodeur (p,q).
- Le nombre de flux de décodage en parallèle ρ.
- La profondeur d'un vecteur de bits aléatoires W utilisé pour la conversion en flux stochastique.
- Les coefficients de correction de performance des codes $\Omega = \sigma_0^2$ et $\psi(\sigma)$.

Deux prototypages qui correspondent à deux différents types de décodeurs, décodeur convolutif stochastique et turbo-décodeur stochastique, ont été étudiés. Dans ces deux prototypes, la taille de la trame de données est égale à $k = 50$ bits et le nombre de flux de décodage en parallèle a été fixé à $\rho = 8$ en raison de la limite des ressources du circuit FPGA Virtex-5 LX330.

Les performances de ces deux prototypages ont ensuite été évaluées et comparées avec les performances correspondantes au niveau algorithmique dans le domaine stochastique conventionnel et pour un décodage classique en virgule flottante. Le prototypage d'un décodeur convolutif stochastique a donné des performances similaires. En revanche, une dégradation de 0,17 dB a été observée à un BER=10^{-6} pour le prototypage du turbo-décodeur stochastique par rapport à la performance du décodage stochastique en virgule flottante. Elle est le résultat d'une première dégradation due au choix du nombre de cycles DC_{max} pour assurer un bon compromis performance/débit. Une seconde dégradation est due à la réutilisation des bits aléatoires entre les deux décodeurs stochastiques SISO et au sein de chaque décodeur SISO. Cependant, cette légère dégradation est tolérable.

La complexité du turbo-décodeur stochastique retenue étant inférieure à la capacité du circuit FPGA, cela a permis d'intégrer la chaîne complète sur un seul FPGA Virtex-5 LX330. Au total, la chaîne de transmission utilise 58% des LUTs et 26% des FlipFlops de la surface disponible au sein d'un Virtex-5 LX330.

Le décodeur convolutif stochastique peut fonctionner jusqu'à une fréquence maximale de 197 MHz et le turbo-décodeur stochastique atteint une fréquence maximale de 133 MHz ce qui correspond respectivement à un débit maximal de 65 Mbps pour le décodeur convolutif et 13,3 Mbps pour le turbo-décodeur. En raison de la limite de la fréquence de fonctionnement de l'émulateur de canal, le récepteur dans notre prototypage ne peut fonctionner qu'à une fréquence de 66,18 MHz. Cela correspond à un débit de sortie de 22Mbps pour un décodeur convolutif stochastique et de 6,6 Mbps pour un turbo-décodeur stochastique.

Le prototype de turbo-décodeur stochastique a permis de valider les études algorithmiques et architecturales présentées au chapitre 2 et 3. Ce prototype n'a pas pu nous fournir un débit de l'ordre de la centaine de Mbits/s voire du Gbits/s comme souhaité. Cette limitation était due à l'utilisation des ressources du circuit FPGA ce qui limite la fréquence de fonctionnement de l'architecture de turbo-décodeur stochastique. Cependant, à notre connaissance, il s'agit du premier prototype pour un décodage stochastique de turbocodes. Il a démontré la faisabilité matérielle d'un turbo-décodeur stochastique en assurant une bonne performance de décodage et laisse de nombreuses voies d'optimisation.

Perspectives

Ces travaux ont permis l'extension de l'approche de décodage stochastique à n'importe quel turbo-décodeur en répondant aux exigences de performance et de complexité matérielle. Ils ont abouti à un prototype intégrant une chaîne complète de transmission numérique incluant un émetteur, un émulateur de canal et un turbo-décodeur stochastique sur un circuit FPGA Virtex-5 LX330.

Au niveau algorithmique, l'approche de décodage stochastique a été appliqué au turbo-décodage convolutif de codes binaires. Or, la modélisation du comportement d'un turbo-décodeur stochastique décrite avec le langage C++ a été écrite de manière générique pour s'adapter à différents types de symboles. Cela signifie qu'à court terme, l'approche de décodage stochastique et l'ensemble des techniques proposées dans le cadre de cette thèse, peut être étendues aux turbocodes double-binaires comme ceux spécifiés dans les normes DVB-RCT [26], et DVB-RCS [27]. Ces turbocodes sont constitués des codes circulaires, une modélisation supportant d'autres techniques de fermeture de treillis comme celle du standard de LTE [25] peut être envisagée. Mais, dans ce cas, le comportement des tels turbo-décodeurs reste à étudier du point de vue des performances et de l'architecture.

L'approche multi-flux utilisée au sein du turbo-décodeur stochastique dans le cadre de cette thèse permet de diversifier l'information stochastique des mémoires. En conséquence, elle permet aux deux décodeurs stochastiques SISO de converger rapidement en réduisant significativement le nombre de cycles DC_{max}. Bien que cette technique produise une montée notable du débit de turbo-décodage ainsi qu'une réduction intéressante de la complexité des mémoires utilisées pour la suppression du problème de corrélation, elle présente un inconvénient : l'augmentation du débit par un facteur de ρ entraîne une augmentation de ρ fois la complexité de calcul stochastique. Dès lors, nous pourrons, dans la suite de cette étude, rechercher d'autres techniques plus efficaces permettant d'optimiser l'architecture multi-flux. Par exemple, nous pourrons utiliser des différentes valeurs de ρ dans les modules Γ, A/B, Ext, Dec.

Comme dans un système de turbo-décodage stochastique, les deux décodeurs stochastiques SISO s'échangent des flux stochastiques extrinsèques pour converger, il est possible de lui associer des techniques existantes comme par exemple un critère d'arrêt en évaluant les flux stochastiques extrinsèques afin de stopper le processus de décodage stochastique plus rapidement.

Par ailleurs, une solution alternative peut être considérée. Elle consiste en des traitements sous-optimaux pour réduire la complexité du décodage stochastique basé sur l'algorithme MAP. Comme justifié par le tableau 4.3, les deux modules *Ext* au sein de deux décodeurs SISO représentent une grande complexité de ressources disponibles du circuit Virtex-5 LX330. Par conséquent, nous pourrions proposer une architecture sous-optimale pour le module *Ext* afin de traiter l'information extrinsèque, par exemple, l'architecture de traitement du module *Ext* dans le domaine de virgule fixe au lieu du traitement dans le domaine stochastique, telle que l'architecture proposée dans [86].

De plus, il est possible d'optimiser l'architecture de manière à accentuer la diminution des bits aléatoires. Pour contourner le problème d'un grand nombre de bits aléatoires nécessaires, nous pourrions proposer une autre architecture pour laquelle une meilleure distribution des bits

aléatoires serait proposée. En outre, nous pourrions intégrer une autre technique de génération des bits aléatoires au lieu d'utilisation des registres à décalage comme de véritables générateurs de valeurs aléatoires (en anglais *True Random Number Generator - TRNG*). Un des générateurs de type TRNG a été exploré par Nicolas Grégis et Arun Kumar au sein du département Electronique de Telecom Bretagne et sera envisagé dans les prochaines architectures de turbo-décodeur stochastique.

Au niveau de la mise en œuvre, nous avons retenue une cible FPGA pour établir un prototype qui valide nos solutions architecturales. Néanmoins, cette implémentation ne nous a pas donné de résultats à la hauteur de nos attentes en terme de débit et de complexité matérielle due à la latence de routage. Une analyse approfondie du placement-routage pourrait donc réduire le chemin critique et fournir une meilleure fréquence de fonctionnement. Par ailleurs, pour mesurer de manière précise l'impact sur le débit de nos approches, de futurs travaux devraient consister à implémenter et tester l'architecture de turbo-décodeur stochastique sur un circuit ASIC.

Un prolongement intéressant de ce travail à plus long terme est également envisageable, pour étendre le principe de décodage stochastique des turbocodes afin d'implémenter des systèmes traitant des fonctions logiques fiables à partir de composants moins fiables. En effet, dans la technologie CMOS (*Complementary Metal-Oxide-Semiconductor*), la plupart des composants logiques perdent en fiabilité en technologies nano-métriques (comme la future 22 nm), notamment à cause des bruits thermiques et d'alimentation. Il en résulte des erreurs transitoires qui rendent le traitement numérique non-déterministe. Plusieurs architectures de nano-dimension ont été proposées pour annuler ces erreurs transitoires. Cependant, ces architectures requièrent des fonctions redondantes fiables qui entraînent plusieurs composants supplémentaires pour chaque porte logique. Ce problème réduit alors l'efficacité de densité des composants dans une technologie nano-métrique. En 2009, C.Winstead a proposé une technique de compensation de ces erreurs transitoires qui exploite le comportement non-déterministe des composants en considérant le principe de décodage stochastique des codes LDPC [103]. Cette technique se base sur la technologie PCMOS (en anglais *probabilistic* CMOS) du pionnier K. Palem [104], dans laquelle, chaque porte logique (AND, OR, XOR,...) est moins fiable et considérée comme une porte probabiliste. L'objectif fondamental de cette technique est de coupler le codage de l'information et le traitement stochastique pour fiabiliser les circuits à moindre coût. L'idée générale est de traiter de manière fiable plusieurs opérations non-déterministes $f(x)$ en parallèle. Pour ce faire, une architecture de codage des codes LDPC est utilisée pour coder "une copie" du résultat des opérations $f(x)$ et produire des signaux de redondance. Ensuite, une architecture de décodage stochastique des codes LDPC est considérée afin de calculer les sorties fiables des opérations $f(x)$. Un avantage indéniable de cette approche est qu'elle requiert très peu de fonctions redondantes fiables. Cela signifie que l'architecture correspondante est moins complexe en comparaison avec d'autres architectures de la littérature. Nous pouvons donc envisager que le décodage stochastique de turbocodes pourrait être considéré pour implémenter des opérations fiables dans des technologies nano-électroniques peu fiables. s

Liste des publications

Article de revue avec comité de lecture

DONG Quang Trung, ARZEL Matthieu, JEGO Christophe, GROSS Warren J., "Stochastic Decoding of Turbo Codes", *IEEE Transactions On Signal Processing*, vol. 58, n° 12, December 2010, pp 6421- 6425.

Communication dans une conférence à comité de lecture

DONG Quang Trung, ARZEL Matthieu, JEGO Christophe, "Design and FPGA Implementation of Stochastic Turbo Decoder", in NEWCAS 2011, *IEEE 9th International NEWCAS conference*, Bordeaux, France, June 17-20, 2011.

DONG Quang Trung, ARZEL Matthieu, JEGO Christophe, "Stochastic Implementation of Turbo Decoder", *GDR SOC-SIP : System on chip - system in package : 5ème colloque national*, Lyon, France, 15-17 Juin, 2011.

DONG Quang Trung, ARZEL Matthieu, JEGO Christophe, "Le principe du calcul stochastique appliqué au décodage des turbocodes", *GRETSI 2011 : XXIIIème colloque sur le traitement du signal et des images*, Bordeaux, France, 5-8 Septembre, 2011.

Bibliographie

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

[2] D. MacKay and R. Neal, "Near shannon limit performance of low density parity check codes," *Electronics Letters*, vol. 33, no. 6, pp. 457 –458, mar 1997.

[3] R. Gallager, "Low-density parity-check codes," *Information Theory, IRE Transactions on*, vol. 8, no. 1, pp. 21 –28, Jan. 1962.

[4] S. Sharifi Tehrani, A. Naderi, G.-A. Kamendje, S. Hemati, S. Mannor, and W. Gross, "Majority-based tracking forecast memories for stochastic LDPC decoding," *Signal Processing, IEEE Transactions on*, vol. 58, no. 9, pp. 4883 –4896, sept. 2010.

[5] B. Gaines, "Advances in information systems science," in *chapter 2, Plenum, New York*, 1969, pp. 37–172.

[6] DVB, *Specification for the use of Video and Audio Coding in Broadcasting Applications based on the MPEG-2 Transport Stream*, v1.9.1 ed., ETSI TS 101 154, Sep. 2009.

[7] M. Bickerstaff, L. Davis, C. Thomas, D. Garrett, and C. Nicol, "A 24mb/s radix-4 logmap turbo decoder for 3gpp-hsdpa mobile wireless," in *Solid-State Circuits Conference, 2003. Digest of Technical Papers. ISSCC. 2003 IEEE International*, 2003, pp. 150 – 484 vol.1.

[8] C.-H. Lin, C.-Y. Chen, and A.-Y. Wu, "High-throughput 12-mode ctc decoder for wimax standard," in *VLSI Design, Automation and Test, 2008. VLSI-DAT 2008. IEEE International Symposium on*, april 2008, pp. 216 –219.

[9] O. Muller, A. Baghdadi, and M. Jezequel, "From parallelism levels to a multi-ASIP architecture for turbo decoding," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 17, no. 1, pp. 92 – 102, january 2009.

[10] D. Wu, R. Asghar, Y. Huang, and D. Liu, "Implementation of a high-speed parallel turbo decoder for 3GPP LTE terminals," in *ASIC, 2009. ASICON '09. IEEE 8th International Conference on*, oct. 2009, pp. 481 –484.

[11] C. Studer, C. Benkeser, S. Belfanti, and Q. Huang, "Design and implementation of a parallel turbo-decoder asic for 3GPP-LTE," *Solid-State Circuits, IEEE Journal of*, vol. 46, no. 1, pp. 8 –17, jan. 2011.

[12] Y. Sun, Y. Zhu, M. Goel, and J. Cavallaro, "Configurable and scalable high throughput turbo decoder architecture for multiple 4g wireless standards," in *Application-Specific Systems, Architectures and Processors, 2008. ASAP 2008. International Conference on*, july 2008, pp. 209 –214.

[13] C.-H. Tang, C.-C. Wong, C.-L. Chen, C.-C. Lin, and H.-C. Chang, "A 952ms/s max-log map decoder chip using radix-4$\times$4 ACS architecture," in *Solid-State Circuits Conference, 2006. ASSCC 2006. IEEE Asian*, nov. 2006, pp. 79 –82.

[14] C.-C. Wong and H.-C. Chang, "High-efficiency processing schedule for parallel turbo decoders using QPP interleaver," *Circuits and Systems I : Regular Papers, IEEE Transactions on*, vol. PP, no. 99, p. 1, 2011.

[15] C.-C. Wong, M.-W. Lai, C.-C. Lin, H.-C. Chang, and C.-Y. Lee, "Turbo decoder using contention-free interleaver and parallel architecture," *Solid-State Circuits, IEEE Journal of*, vol. 45, no. 2, pp. 422 –432, feb. 2010.

[16] S. Sharifi Tehrani, S. Mannor, and W. Gross, "Fully parallel stochastic LDPC decoders," *Signal Processing, IEEE Transactions on*, vol. 56, no. 11, pp. 5692–5703, Nov. 2008.

[17] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate (corresp.)," *Information Theory, IEEE Transactions on*, vol. 20, no. 2, pp. 284–287, Mar. 1974.

[18] C. Douillard, M. Jezequel, C. Berrou, P. Didier, and A. Picart, "Iterative correction of intersymbol interference : turbo-equalization," *European transactions on telecommunications*, vol. 6, no. 5, pp. 507 – 512, september 1995.

[19] A. Picart, P. Didier, and A. Glavieux, "Turbo-detection : a new approach to combat channel frequency selectivity," in *Communications, 1997. ICC 97 Montreal, 'Towards the Knowledge Millennium'. 1997 IEEE International Conference on*, vol. 3, jun 1997, pp. 1498–1502 vol.3.

[20] C. E. Shannon, "Certain results in coding theory for noisy channels," *Information and Control*, vol. 1, no. 1, pp. 6–25, 1957.

[21] R. Gallager, "A simple derivation of the coding theorem and some applications," *Information Theory, IEEE Transactions on*, vol. 11, no. 1, pp. 3 – 18, Jan. 1965.

[22] D. Forney, "Concatenated codes," *MA : MIT Press*, vol. 4, no. 2, p. 8374, 1966.

[23] J. P. Odenwalder, "Optimal decoding of convolutional codes," Ph.D. dissertation, University of California, Los Angeles, 1970.

[24] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Serial concatenation of interleaved codes : performance analysis, design and iterative decoding," in *Information Theory. 1997. Proceedings., 1997 IEEE International Symposium on*, Jun. 1997, p. 106.

[25] 3GPP, *3rd Generation Partnership Project, Technical Specification Group Radio Access Network, Evolved Universal Terrestrial Radio Access (E-UTRA), Multiplexing and channel coding*, v8.1.0, release 8 ed., Nov. 2007.

[26] DVB, *Interaction Channel for Digital Terrestrial Television*, v1.1.1 ed., ETSI EN 301 958, Aug. 2001.

[27] ——, *Interaction Channel for Satellite Distribution Systems*, v1.2.2 ed., ETSI EN 301 790, Dec. 2000.

[28] A. Ghosh, D. Wolter, J. Andrews, and R. Chen, "Broadband wireless access with wimax/802.16 : current performance benchmarks and future potential," *Communications Magazine, IEEE*, vol. 43, no. 2, pp. 129 –136, Feb. 2005.

[29] *Telemetry Channel Coding*, ser. Blue Book, No. 4, Consulative Committee for Space Data Systems (CCSDS) Recommendation for Space Data System Standard 101.0-B-4, May 1999.

[30] K. Narayanan and G. Stuber, "Selective serial concatenation of turbo codes," *Communications Letters, IEEE*, vol. 1, no. 5, pp. 136 –139, Sep. 1997.

[31] D. Divsalar and F. Pollara, "Serial and hybrid concatenated codes with applications," in *Proc. Int Symp. on Turbo codes and Related Topics*, 1997, pp. 80–87.

[32] S. Benedetto and G. Montorsi, "Unveiling turbo codes : some results on parallel concatenated coding schemes," *Information Theory, IEEE Transactions on*, vol. 42, no. 2, pp. 409 –428, Mar. 1996.

[33] S. Dolinar, D. Divsalar, and F. Pollara, "Turbo code performance as a function of code block size," in *Information Theory, 1998. Proceedings. 1998 IEEE International Symposium on*, Aug. 1998, p. 32.

[34] L. Perez, J. Seghers, and J. Costello, D.J., "A distance spectrum interpretation of turbo codes," *Information Theory, IEEE Transactions on*, vol. 42, no. 6, pp. 1698 –1709, Nov. 1996.

[35] C. Berrou, Y. Saouter, C. Douillard, S. Kerouedan, and M. Jezequel, "Designing good permutations for turbo codes : towards a single model," in *Communications, 2004 IEEE International Conference on*, vol. 1, Jun. 2004, pp. 341 – 345.

[36] S. Crozier and P. Guinand, "High-performance low-memory interleaver banks for turbo-codes," in *Vehicular Technology Conference, 2001. VTC 2001 Fall. IEEE VTS 54th*, vol. 4, 2001, pp. 2394 –2398 vol.4.

[37] S. Dolinar, D. Divsalar, and F. Pollara, "Weight distributions for turbo codes using random and non-random permutations," *TDA Progress Report 42-122*, August 1995.

[38] R. Pyndiah, "Near-optimum decoding of product codes : block turbo codes," *Communications, IEEE Transactions on*, vol. 46, no. 8, pp. 1003 –1010, Aug. 1998.

[39] P.Elias, "Coding for noisy channels," *IRE Convention Record*, vol. 3, pp. 37–46, Mar. 1955.

[40] C. Douillard, M. Jezequel, C. Berrou, J. Tousch, N. Pham, and N. Brengarth, "The Turbo Code Standard for DVB-RCS," in *2nd International Symposium on Turbo Codes & Related Topics, Brest, France*. ELEC - Dépt. Electronique (Institut Télécom-Télécom Bretagne), 2000, pp. 535 – 538.

[41] J. Forney, G.D., "The viterbi algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268 – 278, Mar. 1973.

[42] H. Ma and J. Wolf, "On tail biting convolutional codes," *Communications, IEEE Transactions on*, vol. 34, no. 2, pp. 104 – 111, feb 1986.

[43] C. Berrou, C. Douillard, and M. Jezequel, "Designing turbo codes for low error rates," in *Turbo Codes in Digital Broadcasting - Could It Double Capacity ? (Ref. No. 1999/165), IEE Colloquium on*, 1999, pp. 1/1 –1/7.

[44] ——, “Concaténation parallèle multiple de codes convolutifs récursifs systématiques circulaires,” *Annales des télécommunications*, vol. 54, no. 3-4, pp. 166 – 172, 1999.

[45] S. S. Pietrobon, “Super codes : A flexible multi-rate coding system,” *Int. Symp. on Turbo Codes & its Related Topics*, pp. 141–148, Sep. 2000.

[46] J. Wozencraft, “Sequential decoding for reliable communication,” IRE Natl. Conv. Rec., vol. 5, pt.2, pp. 11 – 25, 1957.

[47] R. Fano, “A heuristic discussion of probabilistic decoding,” *Information Theory, IEEE Transactions on*, vol. 9, no. 2, pp. 64 – 74, Apr. 1963.

[48] A. Viterbi, “Error bounds for convolutional codes and an asymptotically optimum decoding algorithm,” *Information Theory, IEEE Transactions on*, vol. 13, no. 2, pp. 260 – 269, Apr. 1967.

[49] G. Battail, “Ponderation des symboles décodés par l’algorithme de viterbi,” *Annal. Telecommunication*, vol. 42, no. 1-2, pp. 31 –38, Jan. 1987.

[50] J. Hagenauer and P. Hoeher, “A viterbi algorithm with soft-decision outputs and its applications,” in *Global Telecommunications Conference, 1989, and Exhibition. Communications Technology for the 1990s and Beyond. GLOBECOM ’89., IEEE*, Nov. 1989, pp. 1680 – 1686 vol.3.

[51] J. Anderson and S. Hladik, “Tailbiting map decoders,” *Selected Areas in Communications, IEEE Journal on*, vol. 16, no. 2, pp. 297 –302, feb 1998.

[52] P. Robertson, E. Villebrun, and P. Hoeher, “A comparison of optimal and sub-optimal map decoding algorithms operating in the log domain,” in *Communications, 1995. ICC ’95 Seattle, ’Gateway to Globalization’, 1995 IEEE International Conference on*, vol. 2, Jun 1995, pp. 1009–1013 vol.2.

[53] D. Gnaedig, “High-speed decoding of convolutional turbo codes,” Ph.D. dissertation, UNIVERSITÉ DE BRETAGNE SUD, 2005.

[54] J. Hagenauer, E. Offer, and L. Papke, “Iterative decoding of binary block and convolutional codes,” *Information Theory, IEEE Transactions on*, vol. 42, no. 2, pp. 429 –445, Mar. 1996.

[55] R. Shao, S. Lin, and M. Fossorier, “Two simple stopping criteria for turbo decoding,” *Communications, IEEE Transactions on*, vol. 47, no. 8, pp. 1117 –1120, Aug. 1999.

[56] A. Shibutani, H. Suda, and F. Adachi, “Reducing average number of turbo decoding iterations,” *Electronics Letters*, vol. 35, no. 9, pp. 701 –702, Apr. 1999.

[57] Y. Wu, B. Woerner, and W. Ebel, “A simple stopping criterion for turbo decoding,” *Communications Letters, IEEE*, vol. 4, no. 8, pp. 258 –260, Aug. 2000.

[58] B.-S. Shim, D.-H. Jeong, S.-J. Lim, and H.-Y. Kim, “A new stopping criterion for turbo codes,” in *Advanced Communication Technology, 2006. ICACT 2006. The 8th International Conference*, vol. 2, Feb. 2006, pp. 5 pp. –1111.

[59] S. ten Brink, “Convergence behavior of iteratively decoded parallel concatenated codes,” *Communications, IEEE Transactions on*, vol. 49, no. 10, pp. 1727 –1737, Oct. 2001.

[60] L. Kocarev, F. Lehmann, G. Maggio, B. Scanavino, Z. Tasev, and A. Vardy, "Nonlinear dynamics of iterative decoding systems : analysis and applications," *Information Theory, IEEE Transactions on*, vol. 52, no. 4, pp. 1366 – 1384, Apr. 2006.

[61] D. Divsalar, S. Dolinar, and F. Pollara, "Iterative turbo decoder analysis based on density evolution," *Selected Areas in Communications, IEEE Journal on*, vol. 19, no. 5, pp. 891 –907, May 2001.

[62] O. Muller, "Archtectures multiprocesseurs monopuces génériques pour turbo-communications haut débit," Ph.D. dissertation, UNIVERSITÉ DE BRETAGNE SUD, 2007.

[63] M. Arzel, C. Lahuec, F. Seguin, D. Gnaedig, and M. Jezequel, "Semi-iterative analog turbo decoding," *Circuits and Systems I : Regular Papers, IEEE Transactions on*, vol. 54, no. 6, pp. 1305 –1316, june 2007.

[64] C.-C. Wong, M.-W. Lai, C.-C. Lin, H.-C. Chang, and C.-Y. Lee, "Turbo decoder using contention-free interleaver and parallel architecture," *Solid-State Circuits, IEEE Journal of*, vol. 45, no. 2, pp. 422 –432, feb. 2010.

[65] J.-H. Kim and I.-C. Park, "A unified parallel radix-4 turbo decoder for mobile wimax and 3gpp-lte," in *Custom Integrated Circuits Conference, 2009. CICC '09. IEEE*, sept. 2009, pp. 487 –490.

[66] C.-C. Wong and H.-C. Chang, "Reconfigurable turbo decoder with parallel architecture for 3gpp lte system," *Circuits and Systems II : Express Briefs, IEEE Transactions on*, vol. 57, no. 7, pp. 566 –570, july 2010.

[67] T. Mohsenin, D. Truong, and B. Baas, "A low-complexity message-passing algorithm for reduced routing congestion in ldpc decoders," *Circuits and Systems I : Regular Papers, IEEE Transactions on*, vol. 57, no. 5, pp. 1048 –1061, may 2010.

[68] B. Brown and H. Card, "Stochastic neural computation. I. computational elements," *Computers, IEEE Transactions on*, vol. 50, no. 9, pp. 891–905, Sep 2001.

[69] A. Dinu, M. Cirstea, and M. McCormick, "Stochastic implementation of motor controllers," vol. 2, 2002, pp. 639 – 644 vol.2.

[70] A. Rapley, C. Winstead, V. Gaudet, and C. Schlegel, "Stochastic iterative decoding on factor graphs," in *Turbo Codes and Related Topics, In Proceedings. 3rd International Symposium on*, 2003, pp. 507–510.

[71] V. Gaudet and A. Rapley, "Iterative decoding using stochastic computation," *Electronics Letters*, vol. 39, no. 3, pp. 299–301, Feb. 2003.

[72] W. Gross, V. Gaudet, and A. Milner, "Stochastic implementation of LDPC decoders," in *Signals, Systems and Computers, 2005. Conference Record of the Thirty-Ninth Asilomar Conference on*, 28 2005-Nov. 1 2005, pp. 713–717.

[73] C. Winstead, V. Gaudet, A. Rapley, and C. Schlegel, "Stochastic iterative decoders," in *Information Theory, 2005. ISIT 2005. Proceedings. International Symposium on*, Sept. 2005, pp. 1116–1120.

[74] S. Sharifi Tehrani, W. Gross, and S. Mannor, "Stochastic decoding of LDPC codes," *Communications Letters, IEEE*, vol. 10, no. 10, pp. 716–718, Oct. 2006.

[75] A. Dinu, M. Cirstea, and M. McCormick, "Stochastic implementation of motor controllers," in *Industrial Electronics, 2002. ISIE 2002. Proceedings of the 2002 IEEE International Symposium on*, vol. 2, 2002, pp. 639 – 644 vol.2.

[76] S. S. Tehrani, S. Mannor, and W. J. Gross, "An area-efficient fpga-based architecture for fully-parallel stochastic LDPC decoding," in *Signal Processing Systems, 2007 IEEE Workshop on*, Oct. 2007, pp. 255–260.

[77] C. Berrou, K. A. Cavalec, M. Arzel, A. Glavieux, M. Jezequel, C. Langlais, R. L. Bidan, S. Saoudi, G. Battail, E. Boutillon, Y. Saouter, E. Maury, C. Laot, S. Kerouedan, F. Guilloud, and C. Douillard, *Codes et turbocodes (sous la direction de Claude Berrou)*, ser. Iris. Paris : Springer, 2007.

[78] T. Richardson and R. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *Information Theory, IEEE Transactions on*, vol. 47, no. 2, pp. 599 –618, Feb. 2001.

[79] F. Quaglio, F. Vacca, C. Castellano, A. Tarable, and G. Masera, "Interconnection framework for high-throughput, flexible ldpc decoders," in *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*, vol. 2, Mar. 2006, p. 6 pp.

[80] A. Blanksby and C. Howland, "A 690-mw 1-gb/s 1024-b, rate-1/2 low-density parity-check code decoder," *Solid-State Circuits, IEEE Journal of*, vol. 37, no. 3, pp. 404 –412, Mar. 2002.

[81] S. Tehrani, S. Mannor, and W. Gross, "Survey of stochastic computation on factor graphs," in *Multiple-Valued Logic, 2007. ISMVL 2007. 37th International Symposium on*, May 2007, pp. 54–54.

[82] C. Winstead, "Error-control decoders and probabilistic computation," *Tohoku Univ. 3rd SOIM-COE Conf., Sendai, Japan*, Oct. 2005.

[83] S. Sharifi Tehrani, C. Jego, B. Zhu, and W. Gross, "Stochastic decoding of linear block codes with high-density parity-check matrices," *Signal Processing, IEEE Transactions on*, vol. 56, no. 11, pp. 5733 –5739, Nov. 2008.

[84] G. Sarkis, S. Mannor, and W. Gross, "Stochastic decoding of LDPC codes over GF(q)," in *Communications, 2009. ICC '09. IEEE International Conference on*, june 2009, pp. 1–5.

[85] W. Gross, V. Gaudet, and A. Milner, "Stochastic implementation of LDPC decoders," in *Signals, Systems and Computers, 2005. Conference Record of the Thirty-Ninth Asilomar Conference on*, Nov. 2005, pp. 713 –717.

[86] F. Leduc-Primeau, S. Hemati, W. Gross, and S. Mannor, "A relaxed half-stochastic iterative decoder for ldpc codes," in *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*, 30 2009-dec. 4 2009, pp. 1 –6.

[87] G. Sarkis, S. Hemati, S. Mannor, and W. Gross, "Relaxed half-stochastic decoding of LDPC codes over GF(q)," in *Communication, Control, and Computing (Allerton), 2010 48th Annual Allerton Conference on*, Oct. 2010, pp. 36 –41.

[88] S. Tehrani, A. Naderi, G.-A. Kamendje, S. Mannor, and W. Gross, "Tracking forecast memories in stochastic decoders," in *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*, april 2009, pp. 561 –564.

[89] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *Information Theory, IEEE Transactions on*, vol. 42, no. 2, pp. 429 –445, mar 1996.

[90] G. Montorsi and S. Benedetto, "Design of fixed-point iterative decoders for concatenated codes with interleavers," *Selected Areas in Communications, IEEE Journal on*, vol. 19, no. 5, pp. 871 –882, may 2001.

[91] C. Janer, J. Quero, J. Ortega, and L. Franquelo, "Fully parallel stochastic computation architecture," *Signal Processing, IEEE Transactions on*, vol. 44, no. 8, pp. 2110–2117, Aug 1996.

[92] I. A. Abramowitz, Milton ; Stegun, *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. New York : Dover Publications, 1970.

[93] M. Arzel, C. Lahuec, C. Jego, W. Gross, and Y. Bruned, "Stochastic multiple stream decoding of cortex codes," *Signal Processing, IEEE Transactions on*, vol. 59, no. 7, pp. 3486 –3491, july 2011.

[94] V. Gaudet, R. Gaudet, and P. Gulak, "Programmable interleaver design for analog iterative decoders," *Circuits and Systems II : Analog and Digital Signal Processing, IEEE Transactions on*, vol. 49, no. 7, pp. 457 – 464, jul 2002.

[95] V. E. Beneš, "Optimal rearrangeable multistage connecting networks," *Bell System Technical*, vol. 43, pp. 1641 – 1656, 1964.

[96] "DN9000K10PCI XILINX Virtex-5 Based ASIC Prototyping Engine."

[97] C. S. Wallace, "Fast pseudorandom generators for normal and exponential variates," *ACM Trans. Math. Softw.*, vol. 22, pp. 119–127, March 1996.

[98] O. D. S. Gonzalez, M. Arzel, C. Jego, A. Garcia, and M. Guerrero, "Design and implementation of a MIMO channel emulator onto FPGA device," in *IWS'09 : 1XV proyecto Iberchip*, 2009.

[99] D.-U. Lee, J. Villasenor, W. Luk, and P. Leong, "A hardware gaussian noise generator using the box-muller method and its error analysis," *Computers, IEEE Transactions on*, vol. 55, no. 6, pp. 659 –671, june 2006.

[100] D.-U. Lee, W. Luk, J. Villasenor, G. Zhang, and P. Leong, "A hardware gaussian noise generator using the wallace method," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 13, no. 8, pp. 911 –920, aug. 2005.

[101] R. D'Agostino and M. Stephens, "Goodness-of-fit techniques," *Marcel-Dekker, New York*, 1986.

[102] R. Santoro, O. Sentieys, and S. Roy, "On-the-fly evaluation of fpga-based true random number generator," in *VLSI, 2009. ISVLSI '09. IEEE Computer Society Annual Symposium on*, may 2009, pp. 55 –60.

[103] C. Winstead and S. Howard, “A probabilistic ldpc-coded fault compensation technique for reliable nanoscale computing,” *Circuits and Systems II : Express Briefs, IEEE Transactions on*, vol. 56, no. 6, pp. 484 –488, june 2009.

[104] K. Palem, “Energy aware computing through probabilistic switching : a study of limits,” *Computers, IEEE Transactions on*, vol. 54, no. 9, pp. 1123 – 1137, sept. 2005.