UNIVERSITÉ DE MONTRÉAL

CONCEPTION ET PROTOTYPAGE DE DÉCODEURS À SEUIL ITÉRATIF À HAUT DÉBIT

ABBAS NEMR
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE ÉLECTRIQUE)
AOÛT 2009

Your file Votre référence
ISBN: 978-0-494-53915-6
Our file Notre référence
ISBN: 978-0-494-53915-6

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

CONCEPTION ET PROTOTYPAGE DE DÉCODEURS À SEUIL ITÉRATIF À HAUT DÉBIT

présenté par: NEMR Abbas

en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de:

M. DAVID Jean-Pierre, Ph.D., président

M. SAWAN Mohamad, Ph.D., membre et directeur de recherche

M. CARDINAL Christian, Ph.D., membre et codirecteur de recherche

M. HACCOUN David, Ph.D., membre

À celui que tout le monde attend...

REMERCIEMENTS

Je tiens à remercier par ces quelques lignes les personnes qui m'ont permis de mener à bien cette recherche.

J'aimerais au début remercier mon directeur de recherche, le Dr. Mohamad Sawan, ainsi que mon co-directeur de recherche, le Dr. Christian Cardinal, dont leurs précieux conseils m'ont aidé tout au long de mon travail. Je teins à remercier aussi le Dr. David Haccoun ainsi que mes directeurs pour l'aide financière qu'ils m'ont accordée.

Je tiens aussi à remercier tous mes collègues du laboratoire pour toutes ces discussions quotidiennes enrichissantes. Je pense particulièrement à Saeid, Amine, Mostapha, Mohamad, Éric et Bassam.

Finalement, je tiens à remercier ma famille et en particulier mon père, ma mère et ma femme ainsi que mes amis pour leur soutien.

RÉSUMÉ

Le travail de recherche effectué dans ce mémoire concerne la conception et le prototypage sur FPGA de décodeurs à seuil itératif à haut débit.

Le décodage à seuil itératif utilisé conjointement avec des codes convolutionnels doublement orthogonaux (CDO) constitue un nouvel algorithme performant de correction d'erreurs. Cependant, les implémentations de ce type de décodeur qui ont été réalisées à ce jour n'atteignent que 42 Mbps de débit et ne supportent qu'un seul taux de codage $R = 1/2$. Afin de satisfaire les critères de communication tels que, par exemple, ceux spécifiés pour des systèmes WiMAX mobiles (IEEE 802.16e), le débit binaire à la sortie du décodeur des codes CDO doit être augmenté. En outre, les codeurs/décodeurs utilisés doivent avoir la capacité de fonctionner à plusieurs taux de codages.

Dans la première partie de ce travail, l'architecture du décodeur à seuil itératif a été détaillée. Ainsi, les composants qui limitent le débit du décodeur ont été améliorés. De plus, une technique efficace de pipelinage a été développée et appliquée à l'architecture du décodeur à seuil afin d'en augmenter son débit. Les modules de perforation ont été développés et intégrés dans le codeur CDO et le décodeur à seuil itératif. Par conséquent, le décodeur à seuil itératif qui a été développé et implémenté sur FPGA est capable de supporter les taux de codages allant de $R = 1/2$ à $R = 8/9$ à un débit qui peut atteindre 269 Mbps.

Dans la deuxième partie de ce travail, une autre approche qui consiste à étudier les codes CDO à multi-registres à décalage (M-CDO) a été utilisée afin d'augmenter le débit du décodeur. Les codes M-CDO sont générés par un codeur à M registres à décalage et offrent un décodage à architecture parallèle. Par conséquent, le débit du décodeur est M fois plus grand qu'un décodeur des codes CDO. Cependant, la complexité du décodeur

est M fois plus grande ce qui a été compromis par la simplification des codes M-CDO. Le prototype du décodeur à seuil itératif des codes M-CDO implémenté est capable d'atteindre des très hauts débits allant jusqu'à 1.3 Gbps pour certains codes.

Au cours de ces étapes, plusieurs nouveaux générateurs de codes CDO et M-CDO ont été introduits. De plus, les résultats expérimentaux des décodeurs ont été exposés. Ces résultats ont montré que les codes M-CDO simplifiés offrent un compromis attrayant entre débit, complexité et performances d'erreur au décodage.

ABSTRACT

The research presented in this master's thesis deals with the design and prototyping on FPGA of very high throughput iterative threshold decoders.

The iterative threshold decoding used in conjunction with convolutional doubly orthogonal (CDO) codes constitutes a new powerful error correcting algorithm. However, the implementations of this type of decoder that have been completed to date can reach a low throughput of 42 Mbps and support only one coding rate $R = 1/2$. In order to meet the communication systems criteria, such as those specific to mobile WiMAX systems (IEEE 802.16e), the throughput at the output of the decoder of CDO codes must be increased. In addition, the used encoders/decoders must have the capacity to operate at multiple coding rates.

In the first part of this work, the architecture and the main components of the iterative decoder threshold are presented. Later, the cited components that limit the throughput of the decoder have been improved. In addition, an effective pipelining technique was developed and applied to the architecture in order to increase its throughput. The new design which was implemented on FPGA is able to handle coding rates ranging from $R = 1/2$ to $R = 8/9$ at a throughput that can reach up to 269 Mbps.

In the second part of this work, another approach that consists of studying multi-shift registers CDO (M-CDO) encoders was used to increase the throughput of the decoder. M-CDO codes are generated by a M-shift registers coder and decoded using a parallel architecture. Thus, the throughput of the decoder is M times larger than that of the decoder of CDO codes. However, the new decoder suffers from complexity (M times) which could be reduced by simplifiying M-CDO codes. The new pipelined parallel decoder achieves a very high throughput of 1.3 Gbps for specific codes.

In this work, several new generators of CDO and M-CDO codes have been introduced. Moreover, experimental results of the implemented decoders are presented. These results prove that the simplified M-CDO codes offer attractive trade-off between throughput, complexity and error performances at decoding.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ANNEXES

LISTE DES SIGLES ET DES SYMBOLES

Sigles

ASIC	Application-Specific Integrated Circuit
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
BPSK	Binary Phase Shift Keying
BSC	Binary Symmetric Channel
C2	format binaire Complèment à 2
CDO	Convolutional Doubly Orthogonal
CP	coefficient de ponderation
DCM	Digital Clock Manager
DSI	Décodeur à Seuil Itératif
EEP	Equal Error Protection
FF	Flip-Flop
FPGA	Field Programmable Gate Array
LDPC	Low Density Parity Check code
LSB	Low Significant Bit
LUT	Look-Up Table
M-CDO	Multi-shift registers Convolutional Doubly Orthogonal
M-CPDO	Multi-shift registers Convolutional Partially Doubly Orthogonal
MSB	Most Significant Bit
PCDO	Punctured Convolutional Doubly Orthogonal
SA	format binaire Signe-Amplitude
SRL16	Shift Registre Look-Up Table
UEP	Unequal Error Protection

Symboles

α_J	Span ou mémoire du codeur convolutionnel
C_{it}	Complexité par itération
D	Débit du décodeur
D_{it}	Débit par itération
E_b	Énergie du symbole binaire u_i
f	fréquence d'opération du décodeur
J	Nombre minimal des équations de parité utilisées pour décoder un bit d'information à l'intérieur du décodeur (codes non perforés)
J_{RA}	Nombre maximal de connexions existant entre un registre à décalage et un additionneur modulo 2 dans le codeur
M	Nombre de registres à décalage du codeur
N	Nombre d'itérations du décodeur
R	Taux de codage
R_{CP}	Résolution binaire du coefficient de ponderation
RDC_{it}	Rapport débit sur complexité par itération
$RInt$	Résolution interne du décodeur
t_{co}	Clock to out time
t_r	Route time
t_{su}	Setup time
$\oplus$	Somme modulo 2
$\diamond$	Opérateur addmin
$\binom{n}{k}$	Combinaison C_n^k, choisir k objets parmi n objets
$\lceil x \rceil$	Le plus grand entier supérieur à x

CHAPITRE 1

INTRODUCTION

1.1 Motivation

Depuis plusieurs années, les communications électroniques sont devenues un besoin essentiel dans notre société moderne. Grâce à de grandes avancées technologiques dans le domaine du micro-électronique, nous sommes aujourd'hui temoin de l'utilisation grandissante des ordinateurs portables et des téléphones cellulaires intelligents. La diversité des applications dont ces technologies peuvent supporter (Internet, Voix, Video, MP3...) a encouragé les consommateurs "mobiles" ou fixes à explorer les nouveaux horizons de services offerts. Cependant, les fournisseurs de services doivent innover en permanence pour rester attrayants, et afin de répondre aux exigences qui en découlent. Ils ont donc besoin de transmettre avec grande efficacité les informations demandées par les clients, sans erreurs et à des débits remarquablement élevés. Pour atteindre l'efficacité requise, un des moyens utilisés consiste à considérer le codage correcteur d'erreur.

L'année 1993 a été marquée par la découverte des codes correcteurs d'erreur "Turbo" [1] qui ont permis, grâce à leur décodage itératif Turbo, de s'approcher de très prêt des limites théoriques de transmission prédites par Shannon [2]. En 1995, les codes LDPC ont été redécouverts [3]. Le processus du décodage itératif des codes LDPC ainsi introduits a permis d'atteindre des performances d'erreurs aussi importantes que celles des codes Turbo. Toutefois, la complexité matérielle engendrée par les décodeurs itératifs des codes Turbo et LDPC a constitué un désavantage qui s'avère limiter l'utilisation à grande échelle de ces algorithmes dans les systèmes de communication à hauts débits.

Le décodage à seuil itératif de codes convolutionnels doublement orthogonaux (CDO) a été introduit récemment par Cardinal *et al.* [4] comme un nouvel algorithme performant de correction d'erreurs. Bien que beaucoup d'efforts aient été déployés pour implémenter et améliorer cet algorithme, les implémentations de ce type de décodeur qui ont été réalisées à ce jour ne sont pas suffisamment rapides pour les applications actuelles en télécommunications. En effet, le débit binaire à l'entrée/sortie du décodeur doit être augmenté afin de satisfaire les critères de communication tels que, par exemple, ceux spécifiés par la norme Wimax [5]. Dans le cadre de ce projet, les objectifs concernent la conception et le prototypage de décodeurs à seuil itératifs à hauts débits. Les différents facteurs qui limitent le débit des décodeurs ont été identifiés. De plus, une nouvelle architecture parallèle qui permet d'augmenter le débit du décodeur à seuil itératif sans exploser sa complexité a été introduite.

1.2 Contributions

Plusieurs contributions ont été apportées dans ce mémoire.

1. Conception de nouvelles architectures de pondérateur et du registre à décalage élémentaire ce qui a abouti à améliorer le débit du décodeur.

2. Élaboration de la technique de pipelinage du décodeur à seuil itératif et introduction de la "capacité de pipelinage" comme un nouveau critère de sélection de générateurs des codes CDO qui permettent un décodage à hauts débits dans le cas où cette capacité est élevée.

3. Conception et prototypage du décodeur à seuil itératif (DSI) à hauts débits des codes CDO.

4. Conception des modules de perforation qui permettent au DSI de fonctionner à

plusieurs taux de codage et prototypage de la première version du DSI à hauts débits des codes CDO perforés à taux compatibles.

5. Réalisation de la recherche d'un ensemble de générateurs des codes CDO perforés à taux compatibles ayant une capacité de pipelinage élevée.

6. Introduction d'un nouveau type des codes CDO à multi-registres à décalage (M-CDO) qui permet un décodage parallèle à haut débit. Ainsi, la simplification de ces codes a été étudiée et la recherche d'un ensemble de générateurs a été effectuée.

7. Conception et prototypage du DSI à hauts débits des codes M-CDO.

8. Génération et comparaison des résultats expérimentaux des codes CDO et M-CDO.

1.3 Organisation du mémoire

Ce mémoire comporte six chapitres. À la suite du présent chapitre, le document se subdivise de la façon suivante.

- Le chapitre 2 présente les notions préliminaires en matière de télécommunication ainsi que celles de la technologie FPGA. Les codes LDPC et les codes Turbo sont exposés brièvement. Ensuite les codes convolutionnels doublement orthogonaux et leur processus de décodage à seuil itératif sont présentés. La technologie FPGA et quelques notions matérielles associées sont également rapportées dans ce chapitre.

- Le chapitre 3 présente l'architecture du décodeur à seuil itératif à hauts débits. Ainsi, la nouvelle technique de pipelinage est décrite. Les modules de perforation sont illustrés et l'architecture du décodeur à seuil itératif qui fonctionne à plusieurs

taux de codage est exposée. Enfin, un ensemble des nouveaux générateurs des codes convolutionnels doublement orthogonaux à taux compatible et qui permettent un décodage à haut débit est présenté.

- Le chapitre 4 définit un nouveau type des codes convolutionnels doublement orthogonaux à mutli-registres à décalage. La simplification de ces codes est étudiée et l'architecture parallèle du décodeur à seuil est exposée. Un ensemble de générateurs des codes convolutionnels doublement orthogonaux à mutli-registres à décalage est fourni à la fin du chapitre.

- Le chapitre 5 présente les résultats expérimentaux des codes convolutionnels doublement orthogonaux. Le débit, la complexité et les performances d'erreur du décodeur sont étudiés et les résultats expérimentaux sont comparés pour les différents codes.

- Finalement, le chapitre 6 résume l'ensemble des travaux effectués et propose certaines idées à étudier dans le futur.

CHAPITRE 2

NOTIONS PRÉLIMINAIRES

2.1 Introduction

Dans ce chapitre, les différents notions et concepts reliés au codage de canal dans un système de communication numérique sont abordés. De plus, les notions matérielles qui concernent le prototypage d'un système numérique sur FPGA sont également présentées. Nous nous intéressons seulement aux codes correcteurs d'erreurs utilisés conjointement avec des décodeurs itératifs tels que, les codes à faible densité de parité (LDPC, en anglais Low Density Parity Check), les codes Turbo et les codes convolutionnels doublement orthogonaux (CDO). Ces derniers sont décodés par un algorithme de décodage à seuil itératif dont l'architecture du décodeur est abordée en détail aux chapitres 3 et 4.

2.2 Codage de canal

2.2.1 Système de communication numérique

Le modèle du système de communication numérique utilisé dans ce mémoire est présenté à la figure 2.1. Une séquence de symboles binaires $\mathbf{u} = (u_0, u_1, \cdots)$ où chaque bit u_i est d'énergie E_b, est émise par une source d'information vers un codeur de taux de codage R. Le codeur ajoute selon certaines règles des symboles de redondance pour former la séquence binaire codée ou tout simplement les codes, $\mathbf{v} = (v_0, v_1, \cdots)$. Les codes sont ensuite envoyés vers un modulateur BPSK (modulation antipodale) qui traduit les symboles codés sous forme analogique avant de les transmettre dans le canal.

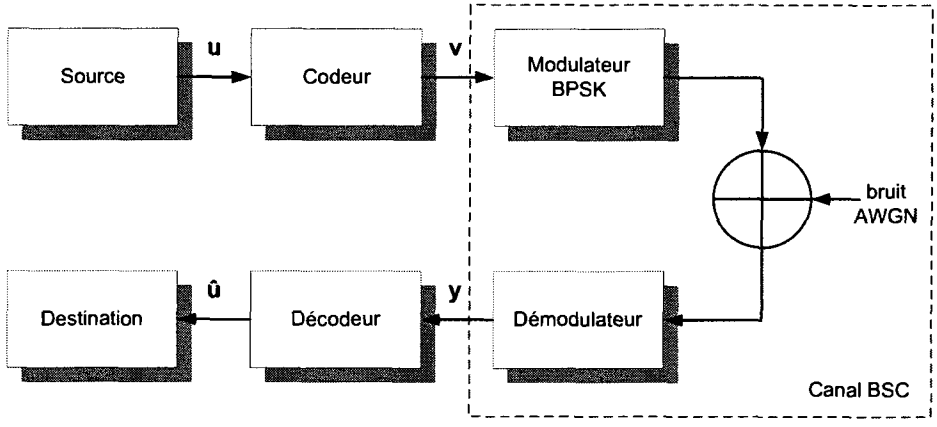

Figure 2.1: Modèle du système de communication numérique

Le canal considéré est binaire symétrique (BSC, en anglais Binary Symmetric Channel) sans mémoire à bruit blanc additif et gaussien (AWGN). Ce bruit AWGN est un processus aléatoire gaussien de moyenne nulle et de densité de puissance spectrale bilatérale égale à $N_0/2$ (Watt/Hz). À la sortie du démodulateur, la séquence reçue $\mathbf{y} = (y_0, y_1, \cdots)$ est quantifiée sur 3 bits (quantification douce), puis fournie au décodeur qui utilise les symboles redondants insérés par le codeur afin de calculer une estimation $\hat{\mathbf{u}}$ sur les symboles d'information.

Pour un canal BSC, la capacité du canal, C, qui représente le débit maximum permis a été établie par Shannon. Elle est donnée par la relation suivante [2] :

$$C = W log_2 \left(1 + \frac{P_{moy}}{W N_0}\right) (bits/s) \tag{2.1}$$

où W (Hz) est la largeur de bande utilisée et P_{moy} est la puissance moyenne d'émis-sion. Cette relation montre que si le débit d'information à la sortie de la source est inférieur à la capacité du canal C, alors il est théoriquement possible d'effectuer une transmission sans erreur en utilisant un code correcteur d'erreur approprié. Sachant (2.1), le rapport signal sur bruit minimal, $(E_b/N_0)_{min}$, à partir duquel il est théoriquement possible de faire une transmission fiable de l'information peut être calculé, et pour un canal à bruit

AWGN, il est exprimé par :

$$(E_b/N_0)_{min} \geq -1.6\ dB \tag{2.2}$$

En d'autres termes, il n'existe aucune technique de codage correcteur d'erreurs qui assure une transmission fiable si le canal est très bruité où $(E_b/N_0) < -1.6$ dB [2].

2.2.2 Principe du codage correcteur d'erreur

Le codage correcteur d'erreur permet, comme son nom l'indique, de corriger les erreurs de la transmission en intégrant selon certaines règles des symboles de redondance dans la séquence transmise dans le canal. Nous pouvons distinguer deux catégories principales du codage correcteur d'erreur : le codage bloc et le codage convolutionnel. Le codage bloc consiste à utiliser un bloc de bits de taille prédéfinie afin de produire un nombre déterminé de symboles de parité. Cependant, le codage convolutionnel consiste à calculer les symboles de parité au fur et à mesure que les bits d'information sont acheminés au codeur. Cette propriété découle du fait que le codeur convolutionnel utilise une fenêtre temporelle fixe où certaines données sont utilisées pour calculer chaque symbole de parité. Ainsi, le taux de codage, R, $R < 1$, d'un codeur (bloc ou convolutionnel) qui associe Q symboles de parité à M symboles d'information est définit par :

$$R = M/(M+Q) \tag{2.3}$$

2.2.3 Les codes LDPC

2.2.3.1 Définition des codes bloc

Les codes bloc ont été découverts par Hamming [6] dans les années 50. Un code bloc $(n, k), n > k$, est spécifié par une matrice génératrice, $\boldsymbol{G}$, de dimensions $k \times n$. Chaque mot de code est représenté par un des 2^k vecteurs de dimension n qui représente la longueur du code. Ainsi, à partir de $\boldsymbol{G}$, le codeur fait correspondre chaque mot d'information de la source $\mathbf{u} = (u_1, \cdots, u_k)$ à un mot de code $\mathbf{v} = (v_1, \cdots, v_n)$ de la façon suivante :

$$\mathbf{v} = \mathbf{u} \cdot \boldsymbol{G} \tag{2.4}$$

Le taux de codage ainsi obtenu à la sortie du codeur est donné par :

$$R = k/n \tag{2.5}$$

La matrice $\boldsymbol{G}$ est construite de façon à ce que toute combinaison linéaire des mots de code donne un autre mot de code. Par conséquent, le code est linéaire. Un exemple de la matrice génératrice $\boldsymbol{G}$ linéaire est celle du code de Hamming (7,4) :

$$\boldsymbol{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{2.6}$$

À toute matrice génératrice $\boldsymbol{G}$ d'un code en bloc, une matrice de parité $\boldsymbol{H}$ de dimensions $(n-k) \times n$ intervenant pour le décodage est calculée telle que :

$$\boldsymbol{G} \cdot \boldsymbol{H}^T = 0 \tag{2.7}$$

Soit pour la matrice $\boldsymbol{G}$ de (2.6) :

$$\boldsymbol{H} = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{2.8}$$

La matrice $\boldsymbol{H}$ sert au niveau du décodeur à détecter et corriger les erreurs en vérifiant si la séquence reçue, $\mathbf{y}$, est valide. Pour cela, il faut satisfaire :

$$\mathbf{y} \cdot \boldsymbol{H}^T = \mathbf{u} \cdot \boldsymbol{G} \cdot \boldsymbol{H}^T = 0 \tag{2.9}$$

2.2.3.2 Les codes bloc à faible densité de parité

Les codes à faible densité de parité, LDPC, sont des longs codes linéaires bloc définis par l'ensemble de $(n-k)$ équations de parité, $\mathbf{v} \cdot \boldsymbol{H}^T = 0$. $\boldsymbol{H}$ est la matrice de parité de dimensions $(n-k) \times n$ dont un exemple est montré par :

$$\boldsymbol{H} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & \cdots & \cdots & \cdots & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & \cdots & \cdots & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & \cdots & \cdots & \cdots & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & \cdots & \cdots & \cdots & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & \cdots & \cdots & \cdots & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & \cdots & \cdots & \cdots & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \cdots & \cdots & \cdots & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & \cdots & \cdots & \cdots & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & \cdots & \cdots & \cdots & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & \cdots & \cdots & \cdots & 0 & 1 \end{bmatrix} \tag{2.10}$$

Les codes LDPC ont été découverts par Gallager en 1962 [7]. Pour que les codes LDPC soient performants, la matrice de parité de ces codes doit être très creuse ou à faible densité de 1's ce qui explique leur appelation. Gallager a proposé une méthode de décodage itératif pour ses codes. Toutefois, la nécessité d'une grande complexité de calculs pour le codeur et le décodeur n'a pas permis d'exploiter les codes LDPC pendant plusieurs années.

Ce n'est qu'en 1995 que Mackay et Neal ont redécouvert les codes LDPC [3, 8, 9]. Ils ont effectué le décodage des codes LDPC en utilisant l'algorithme de décodage itératif "*belief propagation*" (BP) de Pearl [10] qui n'est qu'une version itérative de l'algorithme "*somme-produit*" [11]. À chaque itération, l'algorithme BP consiste à affiner davantage les probabilités a posteriori $\{P(\mathbf{v}/\mathbf{y})\}$ où $\mathbf{v}$ est un mot du code et $\mathbf{y}$ est la séquence reçue du canal. Pour ce faire, le décodeur calcule les probabilités correspondantes aux équations de contrôle de parité dans une première étape, puis dans une deuxième étape, il propage les probabilités et fait la mise à jour des symboles à décoder. Cette dernière étape se fait de façon à ce que les équations de parité qui contiennent le symbole courant à décoder ne soient pas incluses.

Les premiers codes LDPC développés par Gallager étaient des codes réguliers. Les codes LDPC réguliers sont construits de façon à ce que la matrice $\boldsymbol{H}$ ait un nombre constant de 1's sur les lignes ainsi que sur les colonnes. En contrepartie, les codes LDPC irréguliers [12] ne respectent pas cette propriété de régularité.

2.2.4 Les codes Turbo

En 1993, Berrou, Glavieux et Thitimajshima ont introduit une nouvelle technique du décodage itératif baptisée *Turbo* [1]. Cette technique qui utilise un codage concaténé en parallèle permet d'atteindre de très bonnes performances d'erreur s'approchant étroitement de la limite théorique de Shannon [2]. Dans les deux sous-sections suivantes, les processus de codage et de décodage turbo sont décrits brièvement.

2.2.4.1 Principe du codage Turbo

Un codeur turbo de taux de codage $R = 1/3$ est illustré à la figure 2.2. Ce codeur est composé de deux codeurs convolutionnels montés en parallèle et séparés par un entrelaceur (E) qui permute les symboles d'information de sorte que les deux codeurs n'encodent pas la même séquence d'information, $\mathbf{u} \neq \mathbf{u}'$. La taille de l'entrelaceur peut être très élevée, de l'ordre de plusieurs milliers de bits, ce qui permet de minimiser la probabilité d'erreurs au décodage [13]. En effet, l'amélioration des performances d'erreur des codes Turbo est quasi proportionnelle à la taille de l'entrelacement [14]. Les taux de codage $R > 1/3$ peuvent être obtenus par l'application de la technique de perforation au codeur turbo [15].

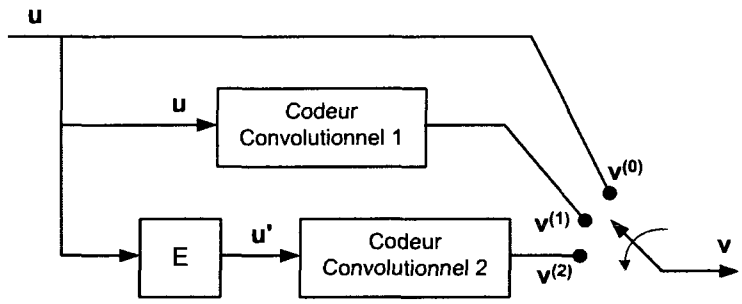

Figure 2.2: Schéma bloc du codeur *Turbo*, $R = 1/3$

2.2.4.2 Décodeur itératif Turbo

Le schéma bloc du décodeur turbo est montré à la figure 2.3. Le décodeur turbo se compose de deux décodeurs, des entrelaceurs identiques à celui utilisé par le codeur turbo et d'un délaceur (E^{-1}) qui effectue l'opération inverse de l'entrelaceur.

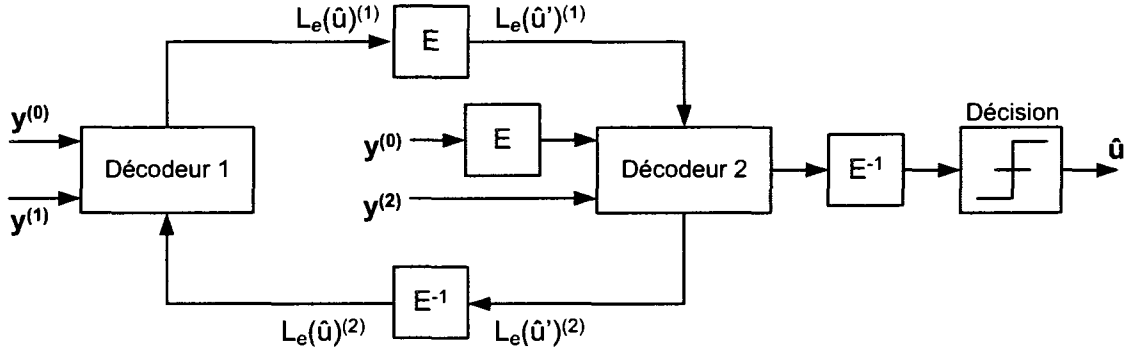

Figure 2.3: Schéma bloc du décodeur itératif *Turbo*, $R = 1/3$

Le principe du décodage turbo repose sur l'échange d'information entre les deux décodeurs qui utilisent généralement les algorithmes SOVA (en anglais Soft Output Viterbi Algorithm) [16] ou BCJR [17]. À la première itération, le premier décodeur génère la valeur extrinsèque $L_e(\hat{u}_i)^{(1)}$ à partir des sorties du canal $y_i^{(0)}$ et $y_i^{(1)}$ associées à $v_i^{(0)}$ et $v_i^{(1)}$ du codeur. Le deuxième décodeur utilise $L_e(\hat{u}_i)^{(1)}$ entrelacé, soit $L_e(\hat{u}_i')^{(1)}$, ainsi que $y_i^{(0)}$ et $y_i^{(2)}$ comme informations a priori afin de calculer $L_e(\hat{u}_i')^{(2)}$ qui sera retourné au premier décodeur en passant par le délaceur à l'itération suivante. Le premier décodeur

turbo proposé dans [1] utilise un entrelaceur pseudo-aléatoire de taille 64 kbits et deux décodeurs BJCR. Ce décodeur a été capable d'atteindre une probabilité d'erreur de 10^{-5} à 0.7 dB de la capacité du canal après 18 itérations.

2.2.5 Les codes CDO

Les performances remarquables de deux techniques des codage du canal, LDPC et Turbo, présentées dans les sections précédentes sont associées à une complexité matérielle au décodage itératif qui pourrait compromettre leurs avantages. En effet, afin de minimiser la probabilité d'erreur, les codes LDPC doivent avoir une longueur de code importante ainsi qu'un trés grand nombre d'itérations, tandis que les codes Turbo ont besoin d'un entrelaceur ayant une grande taille, conduisant ainsi à une complexité très élevée du décodeur itératif.

Une des solutions à ce problème de complexité consiste à réitérer les codes plusieurs fois sur un petit nombre d'itérations implémentées physiquement. Cependant, le débit à la sortie du décodeur sera diminué, comme dans [18] où le débit a été réduit à 9 Mbps seulement dans certains cas. Ces compromis entre complexité/débit/performance du décodeur a motivé les chercheurs à réaliser plusieurs travaux sur cette problématique [19–22].

Dans le but d'offrir un meilleur compromis, les auteurs de [23], [24] et [25] ont proposé une nouvelle technique de codage du canal inspirée par le décodage itératif Turbo et basée sur le concept du décodage à seuil des codes convolutionnels simplement orthogonaux (CSO) de Massey [26]. Les nouveaux codes ainsi introduits doivent respecter certaines conditions de double orthogonalité afin de permettre un décodage à seuil itératif sans entrelacement à la réception. La description de ces codes convolutionnels doublement orthogonaux (CDO) ainsi que leur décodage à seuil itératif, est présentée dans les

sous-sections suivantes.

2.2.5.1 Définition des codes CDO

En principe, un codeur convolutionnel utilise les symboles d'information courants et passés afin de calculer un nouveau symbole de parité. Les symboles d'information sont préservés dans un registre à décalage qui agit comme fenêtre temporelle. Un codeur convolutionnel est dit systématique si l'information présente à l'entrée du codeur est retenue dans la séquence codée à sa sortie sans être modifiée. La figure 2.4 montre un exemple d'un codeur convolutionnel systématique de taux de codage $R = 1/2$. Un tel codeur peut être décrit en utilisant l'ensemble des positions de connexions $A = \{\alpha_1, \cdots, \alpha_J\}$ où J représente le nombre de ces connexions. Chaque élément α_j, $1 \leq j \leq J$, représente la position sur le registre à décalage du codeur où une connexion le relie au sommateur modulo 2 qui calcule le symbole de parité. La longueur du registre à décalage correspond alors à α_J qui représente la longueur de la mémoire ou le "Span" du codeur.

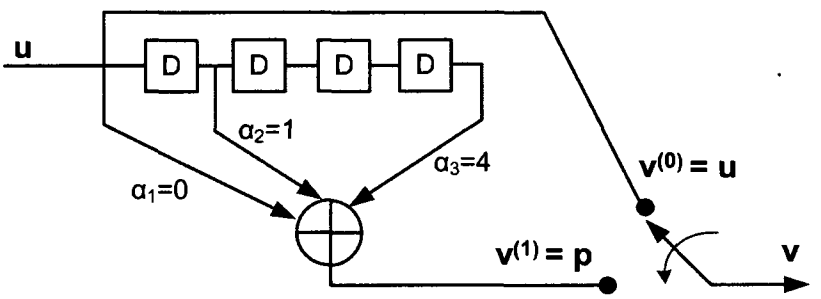

Figure 2.4: Codeur convolutionnel systématique, $R = 1/2$, $\alpha_J = 4$, $A = \{0, 1, 4\}$

Par convention, l'ensemble des positions de connexions A est appelé le générateur du code. Ainsi, les codes CDO sont définis comme suit [24] :

Définition 1 *Un encodeur convolutionnel systématique de taux de codage $R = 1/2$ est dit doublement orthogonal (CDO) si son générateur $A = \{\alpha_j : j = 1, \cdots, J\}$, satisfait aux conditions suivantes :*

1. *Les différences simples,* $(\alpha_j - \alpha_k) : j \neq k$, *sont distinctes.*

2. *Les différences doubles,* $(\alpha_j - \alpha_k) - (\alpha_n - \alpha_m) : j \neq k, j \neq n, m \neq k, m \neq n$, *sont distinctes à l'exception des répétitions inévitables provenant des permutations des indices* j *avec* m *et* k *avec* n.

3. *Les différences doubles sont distinctes des différences simples.*

2.2.5.2 Décodage à seuil itératif des codes CDO

À la réception, les codes CDO sont décodés par un décodeur à seuil itératif (DSI) [24] qui est composé d'un ensemble de N itérations élémentaires où chaque itération est un décodeur à seuil tel qu'illustré à la figure 2.5. L'algorithme du décodage à seuil itératif est décrit par les deux équations suivantes [24] :

À la première itération :

$$\lambda_i^{(1)} = y_i^u + \sum_{j=1}^{J} \psi_{i,j}^{(1)} = y_i^u + \sum_{j=1}^{J} \left(y_{(i+\alpha_j)}^p \diamond \sum_{k=1}^{j-1}{}^{\diamond} y_{(i+\alpha_j-\alpha_k)}^u \diamond \sum_{k=j+1}^{J}{}^{\diamond} \lambda_{(i+\alpha_j-\alpha_k)}^{(\mu)} \right) \quad (2.11)$$

Et à chaque itération μ, $(1 < \mu \leq N)$:

$$\lambda_i^{(\mu)} = y_i^u + \sum_{j=1}^{J} \psi_{i,j}^{(\mu)} = y_i^u + \sum_{j=1}^{J} \left(y_{(i+\alpha_j)}^p \diamond \sum_{k=1}^{j-1}{}^{\diamond} \lambda_{(i+\alpha_j-\alpha_k)}^{(\mu-1)} \diamond \sum_{k=j+1}^{J}{}^{\diamond} \lambda_{(i+\alpha_j-\alpha_k)}^{(\mu)} \right) \quad (2.12)$$

où $\psi_{i,j}$ représente les équations de parité :

$$\psi_{i,j}^{(\mu)} = y_{(i+\alpha_j)}^p \diamond \sum_{k=1}^{j-1}{}^{\diamond} \lambda_{(i+\alpha_j-\alpha_k)}^{(\mu-1)} \diamond \sum_{k=j+1}^{J}{}^{\diamond} \lambda_{(i+\alpha_j-\alpha_k)}^{(\mu)} \quad (2.13)$$

L'opérateur "$\diamond$" est appelé opérateur Addmin qui représente l'approximation du rapport

de vraisemblance de l'addition modulo 2 de variables aléatoires binaires. La fonction Addmin est définie selon l'équation suivante [24] :

$$\sum_{i=1}^{n}{}^{\diamond}(B_i) = \left((-1)^{n+1}\prod_{i=1}^{n} sign(B_i)\right) \min_{1\leq i\leq n}\{|B_i|\} \tag{2.14}$$

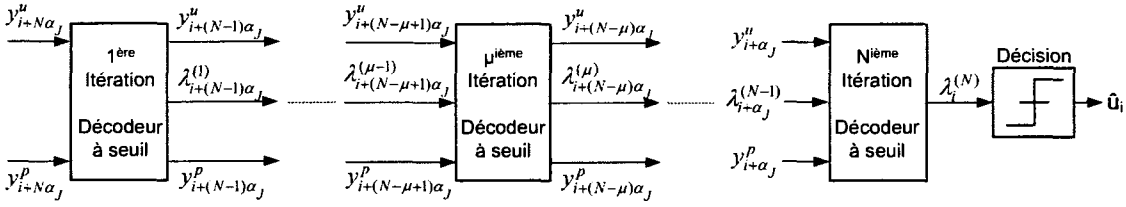

Figure 2.5: Schéma bloc du décodeur à seuil itératif

Selon l'algorithme du décodage à seuil itératif, l'itération μ, ($\mu > 1$), utilise pour ses entrées les symboles d'information et ceux de parités, y^u et y^p respectivement, associés à $v^{(0)} = u$ et $v^{(1)} = p$ du codeur ainsi que la sortie de l'itération $(\mu - 1)$, $\lambda^{(\mu-1)}$. À la première itération où $\mu = 1$, $\lambda^{(\mu-1)}$ est remplacé par y^u. La valeur $\lambda^{(\mu)}$ obtenue à la sortie de l'itération μ est utilisée à l'itération $(\mu+1)$ comme une estimation améliorée du bit à décoder. De plus, y^u et y^p sont fournis à l'itération suivante $(\mu+1)$ si cette dernière a lieu ($\mu < N$). Dans ce cas, les valeurs de y^u et y^p sont préservées pendant un laps de temps qui correspond à la mémoire du codeur, α_J.

À la dernière itération du DSI, la règle de décision est donnée par :

$$\hat{u}_i = \begin{cases} 1 & si\ \lambda_i^{(N)} \geq 0 \\ 0 & si\ \lambda_i^{(N)} < 0 \end{cases} \tag{2.15}$$

Le gain de codage asymptotique, G_c, qui représente la limite des performances d'erreur d'un DSI est donné par [24] :

$$G_c = 10log_{10}(R(J+1)) \quad (dB) \tag{2.16}$$

où J est le nombre minimal d'équations de parité utilisées à chaque itération du DSI pour décoder un bit d'information.

L'auteur de [24] a démontré que la pondération à la sortie de chacune des itérations améliore les performances d'erreurs du DSI. La pondération consiste à multiplier la sortie de chaque itération par un coefficient de pondération, CP, $0 < CP \leq 1$. Dans ce mémoire, nous considérons que la valeur de CP est identique pour toutes les itérations du DSI. Cependant, la recherche d'un vecteur de coefficients de pondération non uniformes qui améliore les performances d'erreurs du DSI a été effectuée par [24] ainsi que par les auteurs de [27]. L'architecture du décodeur à seuil proposée par [24] est montrée à la figure 2.6.

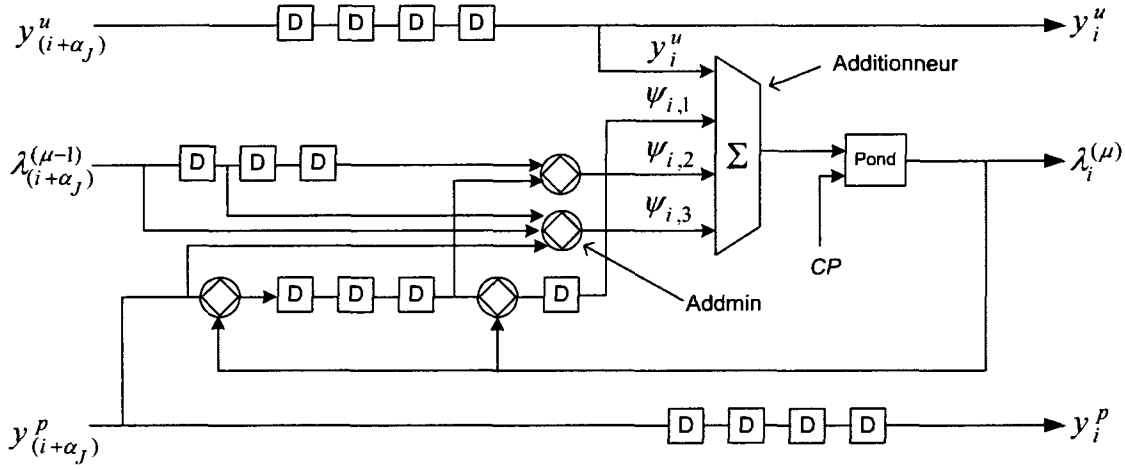

Figure 2.6: Architecture du décodeur à seuil correspondant au générateur $A = \{0, 1, 4\}$

2.2.5.3 Simplification des codes CDO

La figure 2.6 montre que la complexité du décodeur à seuil dépend du Span α_J du codeur. Afin de réduire α_J d'un codeur CDO ayant J connexions, les auteurs de [28–30]

ont démontré que la deuxième condition de la définition 1 peut être relaxée sans trop dégrader les performances d'erreur du DSI. Ils ont défini les codes CDO simplifiés en permettant à un certain nombre de différences doubles d'être égales. Ainsi, le facteur de simplification, δ, des codes CDO a été introduit comme étant le ratio entre le nombre de différences doubles égales et le nombre total de différences doubles en excluant les répétitions inévitables.

Cependant, de nouveaux codes CDO à multi-registres à décalage (M-CDO) ont été introduits et sont présentés dans ce mémoire au chapitre 4. Les générateurs des codes M-CDO incluent un nombre moindre de répétitions inévitables que les générateurs des codes CDO. De plus, certains générateurs M-CDO proposés n'incluent aucune répétition inévitable. Afin de comparer de façon judicieuse les différents types de codes CDO, nous allons redéfinir le facteur de simplification des codes CDO en incluant les répétitions inévitables.

En considérant un générateur des codes CDO, $A = \{\alpha_j, 1 \leq j \leq J\}$, soient N_d le nombre total de différences doubles incluant les répétitions inévitables et N_d^e le nombre de différences doubles égales incluant les répétitions inévitables. Le facteur de simplification, δ, des codes CDO est défini par :

$$\delta = \frac{N_d^e}{N_d} \tag{2.17}$$

En incluant les répétitions inévitables, δ est toujours > 0. Ainsi, dans ce mémoire, tous les codes CDO connus sont considérés comme des codes simplifiés. Le tableau 2.1 montre des exemples de générateurs des codes CDO pour $J = 10$ tirés de [30]. On peut remarquer l'effet de la simplification sur le Span du codeur.

Tableau 2.1: Exemples de générateurs des codes CDO, $J = 10$, $R = 1/2$

Ensemble des connexions $\{\alpha_j\}$	Span (α_J)	δ
$\{0, 29, 40, 43, 1020, 1328, 1495, 1606, 1696, 1698\}$ (non relaxé)	1698	0.685
$\{0, 128, 261, 410, 534, 698, 743, 891, 1038, 1190\}$	1190	0.728
$\{0, 1, 87, 93, 226, 262, 296, 316, 327, 340\}$	340	0.840

2.2.5.4 Perforation des codes CDO

La perforation est une technique simple qui consiste à éliminer de façon périodique certains symboles du mot de code à la sortie du codeur. Ce mécanisme permet d'augmenter le taux de codage ainsi que l'efficacité spectrale des communications. Cependant, la perforation engendre une réduction des performances d'erreur. La perforation des codes CDO a été introduite par Haccoun *et al.* [31] afin de définir les codes CDO perforés (PCDO). Ensuite, les codes PCDO ont été simplifiés par Roy *et al.* [30, 32].

Selon [31] et [32], les codes PCDO sont générés par élimination périodique de certains symboles de parité à la sortie d'un codeur convolutionnels systématiques (SCC) de taux de codage "mère" $R = 1/2$. Ainsi, le taux de codage "perforé" devient $R = b/(b+1) > 1/2$. L'élimination des symboles de parité s'effectue selon un patron de perforation qui est caractérisé par une matrice de perforation $\mathcal{P}$ qui comporte 2 rangées et b colonnes, tel qu'illustré en (2.18).

$$\mathcal{P} = \begin{pmatrix} \underbrace{1 \quad \cdots \quad 1 \quad 1 \quad 1 \quad \cdots \quad 1}_{b \text{ colonnes}} \\ 0 \quad \cdots \quad 0 \quad 1 \quad 0 \quad \cdots \quad 0 \end{pmatrix} \tag{2.18}$$

La première ligne de $\mathcal{P}$ qui ne comporte que des 1 est associée aux symboles d'information transmis par le codeur systématique. Cependant, la deuxième ligne de $\mathcal{P}$ ne contient qu'un seul 1 qui indique le seul symbole de parité transmis avec les b symboles

d'information vers le canal. Les $(b-1)$ autres symboles de parité sont éliminés. Par convention, la position de la valeur 1 sur la deuxième ligne de $\mathcal{P}$ est notée π, $0 \leq \pi \leq b-1$.

Au niveau du décodage, seuls les symboles de parité reçus du canal peuvent intervenir. Ainsi, des zéros doivent être insérés à la place des symboles de parité perforés [31]. En vérifiant les équations (2.11) et (2.12), le nombre d'équations de parité $\psi_{i,j}^{(\mu)}$ disponibles pour décoder le symbole u_i à l'instant i à l'itération μ, $1 \leq \mu \leq N$, est défini par la cardinalité du sous ensemble suivant [31,32] :

$$A_h = \{\alpha_j : \alpha_j = (\pi - i) = h \; mod(b)\}, \quad j = 1, 2, \cdots, J \tag{2.19}$$

où la cardinalité de cet ensemble est définie par $|A_h|$ avec $h = 0, 1, \cdots, b-1$. Si les cardinalités de tous les sous ensembles A_h sont égales :

$$|A_0| = |A_1| = \cdots = |A_{b-1}| = J/b \tag{2.20}$$

alors les codes perforés générés offrent une protection égale (EEP, en anglais Equal Error Protection) sur les symboles décodés pour le taux de codage perforé $R = b/(b+1)$. Dans le cas contraire, les codes perforés offrent une protection inégale (UEP, en anglais Unequal Error Protection) sur les symboles décodés [31,32]. Il a été démontré dans [31,32] que les performances d'erreur des codes EEP sont supérieures à celles des codes UEP. Par conséquent, seuls les codes PCDO de type EEP sont considérés dans ce mémoire.

2.2.5.5 Facteur de simplification associé aux codes PCDO

La définition des codes PCDO est donnée par [31] :

Définition 2 *Un encodeur convolutionnel systématique de taux de codage $R = 1/2$ est dit PCDO si son générateur $A = \{\alpha_j : j = 1, \cdots, J\}$, satisfait pour chaque sous ensemble A_h aux conditions suivantes :*

1. *Les différences simples, $(\alpha_j^{(h)} - \alpha_k) : j \neq k, \alpha_j \in A_h, \alpha_k \in A$, sont distinctes.*
2. *Les différences doubles, $(\alpha_j^{(h)} - \alpha_k^{(h')}) - (\alpha_n - \alpha_m^{(h')}) : j \neq k, j \neq n, m \neq k, m \neq n, h' = 0, \cdots, b-1, \alpha_j \in A_h, \alpha_k, \alpha_m \in A_{h'}, \alpha_n \in A$, sont distinctes à l'exception des répétitions inévitables provenant des permutations des indices j avec m et k avec n.*
3. *Les différences doubles sont distinctes des différences simples.*

La relaxation de la deuxième condition de la définition 2 consiste à permettre à certaines différences doubles d'être égales [32]. Cependant, tenant compte des répétitions inévitables, tous les codes PCDO sont considérés comme des codes simplifiés dans ce mémoire. Ainsi, le facteur de simplification associé aux codes PCDO introduit dans [32] est redéfini pour un sous ensemble A_h par le ratio entre le nombre de différences doubles égales $N_d^{e(h)}$ et le nombre total de différences doubles $N_d^{(h)}$, toujours en incluant les différences doubles inévitables :

$$\delta_h = \frac{N_d^{e(h)}}{N_d^{(h)}} \tag{2.21}$$

Par conséquent, les codes PCDO sont représentés par b facteurs de simplification δ_h, $h = 0, 1, \cdots, b-1$. Toutefois, les codes PCDO sont caractérisés, par convention, à l'aide du facteur de simplification maximal [32] :

$$\delta_{max} = \max_{h \in \{0, \cdots, b-1\}} \delta_h \tag{2.22}$$

2.3 Notions matérielles

2.3.1 La technologie FPGA

Un FPGA (en anglais Field-Programmable Gate Array) est un composant électronique programmable qui peut être reconfiguré (reprogrammé) par l'utilisateur après la fabrication. Le FPGA met à la disposition de l'usager des ressources matérielles où la connectivité des différents éléments est configurable. Grâce à l'évolution de la technologie de semi-conducteurs et à la réduction ininterrompue d'échelle des transistors, les FPGA disponibles dans le marché aujourd'hui comportent un vaste choix de fonctionnalités sophistiquées telles que : des blocs de mémoire de plusieurs méga-octets, des modules précis de gestion d'horloge, des entrées/sorties de hautes performances ainsi que des processeurs embarqués dans certains FPGA.

La simplicité du flot de conception figure parmi les principales particularités des FPGA. Contrairement au ASIC (en anglais Application-Specific Integrated Circuit) où un dessin de masque doit être réalisé, un FPGA possédant une implémentation physique de toutes ses ressources et un réseau configurable d'interconnexions préexistant constitue une alternative qui réduit les risques financiers et le temps de mise en marché d'un nouveau produit. De plus, il est également possible de reconfigurer les FPGA une fois le produit rendu chez le client, ce qui réduit les coûts du support à la clientèle après vente d'un produit. Néanmoins, le coût par unité d'un FPGA est plus élevé que celui d'un ASIC qui offre un faible coût de production bien que son coût d'ingénierie soit important. Ainsi, pour une production en masse la solution FPGA est beaucoup plus dispendieuse. De plus, pour la même technologie, une solution ASIC peut être mieux optimisée qu'une solution FPGA des points de vue de la complexité engendrée, de la consommation de la puissance ainsi que de la fréquence maximale d'opération. Cependant, dans le cadre de ce projet nous considérons une solution FPGA afin de réduire le temps de conception et

de livraison des prototypes du DSI à hauts débits.

2.3.2 Les FPGA de la famille Virtex-II Pro de Xilinx

Dans ce mémoire, les prototypes du DSI sont conçus en visant le FPGA Virtex-II Pro "XC2VP70-7" de la compagnie Xilinx [33]. La famille Virtex-II Pro est fabriquée en utilisant la technologie 0.13 μm CMOS à 9 niveaux de cuivre. L'architecture générique des FPGA de la famille Virtex-II Pro est montrée à la figure 2.7.

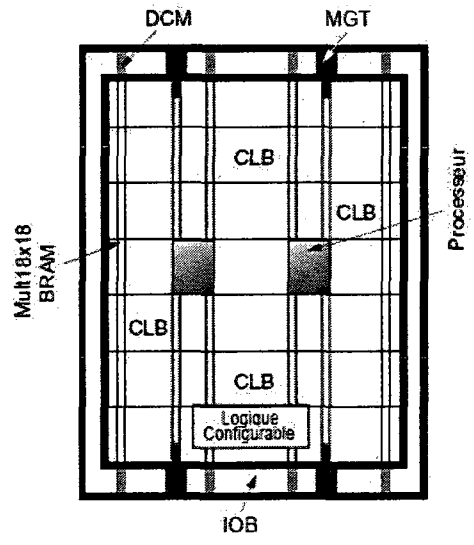

Figure 2.7: Architecture générique des FPGA de la famille Virtex-II Pro

La figure 2.7 montre la disposition des ressources matérielles de la famille Virtex-II Pro. Ainsi, on distingue les blocs de logique configurables (CLB, Configurable Logic Block), les blocs de mémoire RAM (BRAM) et les multiplieurs câblés (Mult18x18), les modules de gestion d'horloge (DCM, Digital Clock Manager), les processeurs, les cellules d'entrée/sortie (IOB, Input/Output Block) et les blocs d'entrée/sortie à multi-Gigabit (MGT, Multi-Gigabit Transceiver).

Les CLB constituent la partie où la logique combinatoire d'un design est implémentée.

Un CLB est formé par quatre "Slices" où chaque "Slice" comporte deux structures similaires appelées cellule logique (LC, en anglais Logic Cell). À son tour, un LC est constitué d'un LUT (Look-Up Table), d'une bascule (FF, Flip-Flop) et d'une chaîne de retenue. Il importe de noter qu'un LUT est une mémoire 16×1 bits qui, en la programmant avec le contenu approprié, implémente la fonction combinatoire désirée. De plus, un LUT offre trois modes d'utilisation. En effet, un LUT peut être configuré pour implémenter soit une fonction combinatoire à quatre entrées ($log_2(16)$), soit une mémoire RAM de capacité 16×1 bits (RAM16) ou un registre à décalage ayant une longueur configurable située entre 1 et 16 bits et une largeur de 1 bit. Dans ce dernier cas, le registre à décalage résultant est appelé SRL16 (Shift Registre LUT). Il s'ensuit qu'un CLB a une capacité maximale de 128 bits (RAM ou registre à décalage) qui correspond à la combinaison de ses 8 LUT.

Les BRAM sont des mémoires de 18 kbits arrangées en colonnes et associées chacune à un multiplieur Mult18x18. Les modules DCM sont spécialisés dans la synchronisation d'horloge. Ils sont distribués avec les IOB et les MGT au pourtour du FPGA proche de ses plots. D'autre part, les processeurs sont des modules PowerPC embarqués et ils ne sont offerts que dans certains modèles de la famille Virtex-II Pro seulement. Le tableau 2.2 montre la disposition des ressources matérielles du FPGA considéré dans ce mémoire, le Virtex-II Pro XC2VP70-7 [33].

Tableau 2.2: Ressources matérielles du FPGA Virtex-II Pro XC2VP70-7

LC	Mult18x18	BRAM (18 Kbits)	DCM	Processeur PowerPC	MGT
74448	308	308	8	2	16 ou 20

Nous constatons qu'il existe trois types de mémoires dans un FPGA de la famille Virtex-II Pro. Soit les LUT (en mode RAM16 ou SRL16), les FF et les BRAM. Dans le cadre de ce projet, seulement les éléments de mémoire de type LUT et FF sont utilisés afin d'implémenter les prototypes du DSI à haut débit.

2.3.3 Délai critique d'un circuit numérique

Un circuit numérique combinatoire (sans éléments de mémoire) est un ensemble de plusieurs portes logiques combinatoires interconnectées entre elles. Désignons par "commutation" le changement de valeur d'un signal binaire. La rapidité d'un circuit combinatoire est alors mesurée par le temps qui s'écoule entre des commutations en entrée et les commutations qui en résultent en sortie. Ce temps de commutation est appelé le délai de propagation d'un circuit combinatoire (t_{comb}). En général, le chemin ayant le délai de propagation le plus élevé dans un circuit numérique est appelé le chemin critique.

D'autre part, dans un circuit numérique séquentiel (ayant des éléments de mémoire), les chemins sont mesurés entre les éléments de mémoire. En outre, une commutation s'amorce avec un front actif du signal d'horloge (généralement le front montant). Ainsi, trois autres délais s'ajoutent au délai combinatoire t_{comb} d'un chemin donné qui existe entre deux éléments de mémoire comme montré à la figure 2.8. Soit le délai de transition de l'entrée vers la sortie d'un élément de mémoire à partir d'un front actif d'horloge (t_{co}, Clock to Out time), le délai de "setup" à l'entrée d'un élément de mémoire (t_{su}) et les délais de routage (t_r). t_{su} représente le temps nécessaire durant lequel l'entrée d'un élément de mémoire doit être stable avant un front actif d'horloge pour que l'entrée de l'élément soit valide. Considérons le temps qui s'écoule entre deux fronts actifs et qui correspond à une période d'horloge T. Afin d'assurer le bon fonctionnement d'un circuit numérique, les données présentes à l'entrée du premier élément de mémoire doivent transiter à sa sortie (t_{co}) puis se propager dans le circuit de logique combinatoire (t_{comb} + t_r) afin que les commutations soient valides avant la valeur t_{su} du prochain front actif d'horloge. Considérant tous les délais de propagation dans le circuit, on a :

$$T \geq \max(t_{co} + t_{comb} + t_r + t_{su}) = T_{min} \tag{2.23}$$

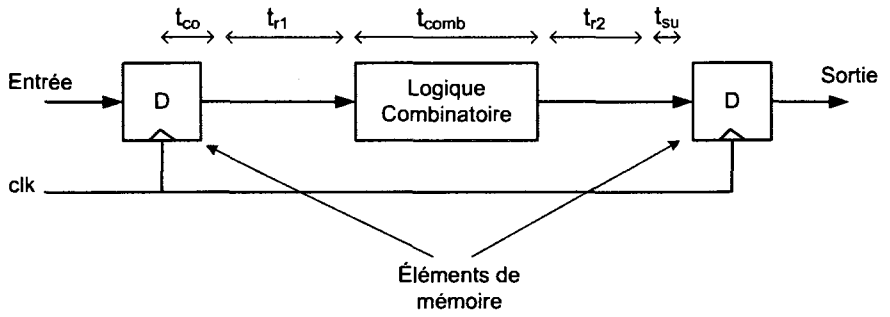

Figure 2.8: Délais de propagation dans un système numérique

où T_{min} est le délai du chemin critique qui correspond à la période minimale du circuit. La fréquence d'opération, f, est alors donné par :

$$f = \frac{1}{T} \leq \frac{1}{T_{min}} = f_{max} \tag{2.24}$$

où f_{max} est la fréquence d'opération maximale du circuit numérique.

2.3.4 Stratégie de resynchronisation d'un circuit numérique

La stratégie de resynchronisation (Retiming) [34] permet de diviser le délai critique d'un circuit numérique en déplaçant les bascules présentes au sein du circuit sans influencer sa fonctionnalité. Le déplacement de registres se fait à travers un opérateur de logique combinatoire. Pour déplacer les registres présents aux entrées d'un opérateur vers ses sorties, la resynchronisation consiste à retirer un registre de chaque entrée afin d'en ajouter un à chaque sortie. Évidemment, pour déplacer les registres présents aux sorties d'un opérateur vers ses entrées, il suffit de retirer un registre de chaque sortie et d'en ajouter un à chaque entrée. Un exemple de resynchronisation d'un circuit numérique est montré à la figure 2.9.

À la figure 2.9, nous supposons que le chemin critique (T_{min}) du circuit numérique est

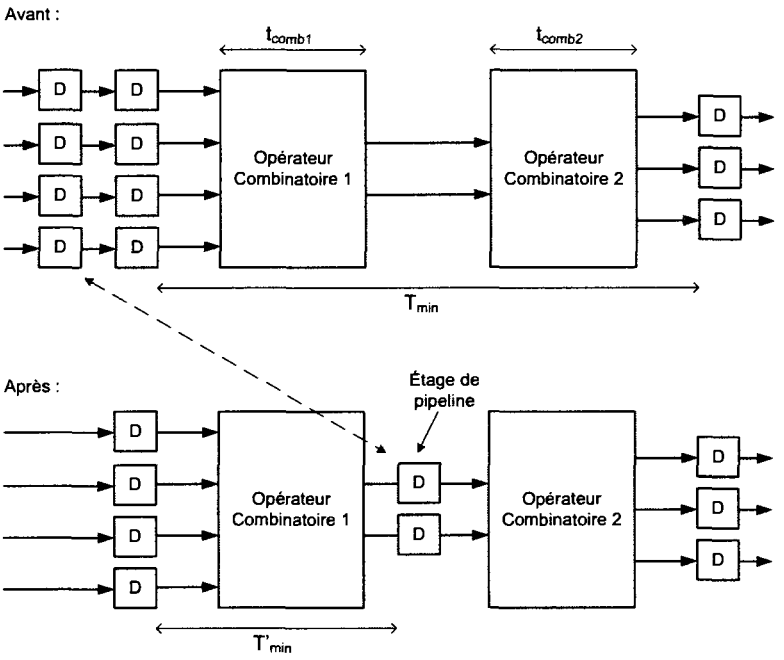

Figure 2.9: Exemple de resynchronisation d'un circuit numérique

composé de deux opérateurs combinatoires. La resynchronisation du premier opérateur a permis de déplacer les registres supplémentaires de ses entrées vers les sorties. Par conséquent, un étage de pipeline est inséré à l'intérieur du chemin critique en le découpant en deux parties. Le nouveau chemin critique T'_{min} du circuit correspond au chemin ayant le délai le plus élevé. En résumé, la stratégie de resynchronisation peut être utilisée pour augmenter la fréquence d'opération maximale d'un circuit numérique. De plus, la resynchronisation s'avère très importante dans le cas d'un circuit comportant une boucle de rétroaction où l'insertion des étages de pipeline à l'intérieur du chemin critique est souvent problématique.

2.3.5 Environnement d'évaluation des performances d'erreur du DSI

L'environnement d'évaluation des performances d'erreur du DSI des codes CDO a été développé par les auteurs de [35] et [36]. Le schéma bloc de l'environnement qui émule

le fonctionnement d'un système de communication réel est illustré à la figure 2.10.

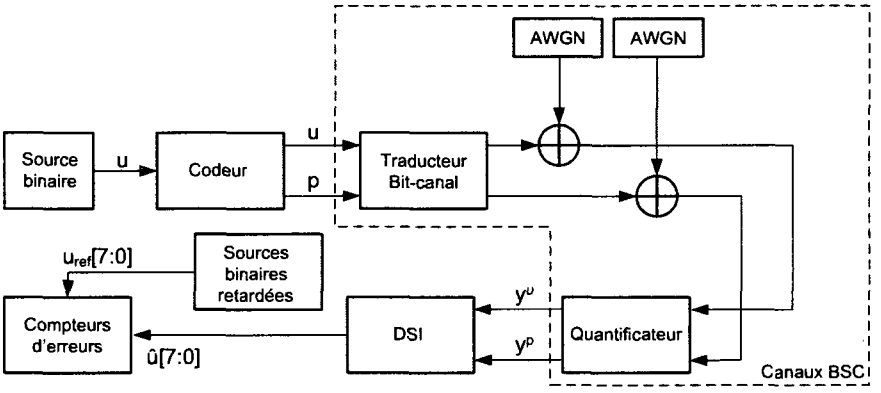

Figure 2.10: Schéma bloc de l'environnement d'évaluation des performances d'erreur

La source binaire utilisée dans l'environnement d'évaluation des performances d'erreur est une source pseudo-aléatoire. Par conséquent, les séquences initiales émises par la source peuvent être régénérées en ajoutant un certain retard qui correspond au nombre de cycles parcourus par un bit pour traverser tous les modules avant qu'il soit décodé à une itération donnée. Ainsi, en comparant les séquences retardées (u_{ref}) avec les séquences décodées ($\hat{u}$) le nombre d'erreurs de transmission peut être comptabilisé pour un nombre de bits émis donné. Pour chacune des itérations du DSI, une source binaire retardée et un compteur d'erreur sont associés.

Deux canaux de communication BSC similaires ont été intégrés dans l'environnement. Un canal pour les symboles d'information (u) et un autre pour les symboles de parité (p). Le traducteur bit-canal à l'entrée des canaux BSC émule d'une façon numérique le fonctionnement d'un modulateur BPSK analogique en bande de base. Il transforme les symboles d'information et de parité binaires présents à ses entrées en des symboles de canal représentés sur 16 bits. Après l'addition du bruit AWGN, le quantificateur fait la tâche inverse en quantifiant les symboles de canal bruités reçus sur 3 bits seulement (quantification douce). Dans le cadre de ce projet, l'environnement d'évaluation des performances d'erreur a été adapté à un seul canal de communication pour tous les symboles transmis.

L'environnement d'évaluation des performances d'erreur a été prototypé en utilisant une plateforme ARM Integrator [37] munie d'un processeur ARM7 et un FPGA Virtex-II de Xilinx. Un bus AMBA (Advanced Microcontroller Bus Architecture) d'ARM a relié le processeur ARM7 au FPGA où l'environnement a été implémenté. Ainsi, un programme de contrôle a été démarré sur le processeur ARM7 afin de lancer les évaluations et ensuite de récupérer les résultats expérimentaux.

L'auteur de [36] a également développé une version matérielle (VHDL) du DSI des codes CDO et a proposé une technique de pipelinage du DSI. Cependant, la version développée du DSI dans [36] a été optimisée du point de vue de la complexité de design et non pas du point de vue du débit. De plus, la technique de pipelinage proposée a été limitée au niveau des opérateurs Addmin et n'a pas touché la plus grande partie du chemin critique du décodeur comme il le sera montré au chapitre suivant. Ajoutons aussi que l'auteur de [36] n'a jamais implémenté une version pipelinée du DSI.

2.4 Conclusion

Dans ce chapitre, les notions préliminaires en matière de télécommunications numériques ainsi que les notions matérielles ont été abordées. Une brève description des codes LDPC et Turbo a été présentée. Cependant, une description quelque peu plus détaillée et une attention particulière ont été portées aux codes CDO. Dans les chapitres qui suivent, l'architecture du DSI à haut débit des codes CDO est abordée.

CHAPITRE 3

DÉCODEUR À SEUIL ITÉRATIF À HAUT DÉBIT DES CODES CDO

3.1 Introduction

Après avoir présenté l'algorithme du décodage à seuil itératif ainsi que l'architecture du Décodeur à Seuil Itératif (DSI) développés dans des travaux précédents, nous présentons dans ce chapitre l'architecture du DSI à haut débit que nous avons implémenté. Pour augmenter le débit d'information à la sortie du DSI, nous utilisons une technique de pipelinage basée sur le principe de resynchronisation des circuits numériques. De plus, afin de supporter les codes perforés ayant des taux de codages $R > 1/2$, les modules de perforation sont développés et intégrés dans le codeur et le DSI. Le DSI obtenu supporte les codes à taux compatibles qui peuvent, comme leur nom l'indique, fonctionner à plusieurs taux de codages. Ainsi, une implémentation VHDL générique du DSI est développée. Finalement, un ensemble de nouveaux générateurs des codes à taux compatibles et à large capacité de pipelinage est présenté à la fin de ce chapitre.

3.2 Architecture du décodeur à seuil

En se basant sur les travaux de Cardinal *et al.* [24] et Provost *et al.* [36], l'architecture du décodeur à seuil développée pour le codeur spécifié par l'ensemble de $J = 3$ connexions $A = \{0, 1, 4\}$ est illustrée à la figure 3.1, et cela avant d'appliquer la technique de pipelinage.

La structure du décodeur à seuil montrée à la figure 3.1 se divise en deux parties princi-

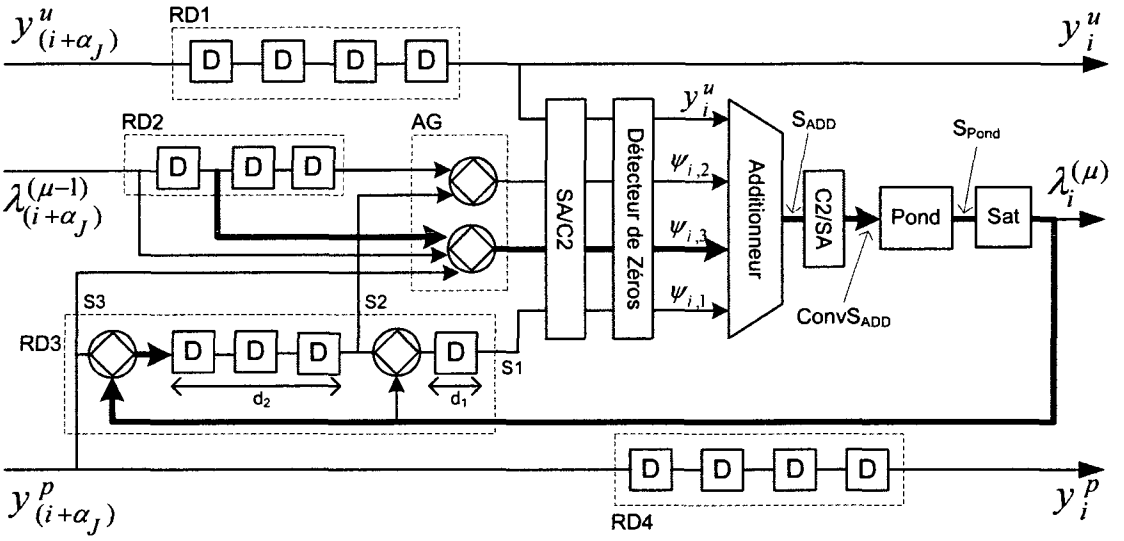

Figure 3.1: Architecture du décodeur à seuil des codes CDO générés par le codeur $A = \{0, 1, 4\}$

pales, soient les registres à décalages et le noyau de logique combinatoire. Les re-gistres à décalages ont généralement pour rôle de fournir au noyau les valeurs nécessaires au calcul de la sortie de l'itération $\lambda^{(\mu)}$. On distingue quatre registres à décalages, soit le registre à décalage (RD1) contenant les symboles d'information, le registre à décalage (RD2) contenant les valeurs de $\lambda^{(\mu-1)}$ provenant de l'itération précédente, le registre à décalage de rétroaction (RD3) et finalement le registre à décalage (RD4) contenant les symboles de parité. Le noyau de logique combinatoire de l'itération contient les différents opérateurs logiques et arithmétiques utilisés dans le calcul de $\lambda^{(\mu)}$. Il est formé d'un opérateur Addmin global (AG), un additionneur, un pondérateur (Pond) qui implémente le coefficient de pondération, un saturateur (Sat), deux opérateurs de conversion binaire (C2/SA) et un détecteur de zéros.

Le décodage à seuil est un processus de décodage symbole par symbole. En d'autres termes, à chaque coup d'horloge un bit est décodé à la sortie du décodeur à seuil. Dans ce cas, le débit maximal d'information que peut atteindre le décodeur à seuil est égal à sa fréquence maximale d'opération. D'autre part, la fréquence maximale d'opération

du décodeur est limitée par le délai total du chemin critique du design. Ce chemin critique, montré par une ligne en gras à la figure 3.1, est formé par l'accumulation de délais combinatoires des différents opérateurs arithmétiques et logiques qui forment le noyau de logique combinatoire du décodeur, par les délais de routage et par les délais de transition à la sortie des registres à décalages. Dans les sous-sections suivantes, les améliorations des composants du décodeur qui réduisent le délai du chemin critique sont présentées[1].

3.2.1 Le pondérateur

La pondération consiste à multiplier la sortie de l'additionneur (S_{ADD}) par un coefficient de pondération (CP) situé dans $]0, 1]$ [24]. À l'intérieur du FPGA, nous disposons de plusieurs multiplieurs qui peuvent être utilisés pour implémenter le pondérateur. Chaque multiplieur est un primitif "MULT18x18" capable de multiplier deux opéran-des représentés en format binaire "Complément à deux" (C2) et ayant chacun une résolution binaire pouvant aller jusqu'à 18 bits. Un multiplieur est donc nécessaire pour implémenter un pondérateur à chaque itération du DSI.

Une architecture du pondérateur qui effectue les multiplications en C2 a été proposée dans [36]. Or, dans le but de réduire le délai combinatoire introduit par les multiplieurs, le fournisseur de FPGA, Xilinx, suggère de brancher les opérandes sur les bits des entrées de poids faible (LSB, Low Significant Bits) [33]. Il y a donc intérêt à utiliser seulement les LSB d'un MULT18x18. Toutefois, une multiplication en C2 exige d'utiliser les bits de poids fort (MSB, Most Significant Bits) qui représentent les signes des opérandes dans ce cas. Il s'en suit alors qu'un pondérateur qui effectue des multiplications en C2 introduit un grand délai au chemin critique du décodeur.

[1] À l'annexe I, les descriptions des autres composants du décodeur à seuil sont présentées

Afin de réduire le délai engendré par le pondérateur, il est proposé dans ce travail d'effectuer des multiplications non signées où les opérandes sont toujours positifs [38]. Dans ce cas, les opérandes peuvent être branchés sur les LSB du MULT18x18 en remettant tous les MSB à zéro. La représentation binaire "Signe-Amplitude" (SA) qui fournit directement la valeur absolue d'un nombre signé est alors plus appropriée, c'est pourquoi un opérateur de conversion binaire (C2/SA) a été placé directement à la sortie de l'additionneur (S_{ADD}).Cet opérateur transforme S_{ADD} de format C2 en format SA, générant ainsi une sortie appelée $ConvS_{ADD}$.

Le calcul de la sortie du pondérateur, $S_{Pond} = CP \times ConvS_{ADD}$, en SA se fait maintenant en calculant le signe (Signe(S_{Pond})) et l'amplitude (Amp(S_{Pond})). Puisque CP est positif alors : Signe(S_{Pond}) = Signe($ConvS_{ADD}$). De plus, l'opération de calcul Amp(S_{Pond}) = $CP\times$ Amp($ConvS_{ADD}$) s'effectue par le multiplieur MULT18x18 en utilisant ses LSB. La nouvelle architecture du pondérateur exploitant le format binaire SA est montrée à la figure 3.2. Dans cette figure, RS_{ADD} et R_{CP} représentent les résolutions binaires de $ConvS_{ADD}$ et CP respectivement. La résolution binaire de leur produit S_{Pond} est donnée par :

$$RS_{Pond} = RS_{ADD} + R_{CP} \tag{3.1}$$

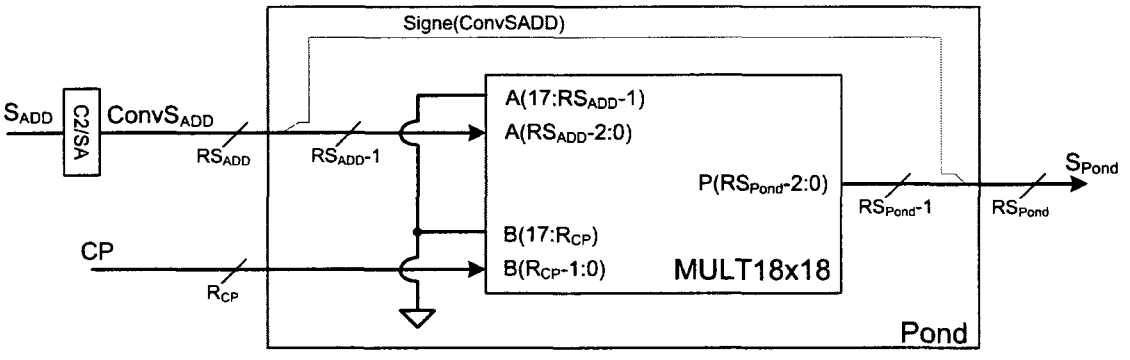

Figure 3.2: Nouvelle architecture du pondérateur exploitant le format binaire SA

3.2.2 Les registres à décalage

Les symboles y^u, y^p, $\lambda^{(\mu-1)}$ et $\lambda^{(\mu)}$ sont observés à l'intérieur du décodeur sur une fenêtre temporelle de longueur α_J. Cependant, la valeur de α_J pour les encodeurs connus des codes CDO croit selon une fonction polynomiale en J^4 comme montré par les résultats de recherche de ces codes [24, 25, 30]. Ainsi, le nombre de bascules D (FF, pour Flip-Flop en anglais) nécessaires pour implémenter les différents registres à décalage d'un décodeur à seuil devient très important à mesure que J augmente. Par exemple, le DSI formé de 8 itérations implémenté pour les codes générés par l'encodeur ayant $J = 11$ et $\alpha_J = 588$ publié dans [30] nécessite environ 75224 FFs. Toutefois, le nombre total de bascules disponibles dans le FPGA considéré dans ce travail est limité à 74448 FFs.

Afin de remédier à ce problème, l'auteur de [39] a proposé deux autres alternatives pour implémenter les registres à décalage du décodeur. La première consiste à utiliser des blocs de mémoire volatile (BRAM) du FPGA en intégrant un mécanisme de conversion des adresses d'entrée et de sortie. Cependant, ce type d'implémentation a été conçu pour les longs registres à décalage c'est-à-dire, ceux ayant une longueur supérieure à 690. Pour les registres à décalage de longueur moyenne ($\leq$ 690), la deuxième alternative proposée par l'auteur de [39] consiste à utiliser les primitifs "SRL16" disponibles sur le FPGA. Dans le cadre de ce projet, étant donné que pour les codes CDO simplifiés considérés la majorité des registres à décalage élémentaires sont de longueurs moyennes, seule l'alternative utilisant les primitifs SRL16 sera explorée.

Se basant sur les résultats de [39], nous présentons une nouvelle architecture d'un registre à décalage élémentaire configurable. Cette nouvelle architecture proposée a permis d'améliorer le débit du décodeur en utilisant d'une façon appropriée les ressources disponibles dans le FPGA considéré. Le registre à décalage élémentaire ainsi obtenu est ensuite utilisé pour implémenter les différents registres à décalage du décodeur.

3.2.2.1 Registre à décalage élémentaire configurable

Le SRL16 est un LUT programmé en mode registre à décalage et qui a une largeur de 1 bit et une longueur configurable variant entre 1 et 16 bits. Une première approche consiste à implémenter un registre à décalage élémentaire configurable de longueur L et de largeur W en utilisant pour chaque bit $\lceil L/16 \rceil$ primitifs SRL16. La figure 3.3 illustre un registre à décalage élémentaire de longueur $L = 22$ et de largeur $W = 3$ implémenté en utilisant les primitifs SRL16. On remarque que la longueur du premier SRL16 utilisé est configurée à 16 et que la longueur du deuxième est configurée à 6.

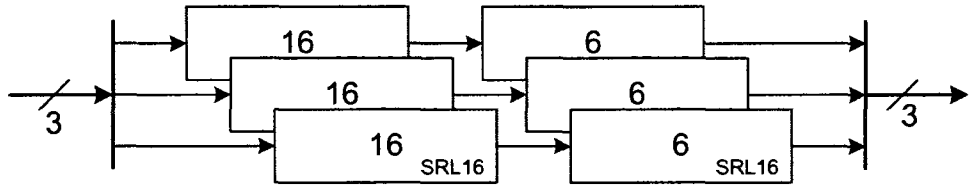

Figure 3.3: Implémentation du registre à décalage élémentaire à partir de SRL16s

Le chemin critique du décodeur s'amorce à la sortie de l'un ou l'autre des registres à décalage élémentaires qui constituent RD2 et RD3 (voir figure 3.1). Notons par t_{coFF} et $t_{coSRL16}$ les temps de transition nécessaires pour obtenir une réponse valide à la sortie d'un FF et un SRL16 respectivement après un front montant d'horloge (en anglais: clock to output time). Pour les registres à décalage élémentaires implémentés à partir des primitifs SRL16, le temps $t_{coSRL16}$ s'ajoute au délai total du chemin critique. Cependant, les résultats de la synthèse pour le FPGA considéré dans le cadre de ce projet ont montré que $t_{coSRL16} = 2.794ns$ tandis que $t_{coFF} = 0.370ns$. Cela veut dire que les registres à décalage élémentaires implémentés à partir de FFs sont plus avantageux du point de vue du délai total du chemin critique. Par ailleurs, les registres à décalage élémentaires implémentés à partir de SRL16 sont plus avantageux du point de vue de complexité du décodeur.

Afin de tenir compte des avantages des SRL16 et FF dans un même design, un FF est

inséré avant la sortie du registre à décalage élémentaire implémenté à partir de SRL16s. Alors, un registre à décalage élémentaire configurable de longueur L et de largeur W est maintenant implémenté en utilisant $\lceil (L-1)/16 \rceil$ primitifs SRL16 et une bascule D par bit. Dans ce cas, le chemin critique du décodeur s'amorce à la sortie de la bascule et un temps t_{coFF} seulement s'ajoute au délai total engendré par le chemin critique. La figure 3.4 illustre la nouvelle architecture du registre à décalage élémentaire de longueur $L = 22$ et de largeur $W = 3$.

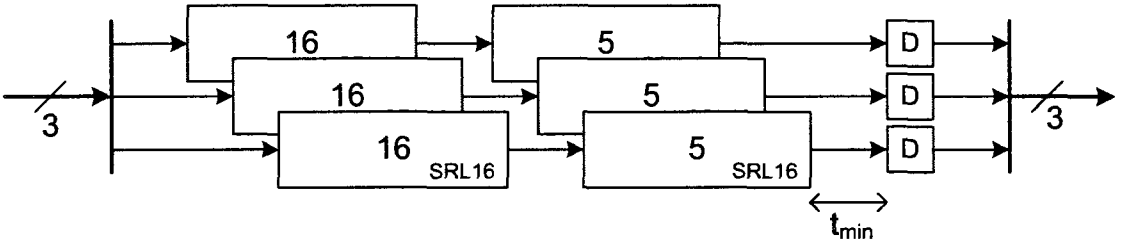

Figure 3.4: Nouvelle architecture du registre à décalage élémentaire

À partir de cette architecture, nous pouvons définir une limite pour la fréquence maximale (f_{max}) que le décodeur peut atteindre. En effet, soit $t_{suFF} = 0.208ns$ le temps de setup à l'entrée de FF et $t_{rSRL16FF}$ le temps de routage entre un primitif SRL16 et un FF, alors, le délai minimal du chemin critique (t_{min}) ne peut pas être plus petit que $t_{coSRL16} + t_{suFF} + t_{rSRL16FF}$ qui représente le délai entre la sortie du primitif SRL16 et l'entrée du FF comme montré à la figure 3.4 :

$$t_{min} = t_{coSRL16} + t_{suFF} + t_{rSRL16FF} \geq t_{coSRL16} + t_{suFF} = 3.002ns \tag{3.2}$$

De sorte que la fréquence maximale devient :

$$f_{max} = 1/t_{min} \leq 1/3.002 \simeq 333MHz \tag{3.3}$$

3.2.2.2 Registre à décalage de rétroaction

Le registre à décalage de rétroaction (RD3) se charge d'effectuer les opérations "Addmin" entre les symboles de parités $y^p_{i+\alpha_j}$ et la rétroaction sur la décision représentée par les valeurs de $\lambda^{(\mu)}_{i+(\alpha_j-\alpha_k)}$, $k > j$. En d'autres termes, RD3 incorpore les opérations Addmin impliquant les différences simples négatives et qui représentent la moitié du nombre total des opérations Addmin nécessaires pour le calcul des J équations de parité $\psi^{(\mu)}_{(i,j)}$ données par l'équation (2.13).

Afin d'expliquer les étapes nécessaires pour déduire l'architecture de RD3 à partir du générateur des codes CDO spécifié par l'ensemble $A = \{\alpha_1, \cdots, \alpha_J\}$, notons par S_j, $1 < j < J$, les J sorties de RD3 où chaque sortie S_j correspond à l'équation de parité $\psi^{(\mu)}_{(i,j)}$. Notons aussi par d_l les $(J-1)$ différences qui existent entre les positions de connexions consécutives du générateur. Les différences d_l sont données par l'équation suivante :

$$d_l = \alpha_{l+1} - \alpha_l, \quad l = 1, 2, \cdots, J-1 \tag{3.4}$$

Notons par, $RInt$, la résolution binaire interne du décodeur qui correspond à la largeur de deux registres à décalage RD2 et RD3. Comme montré à la figure 3.5, RD3 est implémenté de façon à insérer entre les sorties consécutives S_{j+1} et S_j un opérateur Addmin à deux entrées qui combine S_{j+1} et $\lambda^{(\mu)}_i$. Ensuite, un registre à décalage élémentaire de longueur d_j et de largeur $RInt$ est introduit entre la sortie de l'opérateur Addmin et S_j. Enfin, à l'entrée de RD3, S_J est connecté directement à $y^p_{i+\alpha_J}$. Par exemple, pour le générateur $\{0, 1, 4\}$, tel qu'illustré à la figure 3.1, les différences d_1 et d_2 sont $d_1 = 1 - 0 = 1$ et $d_2 = 4 - 1 = 3$.

Dans cette optique, il est possible de représenter S_j en fonction de S_{j+1}. En effet, en

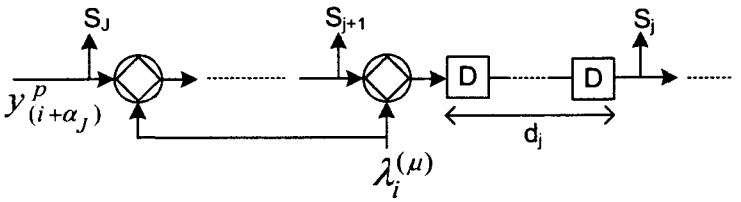

Figure 3.5: Implémentation du registre à décalage de rétroaction

utilisant (3.4) on en déduit la relation suivante :

$$S_j = S_{j+1,-d_j} \diamond \lambda^{(\mu)}_{i-d_j} = S_{j+1,-(\alpha_{j+1}-\alpha_j)} \diamond \lambda^{(\mu)}_{i-(\alpha_{j+1}-\alpha_j)} \tag{3.5}$$

En remplaçant $S_J = y^p_{i+\alpha_J}$ par sa valeur dans (3.5) on obtient :

$$\begin{cases} S_J & = y^p_{i+\alpha_J} \\ S_{J-1} & = y^p_{i+\alpha_J-(\alpha_J-\alpha_{J-1})} \diamond \lambda^{(\mu)}_{i-(\alpha_J-\alpha_{J-1})} \\ & = y^p_{i+\alpha_{J-1}} \diamond \lambda^{(\mu)}_{i+\alpha_{J-1}-\alpha_J} \\ S_{J-2} & = \left(y^p_{i+\alpha_{J-1}-(\alpha_{J-1}-\alpha_{J-2})} \diamond \lambda^{(\mu)}_{i-(\alpha_J-\alpha_{J-1})-(\alpha_{J-1}-\alpha_{J-2})}\right) \diamond \lambda^{(\mu)}_{i-(\alpha_{J-1}-\alpha_{J-2})} \\ & = y^p_{i+\alpha_{J-2}} \diamond \lambda^{(\mu)}_{i+\alpha_{J-2}-\alpha_{J-1}} \diamond \lambda^{(\mu)}_{i+\alpha_{J-2}-\alpha_J} \\ \ldots & \end{cases} \tag{3.6}$$

À partir de (3.6) nous pouvons déduire la valeur de chacune des sorties S_j par l'équation suivante :

$$S_j = y^p_{i+\alpha_j} \diamond \sum_{k=j+1}^{J} \diamond \lambda^{(\mu)}_{(i+\alpha_j-\alpha_k)} \tag{3.7}$$

En utilisant (3.7), l'équation (2.13) peut s'écrire :

$$\psi^{(\mu)}_{i,j} = S_j \diamond \sum_{k=1}^{j-1} \diamond \lambda^{(\mu-1)}_{(i+\alpha_j-\alpha_k)} \tag{3.8}$$

À partir de (3.8), on remarque que la première équation de parité s'obtient directement à la sortie S_1 de RD3, soit $\psi_{i,1}^{(\mu)} = S_1$.

3.3 Technique de pipelinage du décodeur

Le chemin critique du décodeur à seuil passe par le plus grand opérateur Addmin ayant J entrées qui correspond à l'équation $\psi_{(i,J)}^{(\mu)}$ dans AG, puis par l'additionneur, les opérateurs de conversion binaire, le détecteur de zéros, le pondérateur, le saturateur et finalement par un des opérateurs Addmin à 2 entrées dans RD3. Pour augmenter la fréquence d'opération du décodeur et par conséquent, le débit, le décodeur doit être pipeliné et ainsi le chemin critique doit être segmenté en d'autres chemins plus courts et à délais réduits. Toutefois, l'existence de la boucle de rétroaction dans le design limite l'insertion directe des bancs de registres qui forment les étages de pipeline dans le chemin critique. La limitation est causée par la synchronisation nécessaire des différents flux de données.

Prenons par exemple l'opérateur Addmin présent à l'entrée du RD3 montré à la figure 3.1. À chaque instant i, cet opérateur reçoit à ses entrées les valeurs de λ_i et $y_{(i+4)}^p$. Pour que le décodeur puisse fonctionner correctement, le temps écoulé entre λ_i et y_{i+4}^p à l'entrée de l'opérateur en question doit évidemment être fixé à +4. Supposons maintenant qu'un étage de pipeline est inséré entre le pondérateur et le saturateur, à l'entrée de l'opérateur Addmin mentionné ci-haut, se trouve la valeur $\lambda_{(i-1)}$ au lieu de λ_i. Le délai entre $\lambda_{(i-1)}$ et y_{i+4}^p devient égal à +5, ce qui rend les résultats du décodage erronés. En somme, une attention toute particulière à la synchronisation des flux de données doit être tenue en compte lord de l'application des techniques de pipelinage.

3.3.1 Stratégie de resynchronisation du décodeur

La stratégie de resynchronisation cherche à déplacer des registres vers l'intérieur du noyau de logique combinatoire du décodeur. Elle se décrit en trois étapes [38] :

Étape 1 : **Resynchronisation de tous les opérateurs Addmin dans RD3**
Une bascule est déplacée de la sortie de chaque Addmin dans RD3 vers ses entrées. Par conséquent, une bascule libre est obtenue sur la boucle de rétroaction tel que montré à la figure 3.6.

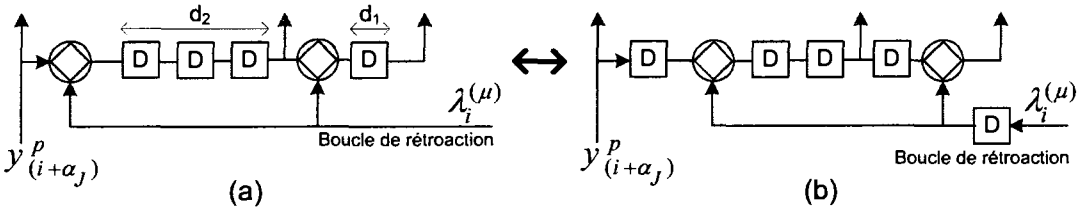

Figure 3.6: Resynchronisation de RD3, Avant (a) et Après (b)

Étape 2 : **Ajout d'un délai de rétention à chacune des entrées du décodeur**
Considérant le bloc Z illustré à la figure 3.7, on constate que Z a une bascule disponible sur chacune de ses entrées y compris la bascule obtenue par *Étape 1*.

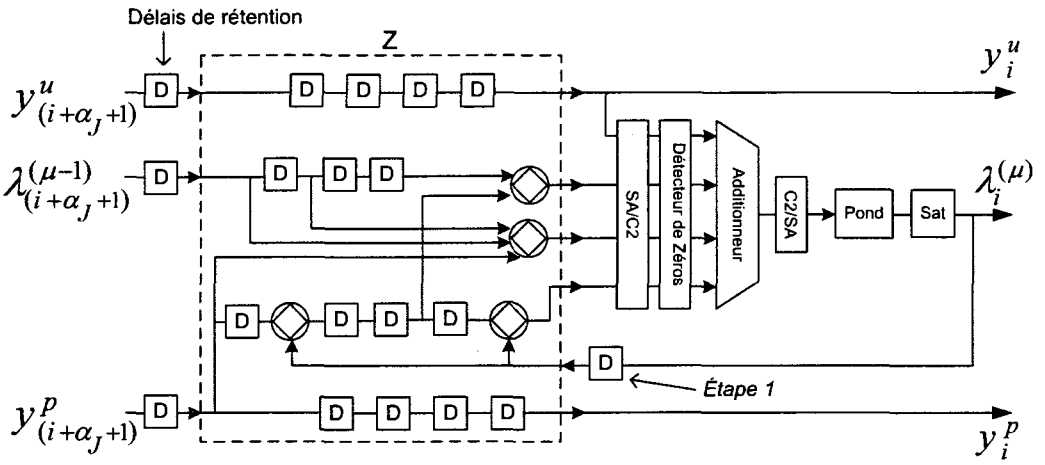

Figure 3.7: *Étape 2*, Bloc Z et ajout des délais de rétention

Étape 3 : **Resynchronisation de Z**

La resynchronisation de Z s'effectue en déplaçant une bascule de chacune de ses entrées vers ses sorties. Étant donné que les sorties de Z sont situées à l'intérieur du noyau de logique combinatoire, les bascules resynchronisées dans cette étape forment un étage de pipeline et le chemin critique est segmenté et réduit à celui montré à la figure 3.8.

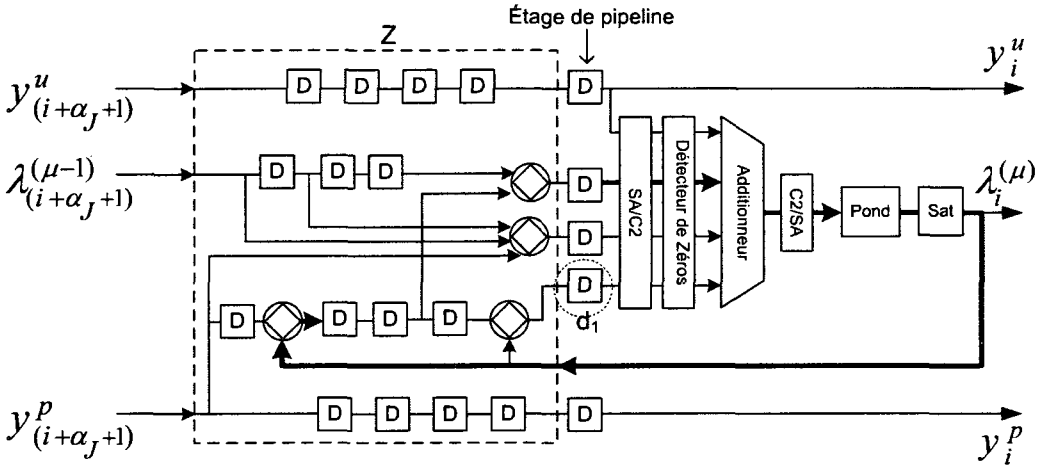

Figure 3.8: *Étape 3*, insertion de l'étage de pipeline

Il résulte de l'application de ces trois étapes, l'insertion d'un étage de pipeline. Pour insérer d'autres étages, ces deux étapes doivent être répétées tant que le nombre de registres disponibles dans RD3 le permet. Ainsi, les étages de pipeline peuvent être placés tout le long du chemin critique.

3.3.2 Emplacements des étages de pipeline

La stratégie de resynchronisation a permis d'insérer les étages de pipeline n'importe où dans le noyau de logique combinatoire du design. Cependant, les emplacements des étages de pipelinage doivent être judicieusement choisis. En fixant $RInt$ à 5 bits, la figure 3.9 illustre les emplacements choisis dans le chemin critique du décodeur des codes CDO $\{0, 1, 4\}$ après l'expansion des arbres de l'additionneur et de plus grand

opérateur Addmin.

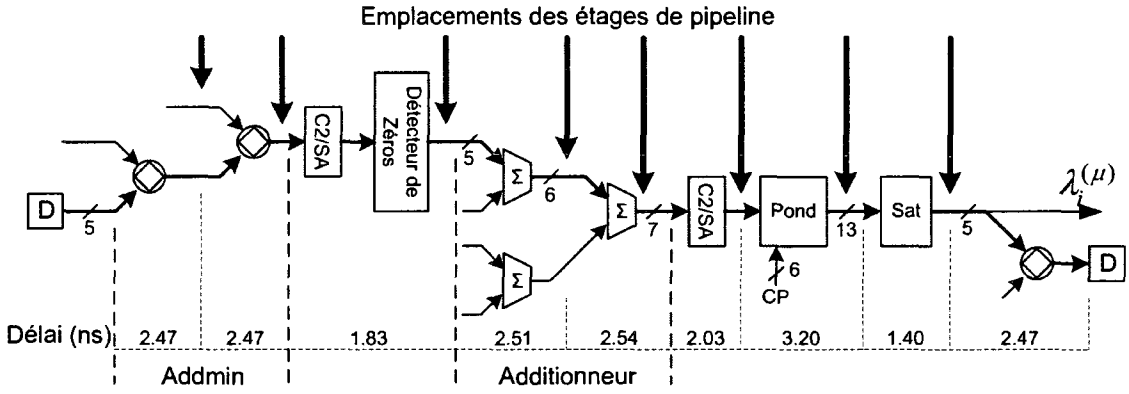

Figure 3.9: Emplacements des étages de pipeline dans le chemin critique du design

À la figure 3.9, les emplacements des étages de pipeline sont présentés ainsi qu'une estimation des délais qui correspondent à chaque composant du chemin critique. Afin de générer des estimations assez précises, une encapsulation VHDL qui enregistre les entrées et les sorties des composants a été développée. Ainsi, le délai combinatoire d'un composant est mesuré entre les registres d'entrée et de sortie. Que ce soit pour l'additionneur ou pour l'opérateur Addmin, un emplacement est fixé à chaque niveau dans l'arbre correspondant à ces deux opérateurs. Les 4 autres emplacements sont choisis de manière à séparer les opérateurs qui restent et à réduire efficacement le chemin critique. Aucune réduction supplémentaire de délais n'est nécessaire car le pondérateur, limité par la technologie, ne peut pas contenir un étage de pipeline. Soit m le nombre des niveaux de l'arbre du plus grand opérateur Addmin ayant J entrées, alors m peut être exprimé par:

$$m = \lceil log_2(J) \rceil \tag{3.9}$$

De même, soit n le nombre des niveaux de l'additionneur ayant $(J+1)$ entrées :

$$n = \lceil log_2(J+1) \rceil \tag{3.10}$$

Le nombre total des emplacements choisis (Nb_{Emp}) est donc donné par:

$$Nb_{Emp} = m + n + 4 = \lceil log_2(J) \rceil + \lceil log_2(J+1) \rceil + 4 \tag{3.11}$$

En somme, Nb_{Emp} dépend du nombre de connexions J du code convolutionnel utilisé. Les valeurs de Nb_{Emp} en fonction de J sont présentées dans le tableau 3.1.

Tableau 3.1: Nombre total des emplacements des étages de pipeline en fonction de J

J	m	n	Nb_{Emp}
4	2	3	9
5-7	3	3	10
8	3	4	11
9-15	4	4	12
16	4	5	13

Au niveau de code VHDL du décodeur, les emplacements sont définis par les trois vecteurs PIPE_ADDMIN, PIPE_ADDER et PIPE_DP (pour Data Path) de dimensions m, n et 4 respectivement. Les éléments de ces vecteurs peuvent avoir la valeur "1" pour indiquer l'existence de l'étage de pipeline et "0" dans le cas échéant. Toutefois, avant d'activer les étages de pipeline dans ces emplacements, il est nécessaire d'étudier la capacité de RD3 à offrir un nombre suffisant des bascules indispensables pour effectuer les trois étapes de resynchronisation au décodeur. Cette capacité se définit comme étant la capacité du pipelinage d'un codeur CDO.

3.3.3 Capacité de pipelinage d'un codeur CDO

Les bascules de la boucle de rétroaction utilisées pour effectuer l'*Étape 3* de resynchronisation sont introduites par l'*Étape 1*. Ces bascules sont obtenues par la resynchronisation de tous les opérateurs Addmin du RD3 montré à la figure 3.1 où les bascules ont été déplacées des sorties vers les entrées. En outre, le nombre de bascules disponibles à

la sortie de chaque opérateur Addmin dans RD3 correspond aux différences d_l données par (3.4) qui existent entre chaque positions de connexions consécutives du codeur CDO utilisé. Le nombre maximal des étages de pipeline qu'il est possible d'insérer dans le décodeur correspond au minimum des d_l, soit $min(d_l)$.

En somme, pour insérer p étages de pipeline dans le décodeur des codes CDO, il faut que p satisfasse la condition suivante :

$$p \leq min(d_l) \tag{3.12}$$

La relation (3.12) signifie que le nombre d'étages insérés ne doit pas dépasser le nombre de bascules libres à la sortie de chaque opérateur Addmin dans RD3. De plus, il faut tenir compte des deux considérations suivantes :

1. Au moins une bascule doit être maintenue à la sortie de chaque opérateur Addmin dans RD3 pour éviter le chevauchement du chemin critique vers le noyau de logique combinatoire.

2. Les étages de pipeline insérés aux m niveaux de l'opérateur Addmin global retournent les bascules initialement déplacées de d_1 à leur place tel que montré à la figure 3.8.

Ces deux considérations une fois combinées avec la condition (3.12) nous permettent de définir la capacité de pipelinage (P_c) d'un codeur CDO donné comme le nombre maximal des étages de pipeline que nous pouvons insérer dans le décodeur à seuil associé. P_c est défini comme suit :

$$P_c = min(d_1 + m, d_2, \cdots, d_{J-1}) - 1 \tag{3.13}$$

où m est le nombre de niveaux de l'opérateur Addmin donné par l'équation (3.9).

Afin de maximiser le débit du décodeur à seuil, la capacité de pipelinage P_c du codeur CDO doit être au moins égale au nombre des emplacements des étages de pipelines donné par le tableau 3.1. La capacité de pipelinage peut donc être introduite comme une nouvelle contrainte dans la recherche des meilleurs codeurs CDO ayant une mémoire α_J la plus courte possible et une capacité de pipelinage P_c élevée.

3.3.4 Implémentation des composants pipelinés

Dans l'architecture du décodeur à seuil, trois composants pipelinés sont distingués, soient l'additionneur, l'opérateur Addmin global (AG) et le registre à décalage de rétro-action (RD3).

3.3.4.1 Additionneur pipeliné

L'additionneur utilise à son entrée les J équations de parité, $\psi_{(i,j)}^{(\mu)}$ $(1 \leq j \leq J)$, et le symbole d'information, y_i^u. Ayant $(J+1)$ entrées représentées en C2, l'additionneur comporte J additionneurs élémentaires à deux entrées structurés selon une architecture arborescente afin de réduire les délais de propagation dans l'additionneur. Si le nombre de symboles à additionner est impair à un niveau donné, alors la résolution binaire du symbole en excès est étendue de 1 bit et cela en recopiant son MSB et le concaténant comme un nouveau MSB (exemple: $1011 \rightarrow 11011$, $0110 \rightarrow 00110$). Pour chaque élément non nul dans le vecteur PIPE_ADDER, une banque de registres de pipeline est installée directement à la sortie des additionneurs élémentaires du niveau correspondant. Un exemple d'implémentation de l'additionneur est montré à la figure 3.10 pour $J = 4$ et PIPE_ADDER = (0,1,1).

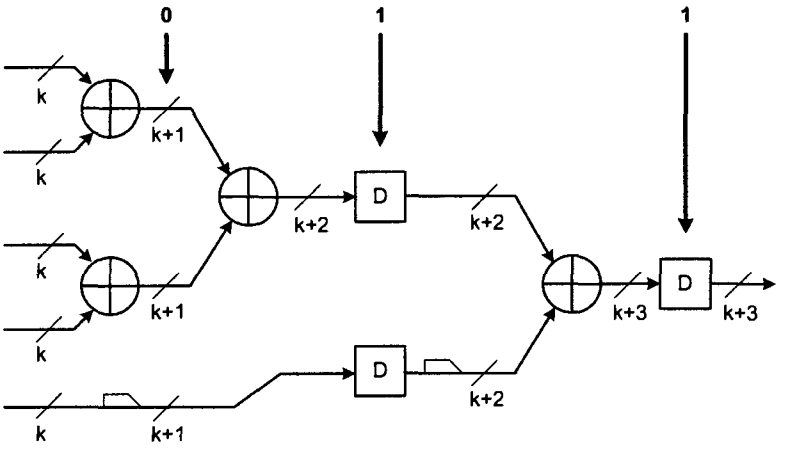

Figure 3.10: Architecture de l'additionneur, $J = 4$, PIPE_ADDER = (0,1,1)

3.3.4.2 Opérateur Addmin global pipeliné

L'opérateur Addmin global (AG) calcule les équations de parité, $\psi_{(i,j)}^{(\mu)}$, en combinant les sorties S_j de RD3 avec les termes impliquant les différences simples positives dans l'équation (2.13). Autrement dit, AG effectue toutes les opérations Addmin indiquées par l'équation (3.8). Sachant que $\psi_{(i,1)}^{(\mu)}$ est obtenue directement à la sortie S_1 de RD3, alors l'opérateur AG comporte $(J-1)$ opérateurs Addmin. Ces opérateurs Addmin ont des structures arborescentes ayant un nombre d'entrées allant de 2 pour celui qui correspond à $\psi_{(i,2)}^{(\mu)}$ jusqu'à la valeur de J pour celui qui correspond à $\psi_{(i,J)}^{(\mu)}$.

L'implémentation de l'arbre de chaque opérateur Addmin à l'intérieur du AG se fait de la même façon que l'arbre de l'additionneur à l'exception de garder la même résolution binaire en traversant les différents niveaux. Toutefois, l'application de la technique de pipelinage nécessite une attention particulière à cause de la différence dans le nombre de niveaux des opérateurs. Si les étages de pipeline excèdent les niveaux des opérateurs ayant un nombre faible d'entrées, alors le AG est implémenté de façon à les ajouter aux sorties de ces opérateurs afin de synchroniser tous les équations de parité. L'architecture du AG est montrée à la figure 3.11 pour $J = 5$ et PIPE_ADDMIN = (1,0,1).

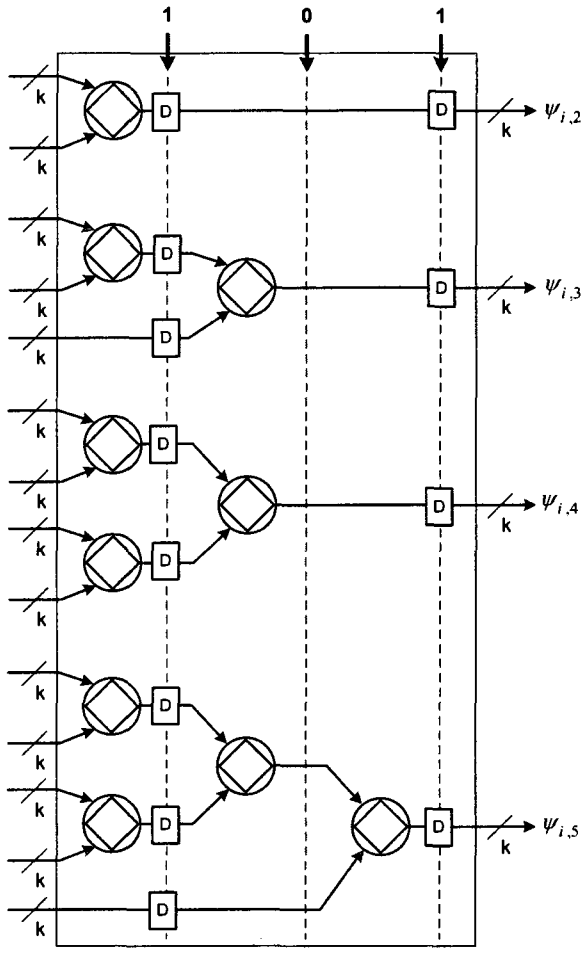

Figure 3.11: Architecture de l'opérateur AG, $J = 5$, PIPE_ADDMIN = (1,0,1)

3.3.4.3 Registre à décalage de rétroaction pipeliné

La nouvelle architecture pipelinée du registre à décalage de rétroaction (RD3) doit tenir compte du nombre de registres resynchronisés lors de l'application de la technique de pipelinage. Notons par Nb_{ADDMIN}, Nb_{ADDER} et Nb_{DP}, les nombres des éléments non nuls dans les vecteurs PIPE_ADDMIN, PIPE_ADDER et PIPE_DP respectivement, et par Nb_{PIPE} le nombre total des étages de pipeline à insérer dans le décodeur d'un code donné. Nb_{PIPE} doit être plus petit ou égale à la capacité de pipelinage du code, P_c :

$$Nb_{PIPE} = Nb_{ADDMIN} + Nb_{ADDER} + Nb_{DP} \leq P_c \tag{3.14}$$

Pour implémenter la nouvelle architecture de RD3, nous déplaçons Nb_{PIPE} registres de la sortie vers l'entrée de chaque opérateur Addmin dans RD3 à l'exception du dernier opérateur qui correspond à d_1. À l'entrée de cet opérateur, nous plaçons Nb_{PIPE} registres (comme les autres) mais à sa sortie nous retirons $Nb_{PIPE} - Nb_{ADDMIN}$ seulement pour tenir compte des étages de pipeline insérés aux niveaux du AG. La figure 3.12 illustre l'architecture de RD3 pour un générateur $J = 4$ donné par $A = \{0, 4, 9, 15\}$ avec $P_c = 4$, $Nb_{PIPE} = 3$, $Nb_{ADDMIN} = 2$, $d_1 = 4$, $d_2 = 5$ et $d_3 = 6$.

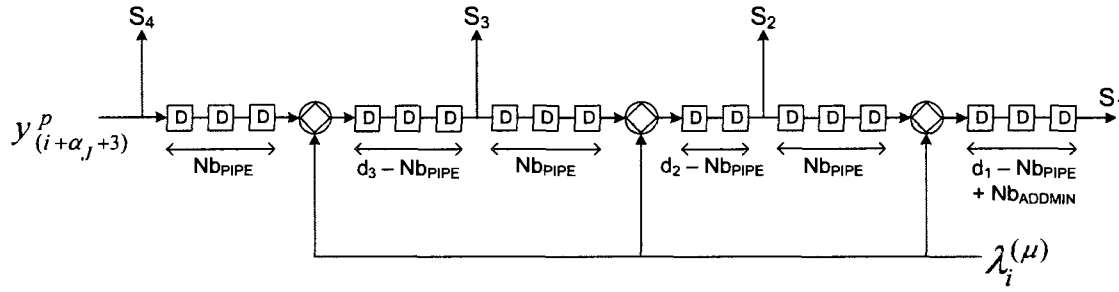

Figure 3.12: Architecture pipelinée de RD3, $A = \{0, 4, 9, 15\}$, $J = 4$, $P_c = 4$, $Nb_{PIPE} = 3$ et $Nb_{ADDMIN} = 2$

3.4 Système de communication adapté aux codes perforés

3.4.1 Introduction

Dans cette section, nous abordons les architectures de l'encodeur perforé et du DSI correspondant utilisées dans un système de communication adapté aux codes CDO perforés (PCDO). Rappelons que dans un tel système, la perforation consiste à éliminer de façon périodique certains symboles de parité générés par le codeur pour augmenter le taux de codage, R, du codeur origine (aussi appelé codeur "mère"). Ainsi, à partir d'un code convolutionnel systématique (SCC) de taux de codage $R = 1/2$, les taux de codage pouvant être obtenus sont égaux à $R = b/(b+1), b > 1$. Le mécanisme de perforation proposé se réalise en installant à la sortie de l'encodeur un module de perforation

appelé perforateur. À la réception, seuls les symboles provenant du canal interviennent dans le processus du décodage. En effet, un "dé-perforateur" qui insère des "0" dans la séquence reçue là où les symboles de parité ont été perforés est utilisé à l'entrée du DSI. Le schéma bloc d'un système de communication adapté aux codes PCDO est illustré à la figure 3.13.

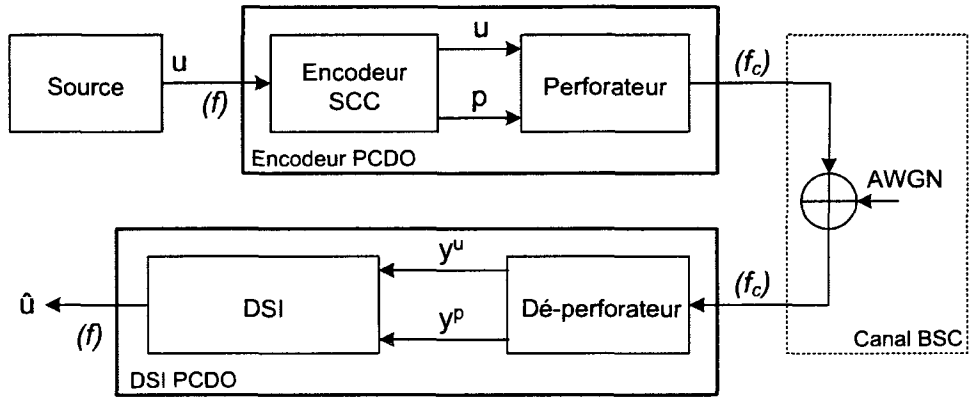

Figure 3.13: Schéma bloc d'un système de communication adapté aux codes perforés

Dans la figure 3.13, f_c représente le taux de transmission en symboles/s dans le canal tandis que f représente la fréquence d'opération maximale du décodeur. Ainsi, la même fréquence f est utilisée aussi dans la source et l'encodeur. À l'intérieur de l'encodeur PCDO, les symboles de parité (p) sont générés par l'encodeur SCC à la même fréquence f de la source. Pour obtenir un taux de codage $R = b/(b+1)$, le perforateur élimine périodiquement $(b-1)$ symboles de parité de chaque bloc de b paires (u, p) et envoie les $(b+1)$ symboles restants dans le canal à la fréquence f_c. Évidemment, à la sortie du perforateur les $(b+1)$ symboles émis à f_c occupent la même fenêtre temporelle que les b symboles d'informations (u) qui rentrent dans le décodeur à f. Alors, f_c peut être exprimée en fonction de f par la relation suivante :

$$f_c = \left(\frac{b+1}{b}\right) \cdot f = \frac{f}{R} \tag{3.15}$$

La relation (3.15) montre que si f est constant alors f_c dépend de la valeur du taux de codage fonctionnel du système. D'autre part, les travaux de recherche des codes PCDO [30, 31] ont montré que ces codes peuvent être générés à partir de codes mères de type SCC ou CDO. Étant donné que le taux de codage d'un code PCDO dépend des propriétés algébriques du code mère, alors il était possible de trouver des codes PCDO qui supportent plusieurs taux de codage [30]. Les codes fonctionnant à plusieurs taux de codage sont appelés "codes à taux compatibles". À titre d'exemple, tous les codes générés à partir de codes CDO sont à taux compatibles puisqu'ils supportent le taux $R = 1/2$ du code mère ainsi que les taux obtenus par perforation. En résumé, les codes PCDO exigent que la fréquence f_c soit synthétisée à partir de f exprimé par (3.15) et qu'elle soit modifiée dynamiquement en fonction du taux de codage pour des codes à taux compatibles. Ces fonctionnalités sont assurées par un module de gestion d'horloge ou le "gestionnaire d'horloge" qui a été intégré dans le système de communication.

3.4.2 Gestionnaire d'horloge

Le FPGA considéré dans ce projet ne dispose pas de sources d'horloge. Cependant, plusieurs primitives DCM (Digital Clock Manager) qui manipulent un signal d'horloge provenant de l'extérieur du FPGA sont disponible dans le type de FPGA utilisé dans ce projet. Chaque primitive DCM peut être utilisée comme un synthétiseur de fréquence qui génère avec haute précision une des fréquences d'horloge ($\frac{b+1}{b} \cdot f$) nécessaire pour implémenter le système de communication. Le gestionnaire d'horloge ainsi développé exploite les primitives DCM afin de répondre au besoin du système[2]. L'architecture du gestionnaire d'horloge est montrée à la figure 3.14.

On trouve à la sortie du gestionnaire d'horloge deux types de signaux, soient les signaux d'horloge (dCLK et cCLK) et les signaux de synchronisation (dLoad et cLoad).

[2] À l'annexe III, le code VHDL du gestionnaire d'horloge est présenté

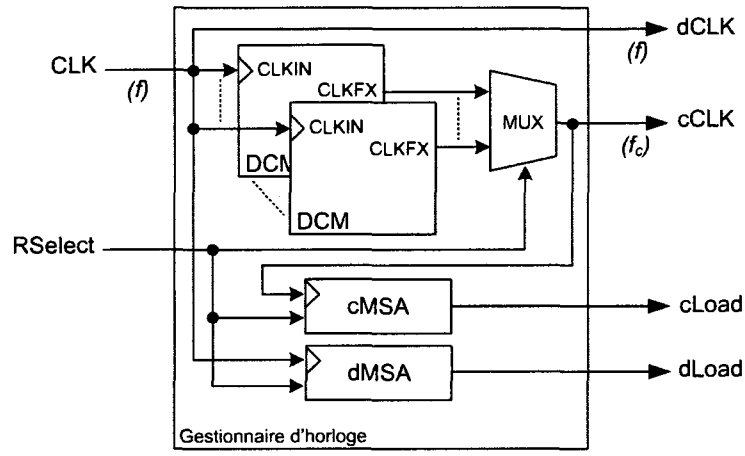

Figure 3.14: Architecture du gestionnaire d'horloge

Le signal d'horloge provenant de l'extérieur, CLK, est appliqué directement dans le gestionnaire d'horloge. La fréquence de CLK étant fixée à f, le même signal est envoyé à la sortie dCLK qui représente l'horloge du décodeur, de l'encodeur et de la source. Le gestionnaire d'horloge exploite une primitive DCM pour chaque taux de codage supporté par le code PCDO utilisé. La sortie "CLKFX" du DCM indique son fonctionnement en mode synthétiseur de fréquence. Si par exemple les taux $R = \frac{2}{3}$ et $\frac{3}{4}$ ($b = 2$ et 3 respectivement) sont demandés, alors deux DCM sont instanciés, un pour fournir la fréquence $\frac{3}{2}f$ et l'autre $\frac{4}{3}f$. L'entrée "RSelect" apporte au gestionnaire d'horloge la valeur de b. Un multiplexeur (MUX) contrôlé par RSelect choisit la fréquence f_c qui correspond au taux de codage fonctionnel. La fréquence choisie est alors envoyée à la sortie cCLK qui fournit l'horloge du canal.

Par ailleurs, les deux contrôleurs dMSA et cMSA sont intégrés dans le gestionnaire d'horloge afin de produire les signaux de synchronisation, dLoad et cLoad respectivement. Nous verrons dans les prochaines sections que dLoad et cLoad sont indispensables afin d'assurer des transferts synchronisés et à taux variable à la sortie du perforateur et à l'entrée du dé-perforateur. En recevant la valeur de b par RSelect en entrée, les contrôleurs dMSA et cMSA sont conçus de sorte que les signaux dLoad et cLoad soient actifs une fois chaque b et $(b + 1)$ cycles de dCLK et cCLK respectivement.

En somme, le gestionnaire d'horloge a permis de générer les signaux d'horloge et de synchronisation nécessaires pour le bon fonctionnement du système. Il a permis également de modifier dynamiquement le taux de codage R et cela en contrôlant son entrée RSelect.

3.4.3 Encodeur perforé

L'encodeur perforé (Encodeur PCDO) est formé d'un encodeur SCC du code mère ($R = \frac{1}{2}$) suivi d'un perforateur comme déjà montré à la figure 3.13. Evidemment, un module matériel ne peut pas se charger d'un nombre infini de taux de codage $R = \frac{b}{b+1}$. Ainsi, le perforateur développé supporte une gamme limitée de taux de codage allant de $R = \frac{1}{2}$ jusqu'à $R = \frac{8}{9}$. Cette gamme couvre tous les codes PCDO connus [30, 31]. Cependant, rien ne nous empêche d'étendre cette gamme pour n'importe quelle autre application. Le schéma bloc du perforateur est donné à la figure 3.15.

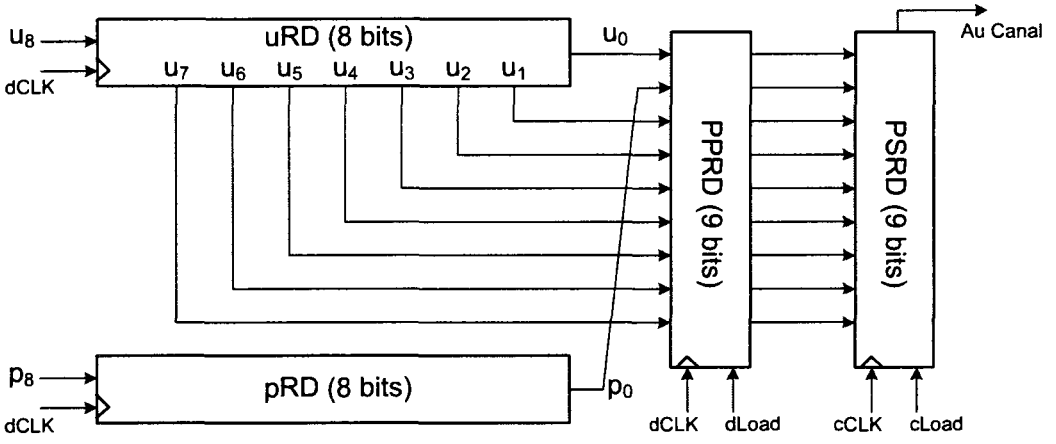

Figure 3.15: Architecture du perforateur

Il est démontré dans [30, 31] que la forme de la matrice de perforation, $\mathcal{P}$, n'influe pas les performances d'erreurs du décodeur. Le perforateur est alors conçu de façon à garder le premier symbole de parité et à éliminer le reste. Il applique sur les séquences d'information (**u**) et de parité (**p**) un patron de perforation ayant une largeur variable de la forme suivante:

$$\mathcal{P} = \begin{pmatrix} \underbrace{\begin{matrix} 1 & 1 & \cdots & 1 \\ 1 & 0 & \cdots & 0 \end{matrix}}_{b \text{ colonnes}} \end{pmatrix} \tag{3.16}$$

où b peut varier entre 1 et 8 pour $\frac{1}{2} \leq R \leq \frac{8}{9}$.

En se référant à la figure 3.15, les deux registres à décalage uRD et pRD de longueur 8 bits chacun reçoivent de l'encodeur SCC les symboles u et p respectivement. À l'entrée du convertisseur parallèle/parallèle (PPRD) les 8 symboles $(u_0, u_1, \cdots, u_7)$ et le symbole p_0 sont regroupés pour former le mot $(u_0, p_0, u_1, \cdots, u_7)$. Ce mot est chargé à la sortie de PPRD avec l'occurrence de dLoad. Notons que dLoad sera actif une fois à chaque b périodes de dCLK et que cLoad sera actif une fois à chaque $(b+1)$ périodes de cCLK. Ainsi, pendant une fenêtre temporelle de $T_{trans} = \frac{b}{f} = \frac{b+1}{f_c}$, dLoad et cLoad occurrent une fois chacun. En d'autres termes, le mot $(u_0, p_0, u_1, \cdots, u_7)$ reste fixe à la sortie de PPRD pendant le temps T_{trans} ce qui permet au convertisseur parallèle/sériel (PSRD) de le charger une seule fois avec l'occurrence de cLoad. De plus, le PSRD dispose de $(b+1)$ cycles de cCLK avant l'occurrence du prochain cLoad. Parmi les 9 symboles chargés par PSDR seulement les $(b+1)$ premiers, qui correspondent à $(u_0, p_0, \cdots, u_{b-1})$, sont envoyés au canal ce qui est en accord avec la matrice $\mathcal{P}$ donnée par (3.16). Les $(9-(b+1))$ symboles ainsi écrasés ne seront pas perdus puisqu'ils sont encore conservés dans uRD et pRD.

Il importe de noter que le perforateur est contrôlé par le gestionnaire d'horloge et plus précisément par RSelect. Par exemple, en fixant la valeur de RSelect à 1 (RSelect = b = 1), cLoad occurre chaque 2 cycles et seulement (u_0, p_0) sont envoyés au canal. Puis, au prochain cLoad, (u_1, p_1) sont envoyés et ainsi de suite. Dans ce cas, $R = \frac{1}{2}$ et le perforateur fonctionne comme un simple convertisseur parallèle/sériel (2 à 1) où aucune perforation n'est effectuée. Ce résultat peut être vérifié par (3.16) où b = 1 est équivalent

à réduire $\mathcal{P}$ à une matrice colonne (2x1) dont les deux éléments valent 1.

3.4.4 Décodeur à seuil itératif adapté aux codes perforés

L'architecture du décodeur à seuil itératif adapté aux codes perforés (DSI PCDO) est montrée à la figure 3.13. Elle est formée d'un dé-perforateur suivi d'un DSI associé au codeur SCC mère utilisé dans l'encodeur perforé. Le dé-perforateur assure que seuls les symboles non perforés reçus du canal interviennent dans le processus de décodage. Au décodage, il suffit donc d'annuler les équations $\psi_{(i,j)}^{(\mu)}$ qui correspondent aux symboles de parité perforés. Ainsi, le dé-perforateur dont l'architecture est illustrée à la figure 3.16 se charge d'effectuer cette tâche en insérant dans la séquence reçue des "0" aux positions où les symboles de parité étaient perforés.

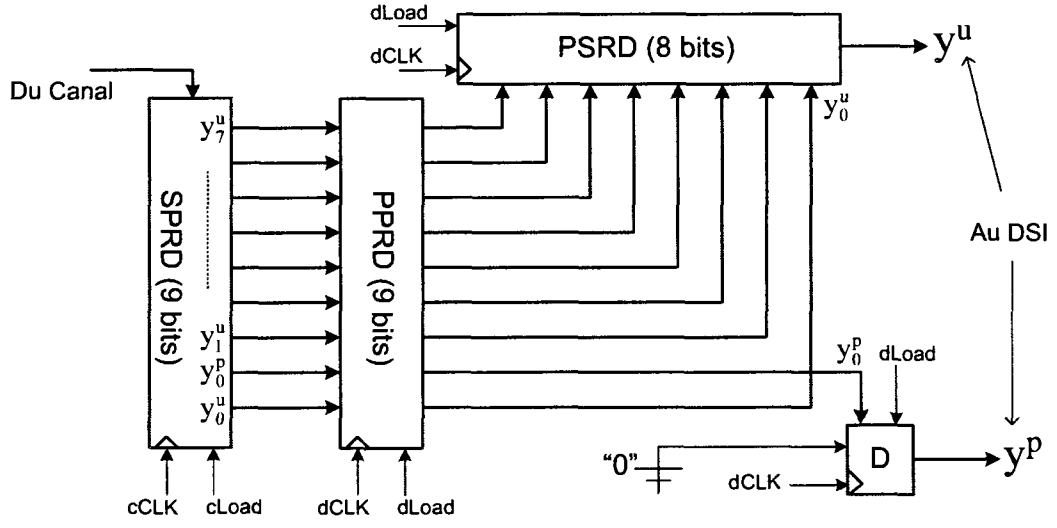

Figure 3.16: Architecture du dé-perforateur

Le dé-perforateur fonctionne exactement d'une façon opposée au perforateur. En effet, à l'entrée, les deux convertisseurs SPRD (sériel/parallèle) et PPRD interfacent les deux domaines d'horloge. Le mot $(y_0^u, y_0^p, y_1^u, \cdots, y_7^u)$ reste fixe à la sortie de PPRD pendant b cycles de dCLK jusqu'au prochain chargement avec dLoad. À la sortie, le PSRD envoie

au DSI les b premiers symboles y^u des 8 symboles chargés. Simultanément, la bascule D envoie au DSI le symbole y_0^p suivi de $(b-1)$ "0". Cette architecture supporte la même gamme de taux de codage que le perforateur et qui peuvent être choisis aussi par RSelect. Cependant, il faut noter que tous les symboles y^u et y^p sont représentés sur 3 bits à l'intérieur du dé-perforateur.

3.5 Recherche de générateurs des codes PCDO à taux compatibles et à haute capacité de pipelinage

Bien que l'auteur de [30] ait présenté un ensemble de générateurs de codes PCDO à taux compatibles, pour les raisons énumérées ci-dessous, il importe d'effectuer une nouvelle recherche de générateurs :

1. La capacité de pipelinage, P_c, des générateurs publiés dans [30] était trop faible pour des applications à haut débit.

2. Dans [30], seulement quelques taux de codage ont été considérés ($\frac{1}{2}$, $\frac{2}{3}$ et $\frac{3}{4}$) pour les codes à taux compatibles ce qui ne répond pas entièrement aux exigences des applications modernes telles que la norme WiMax [5] où quatres taux de codage $\frac{1}{2}$, $\frac{2}{3}$, $\frac{3}{4}$ et $\frac{5}{6}$ sont requis.

Rappelons qu'une protection EEP est offerte à un taux $R = \frac{b}{b+1}$ si et seulement si b est un diviseur de J [30]. Les nouveaux générateurs PCDO à taux compatibles présentés dans le tableau 3.2 ont été trouvés en utilisant un algorithme de recherche semblable à celui développé par [30] mais dans lequel une contrainte liée à la capacité de pipelinage a été incorporée. Chacun de ces codes a une capacité de pipelinage élevée qui nous permet de lui implémenter un DSI à haut débit.

Si b ne constitue pas un diviseur de J pour un des taux de codage considérés, alors le code est recherché de sorte à offrir une protection "quasi-EEP" au taux de codage en question. Par définition, un codeur ayant J connexions offrant une protection UEP à un taux de codage $R = \frac{b}{b+1}$, est dit quasi-EEP au même taux de codage si les cardinalités des sous-ensembles A_h ($|A_h|$), $h = 1, \cdots, b$, sont presque uniforme :

$$\lfloor \frac{J}{b} \rfloor \leq |A_h| \leq (\lfloor \frac{J}{b} \rfloor + 1) \tag{3.17}$$

L'importance de la protection quasi-EEP par rapport à une protection UEP réside dans le fait que chaque bit est décodé en utilisant au moins $\lfloor \frac{J}{b} \rfloor$ équations de parité ce qui garantit la convergence du processus de décodage itératif.

3.6 Conclusion

Une description détaillée du décodeur à seuil des codes CDO a été présentée dans ce chapitre. L'architecture du système de communication adapté aux codes perforés a été également présentée ainsi que les architectures des modules de perforation. La nouvelle technique de pipelinage du décodeur a été aussi décrite. Cette technique a permis d'augmenter le débit du décodeur à seuil. Cependant, cette augmentation reste limitée vu que le décodeur à seuil des codes CDO ne présente aucun aspect de parallélisme. Dans le chapitre suivant, cette limitation est abordée et de nouveaux codes multi-registres à décalage permettant ainsi d'augmenter davantage le débit du décodeur sont présentés.

Tableau 3.2: Ensemble des meilleurs connexions $\alpha_j \in A$ pour les codes PCDO à taux compatibles : Nombre de connexions J, Capacité de pipelinage P_c, Taux de codage supportés R, Facteur de simplification pour chaque taux supporté δ_{max} et Ensemble des connexions $\{\alpha_j\}$

J	P_c	R	δ_{max}	Ensemble des connexions $\{\alpha_j\}$
8	11	1/2 2/3 4/5	0.485 0.227 0.070	{0, 9, 25, 46, 95, 112, 126, 139}
9	13	1/2 2/3 3/4 4/5	0.452 0.214 0.1 0	{0, 10, 29, 61, 123, 179, 202, 228, 242}
10	12	1/2 2/3 3/4 5/6	0.832 0.462 0.292 0	{0, 9, 34, 107, 127, 208, 311, 330, 353, 366}
12	15	1/2 2/3 3/4 4/5 5/6 6/7	0.418 0.264 0.114 0.076 0.061 0.054	{0, 14, 31, 138, 243, 284, 443, 569, 592, 850, 921, 937}
14	12	1/2 2/3 3/4 4/5 5/6 7/8	0.419 0.236 0.140 0.074 0.038 0.009	{0, 9, 29, 46, 59, 73, 158, 265, 527, 1006, 1112, 1342, 1800, 1911}
15	12	1/2 2/3 3/4 4/5 5/6 7/8	0.389 0.215 0.136 0.061 0.055 0.014	{0, 135, 176, 192, 205, 222, 243, 358, 526, 791, 1243, 1889, 2069, 2789, 2932}

CHAPITRE 4

CODES CONVOLUTIONNELS DOUBLEMENT ORTHOGONAUX À MULTI-REGISTRES À DÉCALAGE

4.1 Introduction

La technique de pipelinage appliquée au DSI des codes CDO présentée à la section 3.3 nous a permis d'améliorer efficacement le débit d'information à la sortie du décodeur. Cependant, de par la nature du décodage à seuil, le DSI ne décode qu'un seul bit d'information à chaque coup d'horloge. Afin d'augmenter davantage le débit du processus de décodage, il est indispensable de considérer des codes pouvant être décodés en utilisant un DSI à architecture parallèle permettant de manipuler simultanément plusieurs bits à chaque coup d'horloge. Cependant, les seuls codes CDO qui offrent cet aspect de parallélisme sont les codes CDO définis au sens strict (CDO-SS) introduits dans [24].

Selon [24], un codeur CDO-SS est spécifié par une matrice de valeurs entières de dimension $J \times J$ et un taux de codage $R = J/2J = 1/2$. En d'autres termes, les codeurs CDO-SS ont J registres à décalages dénotés J-CDO-SS. Le décodage à seuil itératif des codes J-CDO-SS permet un décodage de J bits par période d'horloge conduisant ainsi à un débit J fois plus élevé que celui d'un DSI pour des codes CDO utilisant un seul registre à décalage. De plus, la double orthogonalité définie au sens strict élimine toutes les répétitions inévitables de différences doubles présentes à la deuxième itération du DSI. Ainsi, les performances d'erreur des codes J-CDO-SS sont substantiellement plus élevées que celles des codes CDO. En effet, à de faibles rapports signal sur bruit ($2 < E_b/N_0 < 3dB$), les codes J-CDO-SS convergent rapidement, nécessitant un nombre d'itérations réduit comparativement aux codes CDO [24]. En contrepartie, les

codeurs J-CDO-SS ont des longueurs de mémoire élevées ce qui se traduit par une grande complexité matérielle au niveau du processus de décodage, limitant ainsi l'utilisation de ces codes.

Dans ce chapitre, nous visons l'implémentation de DSI à haut débit ayant d'une part des architectures parallèles permettant des débits élevés, et d'autre part ayant des complexités matérielles abordables. Pour ce faire, nous étudions la double orthogonalité des codes convolutionnels systématiques à multiples registres à décalage. Les codes convolutionnels systématiques doublement orthogonaux à multi-registres à décalage sont dénotés par M-CDO. Une nouvelle simplification des conditions de la double orthogonalité de codes M-CDO est par la suite introduite. Cette simplification va permettre de réduire efficacement la complexité des DSI correspondants sans trop dégrader leurs performances d'erreurs. Ainsi, les architectures de l'encodeur M-CDO et celle du DSI à haut débit de codes M-CDO seront abordées. De plus, un ensemble de générateurs pour ces nouveaux codes est présenté.

4.2 Définition des codes M-CDO

Un encodeur M-CDO est spécifié par l'ensemble des connexions $\mathcal{A} = [\alpha_{m,q,j}]$ de trois dimensions $M \times Q \times J_{RA}$ où, M représente le nombre de registres à décalage de l'encodeur, Q est le nombre de symboles de parité générés à la sortie de l'encodeur et qui correspond aussi au nombre de ses additionneurs modulo 2 et finalement, J_{RA} représente le nombre maximal de connexions qui existent entre un registre à décalage et un additionneur modulo 2 donnés. Le taux de codage du codeur M-CDO est alors donné par :

$$R = M/(M+Q) \tag{4.1}$$

Chaque valeur $\alpha_{m,q,j}$ de l'ensemble des connexions $\mathcal{A}$, représente la position d'une des

connexions qui existent entre le $m^{ième}$ symbole d'information et le $q^{ième}$ symbole de parité. Ainsi, un codeur M-CDO peut être représenté par une matrice génératrice de 2 dimensions, $M \times Q$ dont les entrées sont des ensembles de connexions $A_{m,q}$ ayant des dimensions inférieures ou égales à J_{RA}. Un exemple d'un code M-CDO ayant $M = 3$, $Q = J_{RA} = 2$ et par suite $R = 3/5$ est montré par la matrice génératrice $\boldsymbol{\mathcal{A}}$ suivante :

$$\boldsymbol{\mathcal{A}} = [A_{m,q}] = [\alpha_{m,q,j}] = \left[\begin{array}{c|c} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \\ A_{3,1} & A_{3,2} \end{array}\right] = \left[\begin{array}{cc|cc} 0 & 2 & 0 & 5 \\ 0 & 4 & 1 & -1 \\ 1 & -1 & 0 & 3 \end{array}\right] \tag{4.2}$$

où les "-1" indiquent l'absence de connexion. Par exemple, $A_{1,1} = \{0, 2\}$, $A_{3,1} = \{1, -1\}$, etc... Par convention, les codes M-CDO seront notés en utilisant la valeur de M, soit 3-CDO pour les codes générés par (4.2). À la sortie de l'encodeur, le processus de l'encodage consiste à générer à chaque instant i, $i = 1, 2, \cdots$, les symboles de parité de la façon suivante :

$$p_{q,i} = \sum_{m=1}^{M} \bigoplus \sum_{j=1}^{J_{RA}} \bigoplus u_{m,i-\alpha_{m,q,j}} \text{ avec } \alpha_{m,q,j} \geq 0 \text{ et } q = 1, 2, \cdots, Q \tag{4.3}$$

La figure 4.1 montre l'encodeur 3-CDO de taux de codage $R = 3/5$ spécifié par la matrice $\boldsymbol{\mathcal{A}}$ montrée à (4.2).

L'algorithme de décodage à seuil itératif pour les codes M-CDO est semblable à celui des codes CDO. Ainsi, en suivant un développement semblable à celui qui a conduit l'auteur de [24] à (2.12), on obtient à chaque instant i, $i = 1, 2, \cdots$, à la sortie de la première itération:

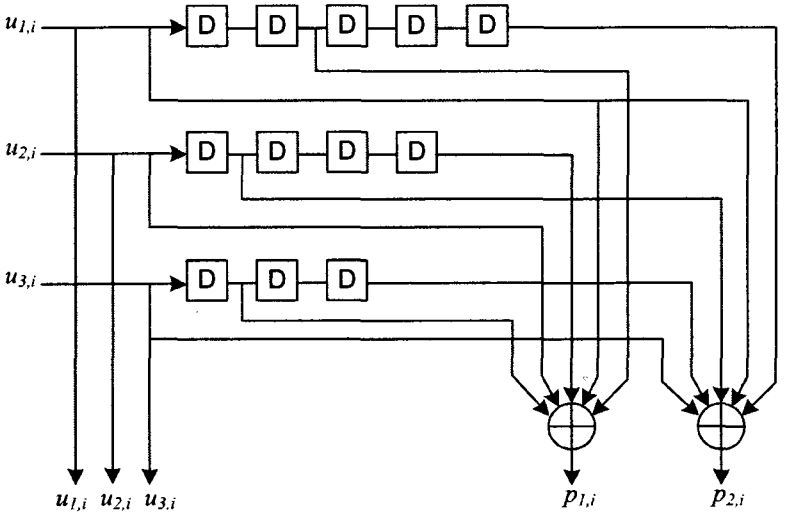

Figure 4.1: Exemple d'un encodeur 3-CDO, $R = 3/5$

$$\lambda_{m,i}^{(1)} = y_{m,i}^{u} + \sum_{q=1}^{Q} \sum_{j=1}^{J_{RA}} \left(y_{q,i+\alpha_{m,q,j}}^{p} \diamond \sum_{n=1}^{M} \diamond \sum_{\substack{k=1,(m,j)\neq(n,k) \\ (\alpha_{m,q,j}-\alpha_{n,q,k})<0}}^{J_{RA}} \lambda_{n,i+(\alpha_{m,q,j}-\alpha_{n,q,k})}^{(1)} \right.$$
$$\left. \diamond \sum_{n=1}^{M} \diamond \sum_{\substack{k=1,(m,j)\neq(n,k) \\ (\alpha_{m,q,j}-\alpha_{n,q,k})\geq 0}}^{J_{RA}} y_{n,i+(\alpha_{m,q,j}-\alpha_{n,q,k})}^{u} \right) \tag{4.4}$$

avec $\alpha_{m,q,j}, \alpha_{n,q,k} \geq 0$, et $m = 1, 2, \cdots, M$. Pour $(\alpha_{m,q,j} - \alpha_{n,q,k}) < 0$, les termes $\lambda_{n,i+(\alpha_{m,q,j}-\alpha_{n,q,k})}^{(1)}$ ayant des indices temporels négatifs par rapport à i représentent la rétroaction sur la décision lors du processus de décodage. En continuant davantage le développement, on obtient à la sortie de la deuxième itération :

$$\lambda_{m,i}^{(2)} = y_{m,i}^{u} + \sum_{q=1}^{Q}\sum_{j=1}^{J_{RA}}\Bigg(y_{q,i+\alpha_{m,q,j}}^{p} \diamond \sum_{l=1}^{M}\diamond \sum_{\substack{k=1,(m,j)\neq(l,k)\\(\alpha_{m,q,j}-\alpha_{l,q,k})<0}}^{J_{RA}}\diamond \lambda_{l,i+(\alpha_{m,q,j}-\alpha_{l,q,k})}^{(2)} \diamond$$
$$\sum_{l=1}^{M}\diamond \sum_{\substack{k=1,(m,j)\neq(l,k)\\(\alpha_{m,q,j}-\alpha_{l,q,k})\geq 0}}^{J_{RA}}\diamond \Bigg(y_{l,i+(\alpha_{m,q,j}-\alpha_{l,q,k})}^{u} + \sum_{p=1}^{Q}\sum_{r=1}^{J_{RA}}\Big(y_{p,i+\alpha_{l,p,r}+(\alpha_{m,q,j}-\alpha_{l,q,k})}^{p}$$
$$\diamond \sum_{n=1}^{M}\diamond \sum_{\substack{s=1\\(l,r)\neq(n,s)}}^{J_{RA}}\diamond \Lambda_{n,i+(\alpha_{m,q,j}-\alpha_{l,q,k})+(\alpha_{l,p,r}-\alpha_{n,p,s})}\Big)\Bigg)\Bigg) \tag{4.5}$$

avec $\alpha_{m,q,j}, \alpha_{n,q,k}, \alpha_{n,p,r}, \alpha_{l,p,s} \geq 0,\ m = 1, 2, \cdots, M$ et :

$$\Lambda = \begin{cases} y^{u} & \text{si } (\alpha_{l,p,r} - \alpha_{n,p,s}) \geq 0 \\ \lambda^{(1)} & \text{si } (\alpha_{l,p,r} - \alpha_{n,p,s}) < 0 \end{cases} \tag{4.6}$$

À partir de (4.5), les codeurs M-CDO peuvent être définis comme suit :

Définition 3 *Un codeur convolutionnel systématique à M registres à décalages et de taux de codage $R = M/(M+Q)$ est dit doublement orthogonal (M-CDO) si les positions de connexions $[\alpha_{m,q,j}]$ répondent aux conditions suivantes :*

1. *Les différences simples, $(\alpha_{m,b,c} - \alpha_{n,b,d})$, sont distinctes.*
2. *Les différences doubles, $(\alpha_{m,q,j} - \alpha_{l,q,k}) + (\alpha_{l,p,r} - \alpha_{n,p,s})$, sont distinctes à l'exception des répétitions inévitables provenant de la permutation d'indices $[(q,j),(p,r)]$ si $m = l$ ou $[(q,k),(p,s)]$ si $l = n$.*
3. *Les différences doubles sont distinctes des différences simples.*

Pour toutes les combinaisons des indices : $l, m, n \in \{1, \cdots, M\}$, $b, p, q \in \{1, \cdots, Q\}$,

$c, d, j, k, s, r \in \{1, \cdots, J_{RA}\}$, $(m, c) \neq (n, d)$, $(m, j) \neq (l, k)$, $(m, q, j) \neq (n, p, s)$, $(l, r) \neq (n, s)$ *et* $(q, k) \neq (p, r)$.

Remarquons que pour $m = n = l$ et $b = q = p$, la définition 3 montre que chacun des ensembles $A_{m,q}$ est un générateur dec codes CDO. En inspectant (4.4) et (4.5), nous constatons que la double sommation $\sum_{q=1}^{Q} \sum_{j=1}^{J_{RA}} (\cdots)$ représente les équations de parité utilisées pour décoder les symboles d'information. Pour chaque sortie $\lambda_{m,i}^{(\mu)}$ de l'itération μ, le nombre de ces équations correspond au nombre des entrées $\alpha_{m,q,j} \geq 0$ à la ligne m de la matrice $\boldsymbol{\mathcal{A}}$. Soit J_m, $J_m \leq (Q \times J_{RA})$, le nombre des équations de parité utilisées pour calculer la sortie $\lambda_{m,i}^{(\mu)}$. Par analogie avec les codes PCDO, si tous les J_m, $m = 1, \cdots, M$, sont égales, alors, tous les symboles d'information seront décodés en utilisant le même nombre d'équations de parité, et nous dirons alors que les codes M-CDO offrent une protection égale sur tous les symboles décodés et seront dénotés EEP (en anglais, Equal Error Protection). Dans le cas contraire, les codes M-CDO offriront une protection inégale sur les symboles décodés et seront dénotés UEP (en anglais, Unequal Error Protection). Pour tous les codes M-CDO, soit J le nombre minimal d'équations de parité utilisées dans le processus du décodage, où :

$$J = min(J_m), \; m \in \{1, \cdots, M\} \tag{4.7}$$

Dans le cas des codes M-CDO définis EEP :

$$J = J_m, \; \forall \, m \in \{1, \cdots, M\} \tag{4.8}$$

Par exemple, en considérant la matrice génératrice montrée par (4.2), on trouve $J_1 = 4$ et $J_2 = J_3 = 3$. Alors $J = 3$ et les codes 3-CDO ainsi générés sont définis UEP.

Soit par définition, α_J, la longueur de mémoire qui correspond au plus long registre à

décalage dans un encodeur M-CDO :

$$\alpha_J = \max_{m,q,j}(\alpha_{m,q,j}) \tag{4.9}$$

La latence L du décodeur à seuil (1 itération) de codes M-CDO sera donnée par :

$$L = M \times \alpha_J \tag{4.10}$$

Il importe de noter que la définition des codes M-CDO contient les définitions des autres codes doublement orthogonaux de taux de codage $R = 1/2$. En effet, si $M = Q = 1$ on retrouve la définition des codes CDO ayant $J = J_{RA}$ positions de connexions. En particulier, si $J_{RA} = 1$ et $M = Q = J$, alors, on retrouve la définition des codes J-CDO-SS de taux de codage $R = J/2J$.

4.3 Simplification des codes M-CDO

La représentation en trois dimensions des positions de connexions, $\alpha_{m,q,j}$, rend l'analyse des différences simples et doubles de codes M-CDO très complexe. Afin de mettre en évidence la signification de ces différences et leur rôle dans le processus de décodage à seuil itératif, une représentation vectorielle des différences a été adoptée. En utilisant cette nouvelle représentation vectorielle, il est possible de définir le facteur de simplification d'une matrice génératrice de codes M-CDO. De plus, une nouvelle simplification qui réduit davantage la mémoire de l'encodeur M-CDO ainsi que la complexité de DSIs sera introduite.

4.3.1 Représentation vectorielle des différences simples et doubles

Chaque terme de (4.4) associé à la différence simple $(\alpha_{m,q,j} - \alpha_{n,q,k})$ représente la contribution du $n^{ième}$ symbole d'information dans le calcul de la fiabilité $\lambda_m^{(1)}$ correspondant au $m^{ième}$ symbole d'information. Cette contribution est introduite à partir du $q^{ième}$ symbole de parité. Nous allons désormais désigner cette contribution par un vecteur $\vec{S}_{nm,q}$ qui passe de l'ensemble $A_{n,q}$ à l'ensemble $A_{m,q}$ dans la matrice génératrice $\boldsymbol{\mathcal{A}}$ des codes M-CDO. Le vecteur de contribution $\vec{S}_{nm,q}$ représente toutes les différences simples $(\alpha_{m,q,j} - \alpha_{n,q,k})$ qui se produisent en considérant toutes les combinaisons admissibles de j et k, soit $(m,j) \neq (n,k)$. Des exemples du vecteur $\vec{S}_{nm,q}$ sont montrés à la figure 4.2.

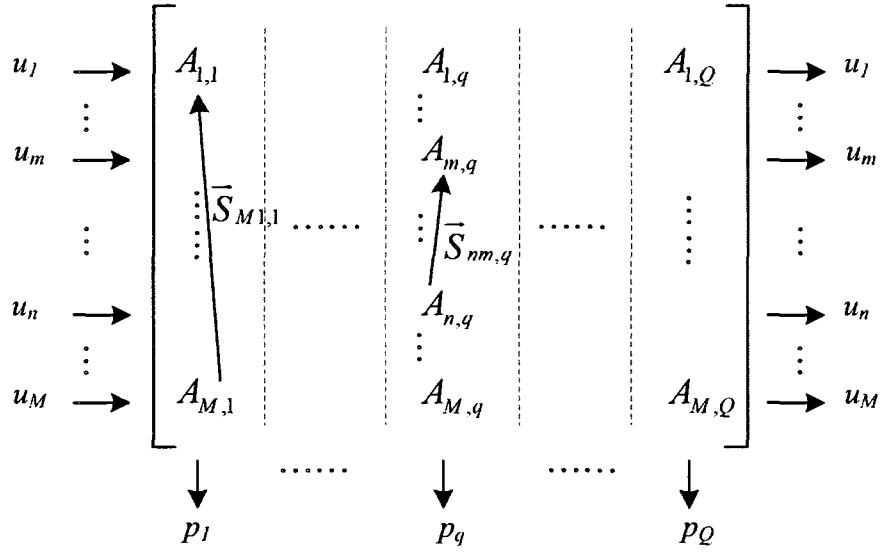

Figure 4.2: Exemples de la matrice $\boldsymbol{\mathcal{A}}$ et du vecteur $\vec{S}$

Sans perte de généralité, notons par $\vec{S}_{nm}$ toutes les différences simples admissibles $(\alpha_{m,q,j} - \alpha_{n,q,k})$, $q \in \{1, \cdots, Q\}$, soit :

$$\vec{S}_{nm} = [\vec{S}_{nm,1}, \cdots, \vec{S}_{nm,Q}] = \bigcup_{1 \leq q \leq Q} \vec{S}_{nm,q} \tag{4.11}$$

Il importe de noter que les vecteurs $\vec{S}_{nm,q}$ sont des vecteurs intra-colonne c'est-à-dire limités à la colonne q et ne se tracent jamais entre deux colonnes de la matrice $\mathcal{A}$.

Les différences doubles $(\alpha_{m,q,j} - \alpha_{l,q,k}) + (\alpha_{l,p,r} - \alpha_{n,p,s})$ seront représentées par des vecteurs de contribution $\vec{D}_{nlm,pq}$, avec :

$$\vec{D}_{nlm,pq} = \vec{S}_{nl,p} + \vec{S}_{lm,q},\ l \in \{1, \cdots, M\} \tag{4.12}$$

En effet, comme montré à la figure 4.3, le vecteur $\vec{D}_{nlm,pq}$ exprime la contribution de la fiabilité du $n^{ième}$ symbole d'information dans le processus du décodage de $l^{ième}$ symbole à la première itération. À son tour, le $l^{ième}$ symbole d'information participe au décodage du $m^{ième}$ symbole d'information à la deuxième itération. Notons par $\vec{D}_{nm}$ toutes les différences doubles $(\alpha_{m,q,j} - \alpha_{l,q,k}) + (\alpha_{l,p,r} - \alpha_{n,p,s})$ admissibles, soit $(m,j) \neq (l,k)$, $(m,q,j) \neq (n,p,s)$, $(l,r) \neq (n,s)$ et $(q,k) \neq (p,r)$, alors :

$$\vec{D}_{nm} = \bigcup_{l,p,q} \vec{D}_{nlm,pq} \ \forall\, l \in \{1, \cdots, M\} \text{ et } p, q \in \{1, \cdots, Q\} \tag{4.13}$$

En utilisant ces nouvelles représentations vectorielles des différences simples et doubles, la définition 3 peut être récrite comme suit :

Définition 4 *Un encodeur convolutionnel systématique à M registres à décalages et de taux de codage $R = M/(M+Q)$ est dit doublement orthogonal (M-CDO) si sa matrice génératrice $\mathcal{A}$ satisfait aux conditions suivantes :*

Pour chaque n et $m \in \{1, \cdots, M\}$:

1. *Les différences simples, $\vec{S}_{nm}$, sont distinctes.*

2. *Les différences doubles, $\vec{D}_{nm}$, sont distinctes à l'exception des répétitions inévi-*

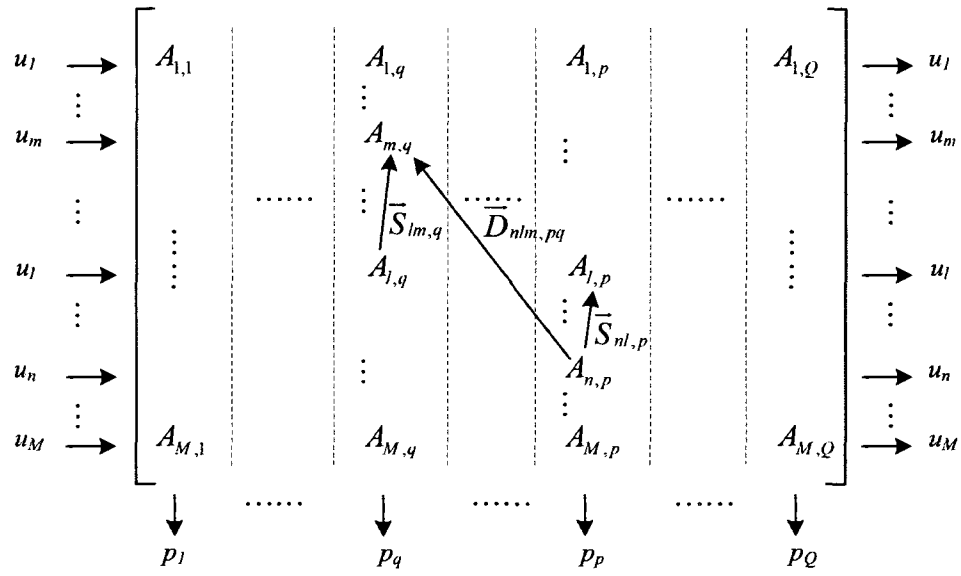

Figure 4.3: Exemple du vecteur $\vec{D}_{nlm,pq}$

tables.

3. *Les différences doubles, $\vec{D}_{nm}$, sont distinctes des différences simples, $\vec{S}_{nm}$.*

4.3.2 Simplification des conditions de la définition des codes M-CDO

Pour les codes CDO, les auteurs de [40] ont étudié le rôle de chacune des conditions de la définition 1 dans le processus du décodage à seuil itératif. De cette analyse, nous pouvons tirer que la condition 1 est la plus importante à respecter. Ensuite, par ordre d'importance, les conditions 3 et 2 doivent être respectées pour maintenir de bonnes performances d'erreur. Cependant, l'auteur de [30] a relaxé la deuxième condition (la moins importante) afin de définir les codes CDO simplifiés. Il a démontré que les performances de codes CDO simplifiés convergent vers les performances des codes CDO non simplifiés à forts rapports signal sur bruit.

Dans cette section, nous allons considérer la simplification des conditions 2 et 3 dans la définition 4 des codes M-CDO en respectant en tous temps la condition 1. Nous verrons

les effets de ces simplifications sur la complexité et les performances d'erreur des DSIs dans le chapitre suivant où nous y présentons les résultats expérimentaux.

4.3.2.1 Simplification de la deuxième condition

Comme dans le cas de codes CDO simplifiés, la simplification de la deuxième condition dans la définition 4 consiste à permettre un certain nombre de différences doubles égales, en plus de celles obtenues par les répétitions inévitables d'indices. Soient N_{Dnm} le nombre total des différences doubles obtenues *en incluant* les répétitions inévitables d'indices et N^e_{Dnm} le nombre de différences doubles égales. Nous définissons les facteurs de simplification [30] de codes M-CDO, δ_{nm}, par l'expression suivante :

$$\delta_{nm} = \frac{N^e_{Dnm}}{N_{Dnm}}, \;\; 0 \leq \delta_{nm} < 1, \;\; n, m \in \{1, \cdots, M\} \tag{4.14}$$

Il est clair que la simplification de la deuxième condition induit M^2 facteurs de simplification, δ_{nm}. Par convention, nous caractérisons les codes M-CDO à l'aide du facteur maximum, δ_{max} :

$$\delta_{max} = \max_{n,m \in \{1,\cdots,M\}} \delta_{nm} \tag{4.15}$$

4.3.2.2 Simplification de la troisième condition

Cette nouvelle simplification consiste à relaxer la troisième condition de la définition 4. La troisième condition exige que les différences simples et doubles soient distinctes. Toutefois, afin de réduire davantage le span, α_J des codeurs M-CDO, nous allons permettre à un nombre limité de différences simples et doubles d'être égales. Les codes M-CDO simplifiés ainsi obtenus sont simplement orthogonaux, tout en gardant un certain degré de double orthogonalité. Ces codes "partiellement" doubles orthogonaux sont

dénotés M-CPDO. Nous les caractérisons par le même facteur de simplification δ_{max} défini ci-haut (4.15) puisque δ_{max} représente le niveau maximal de dégénération de la double orthogonalité des codes[1].

4.4 Décodeur à seuil itératif à haut débit de codes M-CDO

Comme dans le cas des codes CDO et J-CDO-SS, les codes M-CDO seront décodés en utilisant l'algorithme de décodage à seuil itératif proposé par l'auteur de [24]. Pour un DSI formé de N itérations, cet algorithme consiste à appliquer à la première itération l'équation (4.4) sur les symboles M-CDO reçus du canal et, par la suite, de continuer le processus de décodage à seuil itératif en calculant la sortie de chaque itération μ, ($\mu > 1$), de la façon suivante :

$$\begin{aligned}
\lambda_{m,i}^{(\mu)} &= y_{m,i}^{u} + \sum_{j_m=1}^{J_m} \psi_{m,j_m,i}^{\mu} \\
&= y_{m,i}^{u} + \sum_{q=1}^{Q} \sum_{j=1}^{J_{RA}} \Bigg(y_{q,i+\alpha_{m,q,j}}^{p} \diamond \sum_{n=1}^{M}{}^{\diamond} \sum_{\substack{k=1,(m,j)\neq(n,k) \\ (\alpha_{m,q,j}-\alpha_{n,q,k})<0}}^{J_{RA}}{}^{\diamond} \lambda_{n,i+(\alpha_{m,q,j}-\alpha_{n,q,k})}^{(\mu)} \\
&\qquad \diamond \sum_{n=1}^{M}{}^{\diamond} \sum_{\substack{k=1,(m,j)\neq(n,k) \\ (\alpha_{m,q,j}-\alpha_{n,q,k})\geq 0}}^{J_{RA}}{}^{\diamond} \lambda_{n,i+(\alpha_{m,q,j}-\alpha_{n,q,k})}^{(\mu-1)} \Bigg)
\end{aligned} \tag{4.16}$$

Le décodeur à seuil des codes M-CDO est, en effet, une version parallèle du décodeur à seuil des codes CDO présenté au chapitre précédent (figure 3.1). Ainsi, le schéma bloc du décodeur à seuil des codes M-CDO qui correspond à l'itération μ est montré à la figure 4.4. Ce décodeur est aussi composé des deux parties principales : Les registres

[1] Les résultats expérimentaux présentés au chapitre suivant montrent l'exactitude de cette approche

à décalage et le noyau de logique combinatoire. Il importe de noter que l'architecture du décodeur à seuil reste la même que ce soit pour des codes M-CDO simplifiés ou non simplifiés. Nous abordons dans les sections suivantes, les architectures des différents composants du décodeur à seuil et cela en considérant, à titre d'exemple, la matrice génératrice 3-CDO donnée par (4.2).

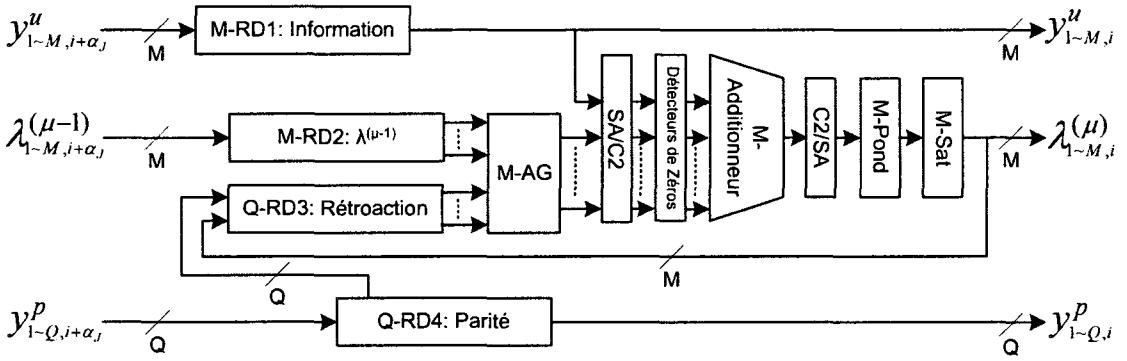

Figure 4.4: Schéma bloc du décodeur à seuil des codes M-CDO

4.4.1 Les registres à décalage

Les différents registres à décalage du décodeur sont implémentés en utilisant le registre à décalage élémentaire que nous avons développé et présenté à la section 3.2.2. Pour ce fait, il suffit de connaître les dimensions et la structure de chacun des registres à décalage. Rappelons que α_J, donné par (4.9), représente la longueur de mémoire qui correspond au plus long registre à décalage parmi les M registres à décalage d'un encodeur M-CDO. Dans l'architecture du décodeur à seuil montrée à la figure 4.4, quatre registres à décalages sont utilisés. La description de ces registres est présentée dans les prochaines sous sections.

4.4.1.1 Registre à décalage d'information

Le registre à décalage d'information (M-RD1) est formé de M registres à décalage élémentaires de longueur α_J et de largeur 3 bits chacun suite à l'utilisation d'une quantification douce sur 3 bits. Un exemple de 3-RD1 est illustré à la figure 4.5.

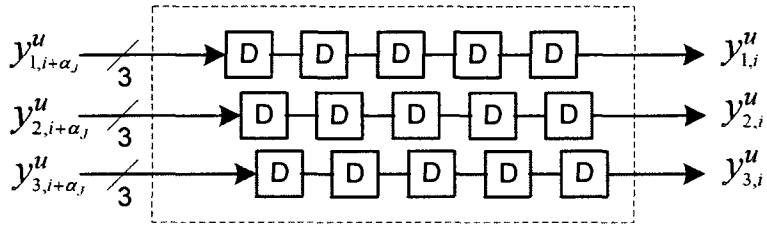

Figure 4.5: Structure de 3-RD1, $M = 3$ et $\alpha_J = 5$

4.4.1.2 Registre à décalage de $\lambda^{(\mu-1)}$

Le registre à décalage de $\lambda^{(\mu-1)}$ (M-RD2) consiste en M registres à décalage de largeur égale à la résolution interne du décodeur, $RInt$, qui conservent les M sorties de l'itération précédente. M-RD2 s'occupe de fournir à l'opérateur Addmin Global (M-AG) tous les termes impliquant les différences simples positives nécessaires afin de calculer les différentes équations de parités qui correspondent à chacune des sorties $\lambda^{(\mu)}_{m,i}$. Les termes impliquant les différences simples positives sont regroupés dans un vecteur $\vec{S}^{+}_{nm,q}$.

Ainsi, le vecteur $\vec{S}^{+}_{nm,q}$ représente les termes $\lambda^{(\mu-1)}_{n,i+(\alpha_{m,q,j}-\alpha_{n,q,k})}$, $(\alpha_{m,q,j} - \alpha_{n,q,k}) \geq 0$, qui contribuent au calcul des équations de parité de la sortie $\lambda^{(\mu)}_{m,i}$. Pour implémenter le registre à décalage qui correspond à $\lambda^{(\mu-1)}_{n}$, la première étape consiste à identifier les vecteurs $\vec{S}^{+}_{nm}$ sur la matrice génératrice $\boldsymbol{\mathcal{A}}$ du code M-CDO. Un exemple est montré à la figure 4.6 pour $n = 2$.

À la figure 4.6, les éléments de chaque vecteur ne sont que les différences simples positives. En effet, $\vec{S}^{+}_{21} = [0, 2, 4]$, $\vec{S}^{+}_{22} = [4]$ et $\vec{S}^{+}_{23} = [1, 2]$. De plus, le numéro encerclé

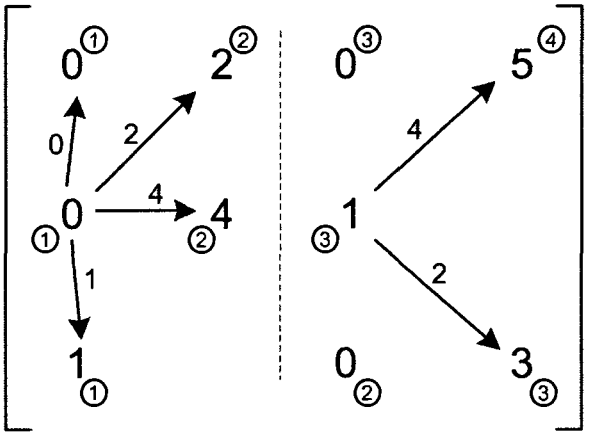

Figure 4.6: Identification des différences simples positives, $\vec{S}^{+}_{2m}, 1 \leq m \leq 3$

à côté de chaque connexion n'est que l'indice j_m de l'équation de parité $\psi^{\mu}_{m,j_m,i}, 1 \leq j_m \leq J_m$.

Ensuite, la deuxième étape consiste à regrouper tous les vecteurs par ordre décroissant de leurs éléments où chaque vecteur est représenté par un triplet (élément;m;j_m) : {(4;2;2), (4;1;4), (2;1;2), (2;3;3), (1;3;1), (0;1;1)}.

Enfin, le registre à décalage de $\lambda_n^{(\mu-1)}$ est implémenté à partir de ses triplets en insérant entre chaque deux éléments consécutifs un registre à décalage élémentaire ayant une longueur équivalente à la différence entre les deux éléments et une largeur $RInt$. À l'entrée de ce registre on trouve $\lambda_{n,i+\alpha_j}^{(\mu-1)}$ et pour chaque élément du triplet (élément;m;j_m) une sortie S_{m,j_m} est créée. Les M registres à décalage qui constituent M-RD2 sont implémentés de la même façon à partir de leurs triplets comme montré à la figure 4.7.

4.4.1.3 Registre à décalage de rétroaction

Le registre à décalage de rétroaction (Q-RD3) est constitué de Q registres à décalage de largeur $RInt$. Il se charge d'effectuer les opérations addmin impliquant les symboles de parité, $y^{p}_{q,i+\alpha_{m,q,j}}$, et la rétroaction sur la décision du décodeur représentée par les

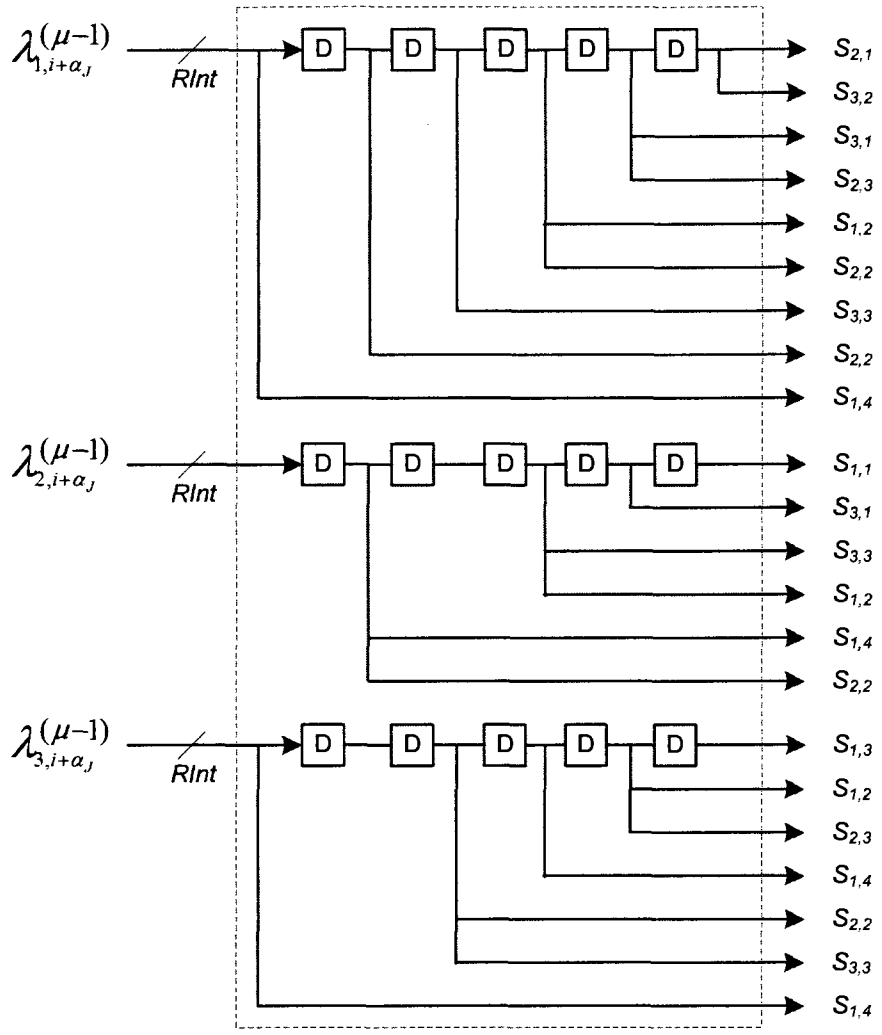

Figure 4.7: Architecture du registre à décalage de $\lambda^{(\mu-1)}$ (3-RD2), $M = 3$ et $\alpha_J = 5$

sorties $\lambda^{(\mu)}_{n,i+(\alpha_{m,q,j}-\alpha_{n,q,k})}$ qui correspondent aux différences simples négatives, $(\alpha_{m,q,j} - \alpha_{n,q,k}) < 0$.

L'implémentation de chaque registre à décalage dans Q-RD3 se fait d'une façon similaire à celle de RD3 du DSI des codes CDO (section 3.2.2). Cependant, dans ce cas nous considérons toutes les connexions $\alpha_{m,q,j}$ de chaque colonne q dans la matrice génératrice $\boldsymbol{\mathcal{A}}$ des codes M-CDO. Soit B_q l'ensemble formé par les connexions ayant des valeurs

distinctes de la colonne q de $\boldsymbol{\mathcal{A}}$:

$$B_q = \bigcup_m A_{m,q} \setminus \bigcap_m A_{m,q} \setminus \{\alpha_{m,q,j} < 0\} = \{\beta_{q,h}, h = 1, \cdots, H_q\} \tag{4.17}$$

où $V \setminus T$ signifie la différence entre les ensemble V et T, $(\beta_{q,h_1} < \beta_{q,h_2})$ si $h_1 < h_2$, $1 \leq m \leq M$ et $H_q \leq (M \times J_{RA})$. Par exemple, pour la matrice génératrice $\boldsymbol{\mathcal{A}}$ donnée par (4.2) on a : $B_1 = \{0, 1, 2, 4\}$ et $B_2 = \{0, 1, 3, 5\}$. Nous définissons alors les différences $d_{q,l}$ entre les connexions consécutives de l'ensemble B_q par la relation suivante :

$$d_{q,l} = \beta_{q,l+1} - \beta_{q,l}, \quad l = 1, 2, \cdots, H_q - 1 \tag{4.18}$$

À partir de l'ensemble B_q, nous implémentons un registre à décalage de la même façon que nous avons faite pour RD3 à la figure 3.5. Toutefois, nous laissons les entrées de rétroaction des opérateurs Addmin flottantes. Le registre à décalage résultant (q-RD3) pour l'ensemble B_1 $(q = 1)$ ci-haut est montré à la figure 4.8 où les C_h représentent ses sorties.

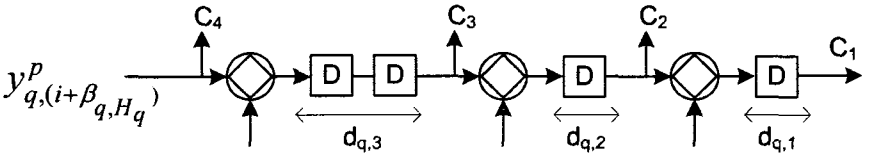

Figure 4.8: Implémentation de q-RD3 à partir de B_q, $\beta_{q,H_q} = 4$

Maintenant, afin d'assigner les entrées et les sorties de q-RD3, nous allons former à partir des éléments de la colonne q de $\boldsymbol{\mathcal{A}}$ un ensemble de triplets $(\alpha_{m,q,j}; m; j_m)$ où j_m représente le numéro de l'équation de parité qui correspond à $\alpha_{m,q,j}$ (figure 4.6). Soit, pour $q = 1$, l'ensemble: {(0,2,1), (0,1,1), (1,3,1), (2;1;2), (4;2;2)}. Il faut alors distinguer deux cas :

1. *La valeur de $\alpha_{m,q,j}$ du triplet $(\alpha_{m,q,j}; m; j_m)$ est unique* :

   Soit h tel que $\beta_{q,h} = \alpha_{m,q,j}$, alors C_h est conectée à une sortie S_{m,j_m} et, si $h > 1$

l'opérateur Addmin qui suit directement C_h reçoit $\lambda_{m,i}^{(\mu)}$ à son entrée de rétroaction.

2. *Plusieurs triplets ont la même valeur de* $\alpha_{m,q,j}$:

   Soit x le nombre de triplets tels que, $\alpha_{m_1,q,j_1} = \cdots = \alpha_{m_x,q,j_x} = \beta_{q,h}$, alors C_h est connectée aux sorties $S_{m_1,j_{m1}} \cdots S_{m_x,j_{mx}}$. De plus, dans le cas où $h > 1$, l'opérateur Addmin à 2 entrées qui suit directement C_h est remplacé par un autre à $(x+1)$ entrées sur lesquelles les $\lambda_{m_1,i}^{(\mu)} \cdots \lambda_{m_x,i}^{(\mu)}$ sont connectées. Un exemple est montré à la figure 4.9 pour $x = 2$.

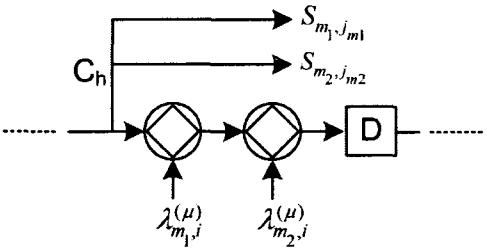

Figure 4.9: Implémentation de q-RD3 : 2 triplets ont la même valeur de $\alpha_{m,q,j}$

Les Q registres à décalage de Q-RD3 sont implémentés de la même façon. Ainsi, l'architecture finale de 2-RD3 du décodeur à seuil de codes 3-CDO en question est présentée à la figure 4.10. Il importe de noter que les symboles y_q^p ayant initialement une résolution de 3 bits à la sortie du registre à décalage de parité (Q-RD4) sont étendues de $(RInt - 3)$ bits à l'entrée de Q-RD3. D'autre part, aucune équation de parité n'est obtenue directement aux sorties de Q-RD3 comme c'était le cas pour RD3 du décodeur à seuil des codes CDO.

4.4.1.4 Registre à décalage de parité

Le registre à décalage de parité (Q-RD4) est formé de Q registres à décalage élémentaires de longueur α_J et de largeur 3 bits chacun. Q-RD4 fournit les symboles $y_{q,i}^p$ à l'itération

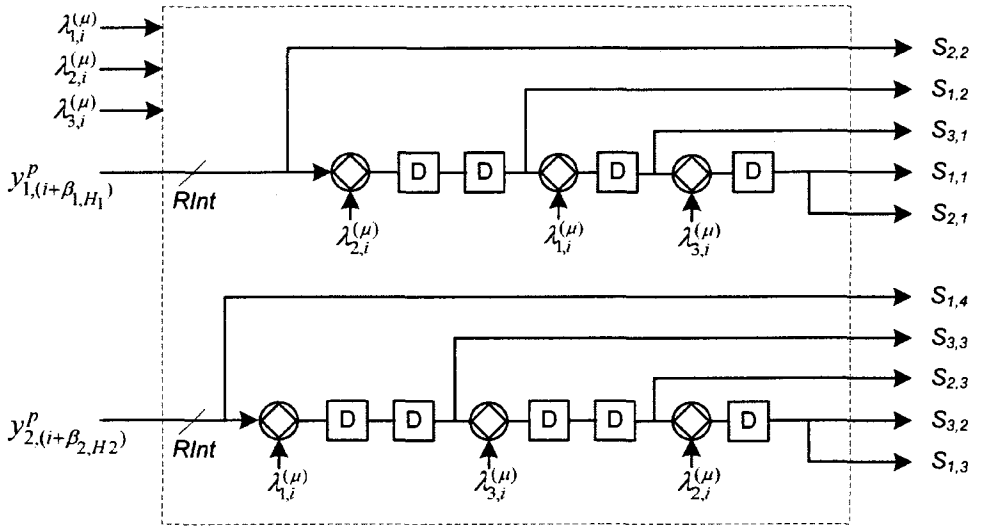

Figure 4.10: Architecture de 2-RD3, $Q = 2$, $\beta_{1,H_1} = 4$ et $\beta_{2,H_2} = 5$

suivante ainsi que les symboles $y^p_{q,(i+\beta_{q,H_q})}$ à Q-RD3. Un exemple de 2-RD4 est illustré à la figure 4.11.

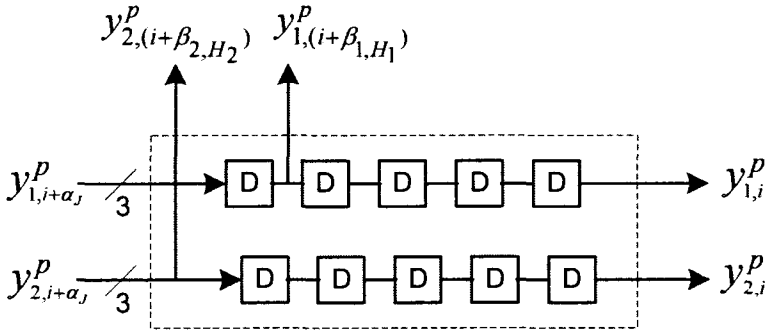

Figure 4.11: Registre à décalage de parité (2-RD4), $Q = 2$, $\alpha_J = 5$, $\beta_{1,H_1} = 4$ et $\beta_{2,H_2} = 5$

4.4.2 Le noyau de logique combinatoire

Le noyau de logique combinatoire est formé principalement de l'opérateur Addmin global (M-AG), de l'additionneur (M-Additionneur), du pondérateur (M-Pond) et du saturateur (M-Sat), ainsi que des opérateurs de conversion binaire et de détecteurs de

zéros (figure 4.4). Les entrées du noyau de logique combinatoire sont toutes fournies par les registres à décalage M-RD1, M-RD2 et Q-RD3.

À la sortie de M-RD2 et Q-RD3, on trouve toutes les entrées du M-AG. Celui-ci est formé de M opérateurs AG_m qui fournissent chacun les J_m équations de parité, $\psi^{\mu}_{m,j_m,i}$, nécessaires pour calculer les sorties $\lambda^{(\mu)}_{m,i}$. Soit z le nombre de sorties S_{m,j_m} de M-RD2 et Q-RD3. L'équation de parité $\psi^{\mu}_{m,j_m,i}$ est calculée à la sortie d'un opérateur Addmin ayant z entrées qui combine toutes les sorties S_{m,j_m}. Il importe de noter que le plus grand opérateur Addmin dans M-AG a un nombre d'entrées équivalent au nombre de connexions de la colonne la plus dense de $\boldsymbol{\mathcal{A}}$, c'est à dire celle qui contient le plus de connexions. Par exemple en se référant aux figures 4.7 et 4.10, nous pouvons compter 5 sorties $S_{1,4}$ dans M-RD2 et Q-RD3. $\psi^{\mu}_{1,4,i}$ est calculée en utilisant un opérateur Addmin à 5 entrées où les 5 "$S_{1,4}$" sont connectées.

Comme montré à la figure 4.4, après conversion et détection de zéros négatifqs à la sortie de l'opérateur M-AG, les J_m équations de parité de chaque symbole d'information à décoder sont additionnées au symbole d'information $y^{u}_{m,i}$ reçu du canal. Cette opération est réalisée par le bloc "M-Additionneur" indiqué à la figure 4.4 qui est formé de M additionneurs ayant $(J_m + 1)$ entrées. Les M sorties de M-Additionneur sont appliquées à des opérateurs de conversion binaire, puis pondérées (M-Pond) et enfin saturées (M-Sat) afin de générer les M sorties de l'itération, $\lambda^{(\mu)}_{m,i}, 1 \leq m \leq M$.

4.5 Pipelinage du décodeur à seuil des codes M-CDO

Le débit du décodeur à seuil des codes M-CDO peut être augmenté en se servant de la même technique de pipelinage utilisée pour le décodeur à seuil des codes CDO (section 3.3). Cependant, dans ce cas la technique consiste à resynchroniser simultanément les Q registres à décalage de Q-RD3. Toutefois, le nombre d'emplacements des étages de

pipeline ne se calcule pas de la même façon qu'auparavant, de sorte que l'équation de la capacité de pipelinage a été légèrement modifiée comme montré dans les sous-sections suivantes.

4.5.1 Nombre d'emplacements des étages de pipeline

Le nombre d'emplacements des étages de pipeline du décodeur à seuil de codes M-CDO, Nb_{EM}, est donné par la relation suivante :

$$Nb_{EM} = m' + n' + 4 \tag{4.19}$$

où m' est le nombre de nivaux du plus grand opérateur Addmin dans M-AG et n' correspond au nombre de nivaux du plus grand additionneur dans M-Additionneur.

Soit H_{max} le nombre de connexions de la colonne la plus dense de $\mathcal{A}$, alors m' est donné par :

$$m' = \lceil log_2(H_{max}) \rceil \tag{4.20}$$

Sachant que les M additionneurs ont chacun $(J_m + 1)$ entrées, $(1 \leq m \leq M)$, n' est donné par la relation suivante :

$$n' = \lceil log_2\big(\max_{1 \leq m \leq M} (J_m) + 1\big) \rceil \tag{4.21}$$

4.5.2 Capacité de pipelinage d'un générateur de codes M-CDO

La capacité de pipelinage d'un générateur de codes M-CDO, P_{cM}, est déduite également à partir des différences qui existent entre les connexions consécutives dans le registre à décalage (Q-RD3). Cependant, toutes les différences $d_{q,l}$ (4.18) des ensembles B_q (4.17) doivent être considérées. D'autre part, la présence des opérateurs Addmin ayant plus de 2 entrées à l'intérieur de Q-RD3 n'est pas permise. Par conséquent, au cours de la recherche des codes M-CDO, il a été imposé que les générateurs de codes M-CDO ne puissent pas avoir plusieurs connexions dans une colonne ayant la même valeur de β_h si $h > 1$ (voir la section 4.4.1). La présence de ces opérateurs compromettrait et limiterait les augmentations du débit du décodeur introduites par la technique de pipelinage. En effet, comme montrés à la figure 4.9, à l'intérieur des registres Q-RD3, des sous-chemins critiques difficiles à manipuler seraient ainsi créés.

Une autre considération dont il faut tenir compte réside dans le fait qu'aucune équation de parité n'est générée directement à la sortie de Q-RD3. Les étages de pipeline insérés au niveau du M-AG ne retournent donc pas les bascules, initialement déplacées de $d_{q,1}$, à leurs places d'origines comme c'était le cas pour le décodeur des codes CDO. Cependant, les bascules servent au pipelinage des opérateurs Addmin qui calculent $\psi_{m,1,i}$ à l'intérieur du M-AG. En conséquence et par analogie avec (3.13), la capacité de pipelinage, P_{cM}, est obtenue en utilisant l'équation suivante :

$$P_{cM} = min(d_{q,l}) - 1, \quad 1 \leq q \leq Q, \quad 1 \leq l \leq H_q \tag{4.22}$$

4.6 Recherche des meilleurs générateurs de codes M-CDO

La recherche des meilleurs générateurs de codes M-CDO et M-CPDO peut être effectuée pour couvrir une large gamme de taux de codage R ainsi qu'une grande variété

du nombre minimal d'équations de parité J. Les techniques de recherches des codes utilisées sont basées sur celles décrites dans [30]. Il est nécessaire de souligner qu'une telle recherche dépasse largement le cadre de ce projet. Cependant, une partie de cette recherche a été effectuée et nous présentons dans les tableaux qui suivent un nombre limité de générateurs de codes M-CDO et M-CPDO afin de valider nos résultats. Nous nous limitons aux deux cas où $Q = M$ ($R = M/2M = 1/2$) et $Q = 1$ ($R = M/(M+1)$). De plus, seuls les générateurs qui offrent une protection égales des erreurs, EEP, sont considérés.

4.7 Conclusion

Dans ce chapitre, les codes convolutionnels doublement orthogonaux à multi-registres à décalage, M-CDO, ont été introduits. Une nouvelle simplification a été appliquée sur les codes M-CDO permettant de réduire significativement la longueur de mémoire de l'encodeur. Par exemple, le Span du meilleur générateur des codes 2-CDO trouvé, $J = 10$, a passé de 577 à 145 en appliquant cette nouvelle simplification. L'architecture du décodeur à seuil des codes M-CDO a été présentée. Cette architecture qui est en fait une version parallèle de l'architecture du décodeur des codes CDO, permet de franchir un très haut débit sans faire exploser la complexité du décodeur, ce qui est démontré par les résultats expérimentaux de différentes architectures qui sont présentées dans le chapitre suivant.

Tableau 4.1: Ensemble de meilleurs connexions $\alpha_{m,q,j} \in \mathcal{A}$ pour les codes M-CDO où $M = Q$, $R = 1/2$: Nombre minimal des équations de parité J, nombre de registres à décalage M, Span α_J, Capacité de pipelinage P_{cM}, Facteur de simplification δ_{max} et Ensemble des connexions $[\alpha_{m,q,j}]$

J	M	α_J	P_{cM}	δ_{max}	Ensemble des connexions $[\alpha_{m,q,j}]$
6	2	71	10	0.515	$\left[\begin{array}{ccc\|ccc} 0 & 26 & 71 & 11 & 23 & 71 \\ 14 & 38 & 56 & 0 & 39 & 52 \end{array}\right]$
	3	67	10	0.407	$\left[\begin{array}{cc\|cc\|cc} 0 & 33 & 22 & 60 & 0 & 37 \\ 11 & 65 & 11 & 46 & 12 & 51 \\ 22 & 51 & 0 & 34 & 24 & 67 \end{array}\right]$
8	2	256	11	0.51	$\left[\begin{array}{cccc\|cccc} 0 & 13 & 43 & 234 & 12 & 41 & 145 & 253 \\ 0 & 25 & 79 & 256 & 0 & 26 & 86 & 204 \end{array}\right]$
	4	173	12	0.491	$\left[\begin{array}{cc\|cc\|cc\|cc} 0 & 57 & 39 & 166 & 0 & 94 & 42 & 173 \\ 13 & 74 & 26 & 145 & 14 & 147 & 28 & 74 \\ 26 & 91 & 13 & 125 & 28 & 117 & 14 & 93 \\ 39 & 144 & 0 & 108 & 42 & 164 & 0 & 143 \end{array}\right]$
9	2	383	12	0.561	$\left[\begin{array}{ccccc\|ccccc} 0 & 26 & 63 & 208 & 376 & 0 & 30 & 157 & 361 & -1 \\ 13 & 41 & 112 & 314 & -1 & 14 & 46 & 96 & 296 & 383 \end{array}\right]$
	3	359	12	0.457	$\left[\begin{array}{ccc\|ccc\|ccc} 0 & 39 & 255 & 26 & 73 & 191 & 0 & 50 & 359 \\ 13 & 55 & 358 & 13 & 59 & 354 & 14 & 126 & 343 \\ 26 & 71 & 133 & 0 & 40 & 311 & 28 & 151 & 321 \end{array}\right]$
10	2	577	12	0.583	$\left[\begin{array}{ccccc\|ccccc} 0 & 13 & 61 & 192 & 540 & 13 & 41 & 106 & 420 & 443 \\ 31 & 45 & 82 & 161 & 577 & 0 & 26 & 83 & 294 & 575 \end{array}\right]$
	5	462	12	0.43	$\left[\begin{array}{cc\|cc\|cc\|cc\|cc} 0 & 45 & 26 & 222 & 0 & 175 & 59 & 328 & 0 & 459 \\ 13 & 79 & 0 & 208 & 14 & 212 & 44 & 399 & 17 & 353 \\ 27 & 98 & 0 & 154 & 29 & 395 & 30 & 462 & 35 & 280 \\ 0 & 120 & 13 & 94 & 43 & 432 & 15 & 246 & 52 & 399 \\ 0 & 144 & 0 & 67 & 57 & 457 & 0 & 438 & 69 & 421 \end{array}\right]$

Tableau 4.2: Ensemble de meilleurs connexions $\alpha_{m,q,j} \in \mathcal{A}$ pour les codes M-CPDO où $M = Q$, $R = 1/2$: Nombre minimal des équations de parité J, nombre de registres à décalage M, Span α_J, Capacité de pipelinage P_{cM}, Facteur de simplification δ_{max} et Ensemble des connexions $[\alpha_{m,q,j}]$

J	M	α_J	P_{cM}	δ_{max}	Ensemble des connexions $[\alpha_{m,q,j}]$
8	2	100	11	0.765	$\left[\begin{array}{cccc\|cccc} 0 & 13 & 42 & 77 & 12 & 40 & 72 & 84 \\ 0 & 25 & 55 & 92 & 0 & 26 & 57 & 100 \end{array}\right]$
	4	97	11	0.691	$\left[\begin{array}{cc\|cc\|cc\|cc} 0 & 14 & 36 & 92 & 0 & 49 & 39 & 83 \\ 0 & 27 & 24 & 77 & 12 & 66 & 26 & 97 \\ 0 & 40 & 12 & 63 & 24 & 80 & 13 & 71 \\ 0 & 53 & 0 & 49 & 36 & 94 & 0 & 55 \end{array}\right]$
9	3	107	12	0.78	$\left[\begin{array}{ccc\|ccc\|ccc} 0 & 27 & 73 & 13 & 57 & 106 & 0 & 28 & 89 \\ 13 & 41 & 89 & 0 & 42 & 87 & 0 & 43 & 107 \\ 0 & 57 & 107 & 0 & 27 & 73 & 13 & 58 & 71 \end{array}\right]$
10	2	145	12	0.864	$\left[\begin{array}{ccccc\|ccccc} 0 & 14 & 45 & 80 & 121 & 13 & 43 & 77 & 92 & 144 \\ 0 & 27 & 59 & 96 & 145 & 0 & 28 & 61 & 111 & 127 \end{array}\right]$
	5	128	12	0.803	$\left[\begin{array}{cc\|cc\|cc\|cc\|cc} 0 & 65 & 52 & 126 & 0 & 70 & 56 & 115 & 0 & 86 \\ 13 & 81 & 39 & 111 & 14 & 97 & 42 & 86 & 15 & 112 \\ 26 & 97 & 26 & 96 & 28 & 127 & 28 & 128 & 30 & 73 \\ 39 & 125 & 13 & 81 & 42 & 111 & 14 & 102 & 45 & 99 \\ 52 & 110 & 0 & 65 & 56 & 84 & 0 & 72 & 60 & 126 \end{array}\right]$

Tableau 4.3: Ensemble de meilleurs connexions $\alpha_{m,q,j} \in \boldsymbol{\mathcal{A}}$ pour les codes M-CDO où $Q = 1$, $M = 2$, $R = M/(M+1) = 2/3$: Nombre minimal des équations de parité J, Span α_J, Capacité de pipelinage P_{cM}, Facteur de simplification δ_{max} et Ensemble des connexions $[\alpha_{m,q,j}]$

J	α_J	P_{cM}	δ_{max}	Ensemble des connexions $[\alpha_{m,q,j}]$
3	49	10	0.292	$\begin{bmatrix} 0 & 11 & 35 \\ 0 & 23 & 49 \end{bmatrix}$
4	85	11	0.438	$\begin{bmatrix} 0 & 12 & 38 & 67 \\ 0 & 25 & 53 & 85 \end{bmatrix}$
5	132	11	0.585	$\begin{bmatrix} 0 & 12 & 38 & 96 & 132 \\ 0 & 25 & 55 & 78 & 112 \end{bmatrix}$
6	296	11	0.597	$\begin{bmatrix} 0 & 12 & 38 & 67 & 142 & 232 \\ 0 & 25 & 53 & 85 & 263 & 296 \end{bmatrix}$
7	613	11	0.606	$\begin{bmatrix} 0 & 12 & 38 & 67 & 112 & 295 & 579 \\ 0 & 25 & 53 & 85 & 189 & 426 & 613 \end{bmatrix}$
8	985	12	0.631	$\begin{bmatrix} 0 & 38 & 65 & 95 & 136 & 284 & 646 & 870 \\ 25 & 51 & 80 & 114 & 227 & 465 & 965 & 985 \end{bmatrix}$
9	1746	13	0.641	$\begin{bmatrix} 0 & 28 & 59 & 94 & 140 & 316 & 530 & 1057 & 1690 \\ 14 & 43 & 75 & 114 & 252 & 386 & 765 & 1223 & 1746 \end{bmatrix}$

CHAPITRE 5

RÉSULTATS EXPÉRIMENTAUX

5.1 Introduction

Après avoir présenté dans les chapitres précédents l'architecture du DSI à haut débit de codes CDO et PCDO à taux compatibles (1 registre à décalage) ainsi que l'architecture parallèle du DSI à haut débit de codes M-CDO, nous rapportons dans ce chapitre les résultats expérimentaux de l'implémentation sur FPGA de ces deux architectures. Le débit, la complexité et les performances d'erreur (BER : Bit Error Rate) du DSI sont exposés. Les données sont générés en considérant le cas d'une modulation BPSK sur un canal à bruit Gaussien blanc additif (AWGN). Une comparaison entre les résultats expérimentaux de générateurs de codes CDO est également illustrée.

Pour chacun des codes CDO, PCDO et M-CDO, une version matérielle du DSI a été codée en VHDL. Afin de vérifier le fonctionnement de la version matérielle, une autre version à "virgule fixe" (quantifié) a été codée sur Matlab. Ainsi, en utilisant le logiciel de modélisation "Modelsim", les résultats de la version matérielle ont été comparés à ceux de la version Matlab. Ensuite, le DSI a été prototypé en utilisant la plateforme de prototypage rapide "ARM Integrator" et une version adaptée de l'environnement d'évaluation des performances d'erreur développé dans [36]. Le design du DSI est alors validé en comparant les résultats expérimentaux aux limites théoriques et à des résultats obtenus par simulations logicielles sur ordinateur et effectués dans d'autres travaux de recherche [29,30]. La fréquence d'opération et la complexité de différents modules sont tirés des rapports post placement et routage fournis par l'outil ISE 9.2.04i en visant le FPGA Virtex-II Pro (XC2VP70-7) de Xilinx.

5.2 Notations utilisées

Nous présentons à cette section quelques notations pertinentes à la présentation des résultats expérimentaux.

1- *Abréviation de générateurs des codes* :
Les générateurs des codes CDO et PCDO sont désormais dénotés par $\{J, \alpha_J\}$, tandis que les générateurs des codes M-CDO et M-CPDO sont dénotés par $\{R, J, \alpha_J\}$. Par exemple, le générateur des codes PCDO à taux compatibles spécifié par $J = 10$ et l'ensemble de connexions $A = \{0, 9, 34, 107, 127, 208, 311, 330, 353, 366\}$ dans le tableau 3.2 est dénoté par $\{J = 10, \alpha_J = 366\}$.

2- *Débit par itération, D_{it}* :
Le débit par itération, D_{it}, est le débit d'information à la sortie d'un décodeur élémentaire formé de $N = 1$ itération. D_{it} est donné par la relation suivante :

$$D_{it} = M \times f \text{ Mbps (Mega bits par seconde)} \tag{5.1}$$

où f (MHz) est la fréquence d'opération du décodeur et $M = 1$ pour un décodeur des codes CDO ou PCDO.

Le débit d'un DSI, D, peut être facilement déduit de D_{it}. En effet, un DSI de N itérations est constitué, généralement, de "N_{imp}" itérations implémentées et réitérées[1] N/N_{imp} fois afin d'atteindre les performances de N itérations voulues. Alors, D est donné par :

$$D = \frac{N_{imp}}{N} \times D_{it} \tag{5.2}$$

[1]Ce type d'implémentation n'est pas couvert dans ce mémoire, mais plutôt N_{imp} est toujours considéré égal à N

3- *Rapport débit sur complexité, RDC_{it} :*

Le rapport débit sur complexité, RDC_{it}, désigne le débit fourni par chaque unité de complexité, C_{it}, d'une itération. C_{it} est donnée par le nombre de portes, en "kG" (kGates), utilisés pour implémenter le décodeur. En d'autres termes, c'est l'efficacité d'une implémentation du point de vue du débit. RDC_{it} est donné par :

$$RDC_{it} = D_{it}/C_{it} \text{ (Mbps/kG)} \tag{5.3}$$

4- *BER à meilleurs CPs :*

Dans le cadre de ce projet, un coefficient de pondération, CP uniforme est utilisé pour les N itérations d'un DSI. La figure 5.1 montre les BERs de deux versions ($RInt$ = 4 et 5 bits) du DSI des codes CDO $\{J = 10, \alpha_J = 366\}$, $R = 1/2$, à la $8^{ième}$ itération en fonction de CP. Deux rapports E_b/N_0 = 2.5 et 3.5 dB sont considérés.

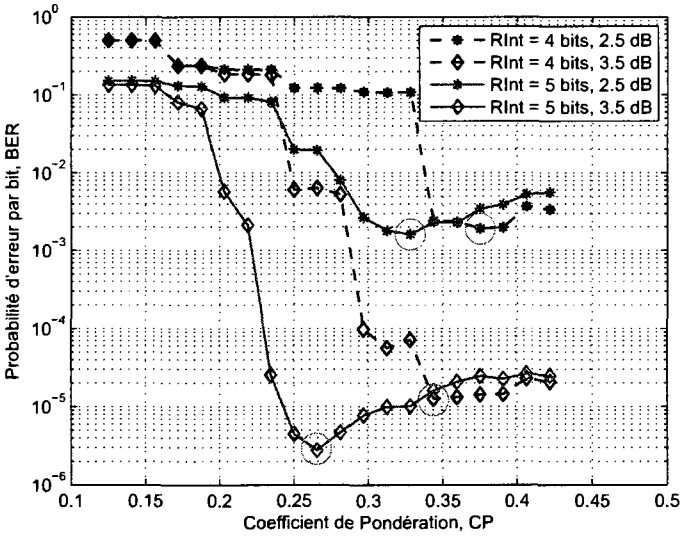

Figure 5.1: BER en fonction de CP, Codes CDO $\{J = 10, \alpha_J = 366\}$, $R = 1/2$, 8 itérations

On remarque que la valeur de CP qui minimise le BER du DSI dépend de E_b/N_0 et de $RInt$. Cependant, afin de bien comparer les performances d'erreurs de plusieurs DSIs,

il importe de noter le BER minimal que chaque DSI peut atteindre à un rapport E_b/N_0 donné. Les courbes de "BER à meilleurs CPs" sont utilisées où seulement les meilleures performances d'erreurs sont considérées à chaque niveau de E_b/N_0 tel que montré à la figure 5.2 pour les mêmes versions du DSI considérées ci-dessus.

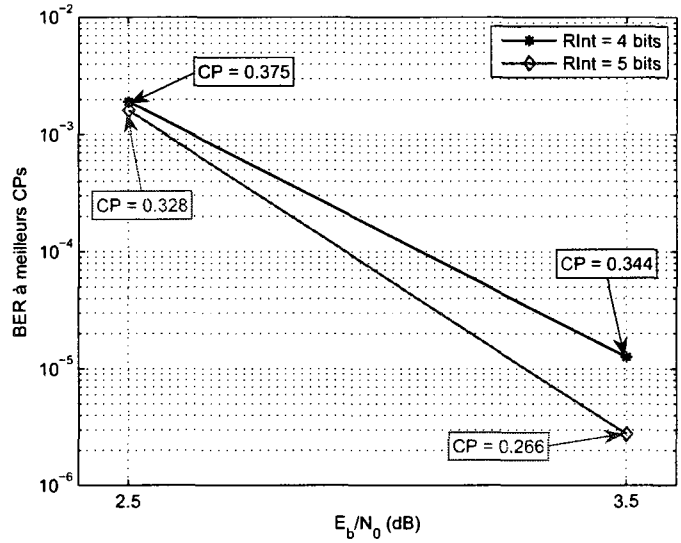

Figure 5.2: BER à meilleurs CPs, Codes CDO $\{J = 10, \alpha_J = 366\}$, $R = 1/2$, 8 itérations

5.3 Comparaison des délais des deux architectures du pondérateur

Dans cette section, les délais engendrés par les deux architectures du pondérateur sont étudiés. Soient l'architecture complément à deux (C2) utilisée dans [36] et l'architecture signe-amplitude (SA) proposée dans ce mémoire (section 3.2.1). Le tableau 5.1 compare les délais qui correspondent aux deux architectures du pondérateur. Ces valeurs ont été prises en fixant RS_{ADD} et R_{CP} à 9 et 6 bits respectivement. On trouve également dans le même tableau les valeurs de la fréquence maximale atteignable dans le cas où le chemin critique du décodeur correspond au délai du pondérateur, f'_{max}.

Suite à l'utilisation de l'architecture SA, le délai engendré par le pondérateur a été

Tableau 5.1: Comparaison de délais des deux architectures du pondérateur

Architecture	Délai (ns)	f'_{max} (MHz)
C2	5.520	181.16
SA	3.659	273.30

réduit de 5.520 ns à 3.659 ns, soit 33.7% de moins. L'importance de cette amélioration réside dans le fait qu'un pondérateur ayant une architecture C2 va former un goulot d'étranglement qui limite la fréquence maximale atteignable du décodeur à $f'_{max} = 181.16 << f_{max} \simeq 333$ MHz où f_{max} est donnée par (3.3). D'ailleurs, l'architecture SA permet au décodeur de fonctionner à une fréquence plus élevée et d'atteindre 273.30 MHz dans le cas où RS_{ADD} et R_{CP} sont égales à 9 et 6 bits respectivement.

Un autre facteur qui peut affecter le délai du pondérateur est la valeur de la résolution binaire du coefficient de pondération, R_{CP}. Il est intéressant d'utiliser une grande résolution R_{CP} afin de raffiner la recherche de la meilleure valeur du CP. D'autre part, une grande résolution R_{CP} contribue à augmenter le délai du pondérateur. Le tableau 5.2 fournit les délais du pondérateur ayant une architecture SA pour les trois cas où R_{CP} est égal à 4, 6 et 8 bits respectivement. RS_{ADD} est toujours fixé à 9 bits.

Tableau 5.2: Délai du pondérateur (SA) pour R_{CP} = 4, 6 et 8 bits, RS_{ADD} = 9 bits

R_{CP}	Délai (ns)	f'_{max} (MHz)
4	3.531	283.21
6	3.659	273.30
8	3.767	265.46

On observe clairement que le délai engendré par le pondérateur croit avec R_{CP}. Dans le cadre de ce projet, une résolution de $R_{CP} = 6$ bits a été adoptée. Cependant, si la meilleure valeur du CP peut être représentée en utilisant un nombre de bits < 6, il reste toujours possible de réduire R_{CP} afin de réduire davantage le délai engendré.

5.4 Choix de la résolution interne du décodeur

La résolution interne, $RInt$, du décodeur influence son débit, sa complexité et ses performances d'erreurs. Ainsi, le choix de $RInt$ doit servir à augmenter le débit et réduire la complexité, sans dégrader les performances d'erreurs du décodeur. Pour ce faire, quatre versions non-pipelinées du DSI des codes PCDO $\{J = 10, \alpha_J = 366\}$ ont été étudiées pour $RInt$ = 4, 5, 6 et 9 bits. Les différents paramètres (C_{it}, D_{it}, RDC_{it}...) qui correspondent à chacune des quatre versions sont listés dans le tableau 5.3.

Tableau 5.3: Comparaison de la complexité et du débit par itération du décodeur à seuil des codes PCDO $\{J = 10, \alpha_J = 366\}$ pour $RInt$ = 4, 5, 6 et 9 bits

$RInt$	Complexité par itération					Délai critique (ns)	D_{it} (Mbps)	RDC_{it} (Mbps/kG)
	LUT	FF	MP*	Slices	C_{it} (kG)			
4	976	76	1	743	33.875	21.766	45.943	1.36
5	1257	95	1	932	39.596	23.952	41.750	1.05
6	1464	114	1	1085	45.967	24.961	40.062	0.87
9	2173	171	1	1563	63.291	27.278	36.660	0.58

* MULT18x18

À partir de ces résultats, nous constatons que la réduction de $RInt$ avantage une implémentation plus rapide et moins complexe du décodeur. Cependant, il faut tenir compte de la dégradation des performances d'erreurs qu'entraîne une telle réduction. Les BERs à meilleurs CPs à la $8^{ième}$ itération des quatre versions du DSI sont illustrées à la figure 5.3.

On observe à la figure 5.3 que pour $RInt$ = 5, 6 et 9 bits, les courbes de BERs sont proches l'une de l'autre, tandis que pour $RInt$ = 4 bits, les performances d'erreurs de DSI sont trop dégradées. Une résolution interne de 5 bits est donc suffisante afin d'implémenter un DSI rapide et performant. En outre, les résultats expérimentaux de DSI des codes M-CDO ont montré aussi qu'une résolution interne de 5 bits est suffisante pour que le décodeur atteigne ses meilleures performances d'erreurs. Par conséquent,

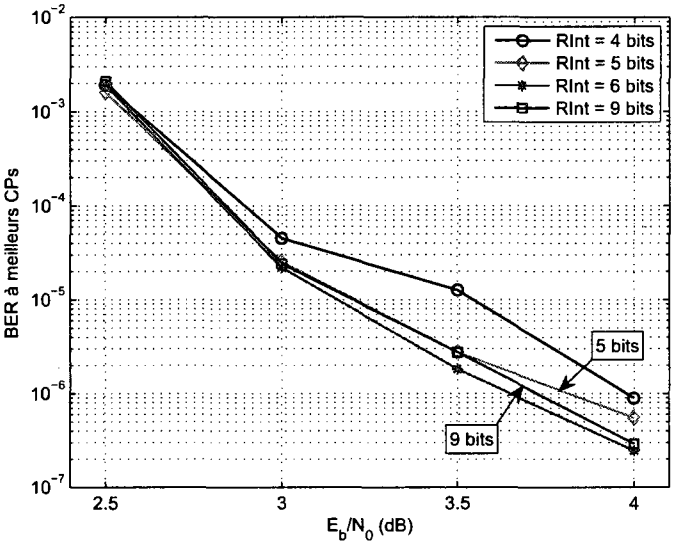

Figure 5.3: BER à meilleurs CPs, PCDO à taux compatibles $\{J = 10, \alpha_J = 366\}$, $R = 1/2$, $RInt$ = 4, 5, 6 et 9 bits, 8 itérations

dans le reste de ce mémoire, tous les résultats expérimentaux sont générés en fixant la résolution interne $RInt$ de DSIs à 5 bits.

5.5 Pipelinage du décodeur à seuil

La technique de pipelinage du décodeur à seuil a permis de réduire le délai de son chemin critique, ce qui a conduit à une augmentation de la fréquence d'opération et ainsi que du débit. Dans cette section, les effets de l'application de cette technique sont présentés. Le tableau 5.4 liste les résultats post placement et routage du décodeur à seuil des codes PCDO $\{J = 10, \alpha_J = 366\}$ avant et après l'application de la technique de pipelinage.

On observe que l'application de la technique de pipelinage a augmenté considérablement le débit par itération, D_{it}, du décodeur de 41.75 à 196.19 Mbps. Soit une amélioration de 78.7 %. Le coût de cette augmentation de débit était de l'ordre de 5.486 kG seulement. De plus, le rapport débit sur complexité, RDC_{it}, a augmenté de 1.05 à 4.35 Mbps/kG.

Tableau 5.4: Comparaison de la complexité et du débit par itération du décodeur à seuil des codes PCDO $\{J = 10, \alpha_J = 366\}$ avant et après l'application de la technique de pipelinage, $RInt = 5$ bits

Étages de pipeline	Complexité par itération					Délai critique (ns)	D_{it} (Mbps)	RDC_{it} (Mbps/kG)
	LUT	FF	MP*	Slices	C_{it} (kG)			
0	1257	95	1	932	39.596	23.952	41.75	1.05
12	1158	532	1	937	45.082	5.097	196.19	4.35

* MULT18x18

Ainsi, on constate que l'implémentation pipelinée du décodeur est 4.14 fois plus efficace du point de vue de débit que l'implémentation non-pipelinée.

5.5.1 Influence de l'architecture du registre à décalage élémentaire

Les résultats du tableau 5.4 sont générés en utilisant une architecture du registre à décalage élémentaire semblable à celle développée dans [39] (sans insertion du FF). Afin d'analyser le chemin critique du décodeur pipeliné, un extrait du rapport de synchronisme (timing) post placement et routage est montré à la figure 5.4.

```
Maximum Data Path: U_sr_feedback/do_index.2.srlreg_pipe/do_last_SRL16.do_last_SRL16_pins.1.U_last_SRL16/SRL1
  Location             Delay type         Delay(ns)  Physical Resource
                                                     Logical Resource(s)
  -------------------------------------------------  -------------------
  SLICE_X145Y41.Y      Treg                  2.794    U_sr_feedback.map_tab_7(1)
                                                      U_sr_feedback/do_index.2.srlreg_pipe/do_last_SRL16.do_l
                                                      U_sr_feedback/do_index.2.srlreg_pipe/do_last_SRL16.do_l
  SLICE_X145Y38.G1     net (fanout=2)        0.308   U_sr_feedback.map_tab_7(1)
  SLICE_X145Y38.Y      Tilo                  0.275   U_sr_feedback.map_tab_8(1)
                                                      g0_2
  SLICE_X145Y39.G2     net (fanout=1)        0.097   g0_2
  SLICE_X145Y39.Y      Tilo                  0.275   U_sr_feedback.map_tab_8(2)
                                                      g0_0
  SLICE_X144Y40.G3     net (fanout=3)        0.227   U_sr_feedback.do_index.2.addmin.do_addmin.un1_amp1
  SLICE_X144Y40.Y      Tilo                  0.275   U_sr_feedback.map_tab_14(0)
                                                      g0
  SLICE_X145Y37.BY     net (fanout=1)        0.507   U_sr_feedback.map_tab_8(0)
  SLICE_X145Y37.CLK    Tds                   0.339   map_fb_addmin(19)
                                                      U_sr_feedback/do_index.2.do_all_stages.srlreg/do_last_S
  -------------------------------------------------  ---------------------------
  Total                                      5.097ns (3.958ns logic, 1.139ns route)
                                                     (77.7% logic, 22.3% route)
```

Figure 5.4: Extrait du rapport de synchronisme post placement et routage, architecture du registre à décalage élémentaire sans FF utilisée

Remarquons que le chemin critique du décodeur s'amorce à la sortie d'un LUT (SLICE X145Y41.Y) qui ajoute 2.794 ns aux délais combinatoires ainsi qu'aux délais de routage qui s'accumulent pour former le délai total du chemin critique.

En utilisant la nouvelle architecture (avec FF) du registre à décalage élémentaire proposée dans ce mémoire, les nouveaux résultats du décodeur sont présentés dans le tableau 5.5.

Tableau 5.5: Comparaison de la complexité et du débit par itération du décodeur à seuil des codes PCDO $\{J = 10, \alpha_J = 366\}$ pour les deux architectures du registre à décalage élémentaire, $RInt = 5$ bits, 12 étages de pipeline insérés

Arch.	Complexité par itération					Délai	D_{it}	RDC_{it}
R.D.E.*	LUT	FF	MP†	Slices	C_{it} (kG)	critique (ns)	(Mbps)	(Mbps/kG)
Sans FF	1158	532	1	937	45.082	5.097	196.19	4.35
Avec FF	1149	801	1	1076 (3%)	46.948	3.760	265.96	5.66

*:Architecture du registre à décalage élémentaire, †:MULT18x18

À partir des résultats du tableau 5.5, il s'ensuit que la nouvelle architecture du registre à décalage élémentaire (avec FF) a permis au décodeur pipeliné d'atteindre un haut débit de 265.96 Mbps pour 1.866 kG seulement de complexité supplémentaire. Soit une augmentation de 69.77 Mbps comparativement au décodeur pipeliné utilisant l'architecture du registre à décalage élémentaire sans FF. De plus, le nouveau rapport RDC_{it} est maintenant 5.66 Mbps/kG, ce qui est 5.39 fois plus efficace que l'implémentation non-pipelinée du décodeur.

La figure 5.5 montre un extrait du nouveau rapport de synchronisme post placement et routage du décodeur pipeliné. Sachant que "Treg" (clock-to-out time of a shift register SRL16) est le temps de transition à la sortie d'un primitif SRL16 (Treg = $t_{coSRL16}$), "net" est le temps de routage et en particulier dans ce cas entre un primitif SRL16 et un FF (net = $t_{rSRL16FF}$) et "Tdick" (internal capture register intrinsic setup time) est le temps de setup à l'entrée de FF (Tdick = t_{suFF}) [33], alors, le délai du chemin critique analysé dans ce rapport n'est que le délai qui existe entre un primitif SRL16 et un FF (deux

éléments de mémoire) sans aucune logique combinatoire entre eux comme montré à la figure 3.4. Il en résulte que la technique de pipelinage appliquée au décodeur à seuil utilisant les nouvelles architectures du registre à décalage élémentaire et du pondérateur a permis d'atteindre la fréquence maximale d'opération et le débit maximal atteignables par la technologie Virtex-II Pro XC2VP70 considérée dans le cadre de ce projet.

```
Data Path: U_sr_feedback/do_index.4.do_all_stages.srlreg/doSR.do_last_SRL16.do_last_SRL16_pins.1
  Delay type         Delay(ns)  Logical Resource(s)
  -------------------------  -------------------
  Treg                  2.794   U_sr_feedback/do_index.4.do_all_stages.srlreg/doSR.do_last_SRL16
                                U_sr_feedback/do_index.4.do_all_stages.srlreg/doSR.do_last_SRL16
  net (fanout=1)        0.758   U_sr_feedback/do_index.4.do_all_stages.srlreg/delayed_5(1)
  Tdick                 0.208   U_sr_feedback/do_index.4.do_all_stages.srlreg/doSR.uOutFF/q[1]
  -------------------------  -------------------------
  Total                 3.760ns (3.002ns logic, 0.758ns route)
                                (79.8% logic, 20.2% route)
```

Figure 5.5: Extrait du rapport de synchronisme post placement et routage, architecture du registre à décalage élémentaire avec FF utilisée

5.5.2 Exemple de pipelinage d'un décodeur à seuil des codes M-CPDO

Pour cet exemple, le décodeur à seuil des codes 5-CPDO $\{R = 5/10,\ J = 10,\ \alpha_J = 128\}$ est considéré. Les résultats de l'application de la technique de pipelinage sont montrés au tableau 5.6.

Tableau 5.6: Comparaison de la complexité et du débit par itération du décodeur à seuil des codes 5-CPDO $\{R = 5/10,\ J = 10,\ \alpha_J = 128\}$ avant et après l'application de la technique de pipelinage, $RInt = 5$ bits

Étages de	Arch.	Complexité par itération					$f^{\dagger\dagger}$	D_{it}	RDC_{it}
pipeline	R.D.E.*	LUT	FF	MP†	Slices	C_{it} (kG)	(MHz)	**(Gbps)**	(Mbps/kG)
0	Sans FF	3953	657	5	2838	95.147	38.70	0.193	2.03
12	Avec FF	4281	3659	5	3679 (11%)	136.824	264.48	**1.322**	**9.66**

*:Architecture du registre à décalage élémentaire, †:MULT18x18, ††:fréquence maximale d'opération

Ces résultats montrent que le décodeur à seuil des codes 5-CPDO $\{R = 5/10,\ J = 10,\ \alpha_J = 128\}$ pipeliné peut atteindre un très haut débit allant jusqu'à 1.322 Gbps. Ce

débit remarquable se justifie par le fait qu'un décodeur des codes 5-CPDO est 5 fois plus rapide qu'un décodeur de codes CDO.

5.6 Prototypage du DSI des codes PCDO à taux compatibles

5.6.1 Simulation du gestionnaire d'horloge

Le prototypage de DSI des codes PCDO à taux compatibles exige que les différentes horloges ainsi que les signaux de synchronisation du système soient générés de façon synchronisée. Cette synchronisation est assurée par le gestionnaire d'horloge dont le résultat de simulation post placement et routage (avec délai) est montré à la figure 5.6.

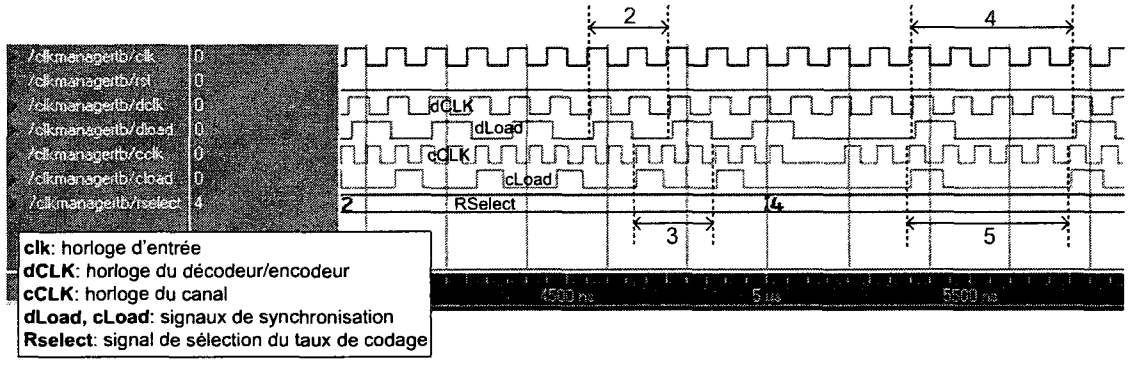

Figure 5.6: Simulation post placement et routage du gestionnaire d'horloge

Notons que le taux de codage, $R = b/(b+1)$, est choisi par le signal RSelect dont sa valeur est égale à b. Ainsi, dans cette simulation le taux de codage du système se change dynamiquement de $2/3$ à $4/5$ en changeant la valeur de RSelect de 2 à 4. Le gestionnaire d'horloge se resynchronise au nouveau taux de codage après quelques cycles seulement de l'horloge d'entrée, clk. À un taux de codage, $R = b/(b+1)$, les signaux dLoad et cLoad doivent être générés une fois chaque b cycles de dCLK et $(b+1)$ cycles de cCLK respectivement, ce qui est validé par cette simulation pour les deux taux de codage considérés.

5.6.2 Résultats expérimentaux du DSI des codes PCDO à taux compatibles

Dans cette section les résultats expérimentaux du DSI des codes PCDO à taux compatibles $\{J = 10, \alpha_J = 366\}$ sont présentés[2]. Les codes $\{J = 10, \alpha_J = 366\}$ supportent quatre taux de codage : $R = 1/2,\ 2/3,\ 3/4$ et $5/6$. Les résultats expérimentaux à la $6^{ième}$ itération du DSI sont illustrés à la figure 5.7. À chaque taux de codage, la valeur de CP qui minimise les BERs à 3 et 3.5 dB simultanément a été choisie.

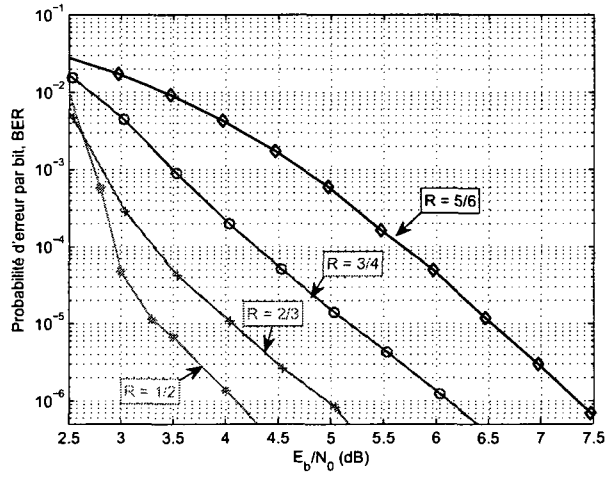

Figure 5.7: Résultats expérimentaux du DSI des codes PCDO à taux compatibles $\{J = 10, \alpha_J = 366\}$, $RInt = 5$ bits, 6 itérations, $CP_{R=1/2} = 0.297$, $CP_{R=2/3} = 0.422$, $CP_{R=3/4} = 0.547$ et $CP_{R=5/6} = 0.75$

5.6.3 Influence de la protection quasi-EEP sur les performances

Nous pouvons vérifier à la figure 5.8 la dégradation des performances du DSI des codes PCDO qu'entraîne une protection UEP par rapport à une protection quasi-EEP. Sachant que le spectre de perforation représente la distribution des équations de parité sur les différents sous-ensembles A_h pour un taux de codage perforé donné [30], les DSIs con-

[2]À l'annexes IV, V et VI, plusieurs résultats expérimentaux complémentaires à ce chapitre sont présentés.

sidérés dans cette comparaison sont ceux des codes listés au tableau 5.7. Le générateur $\{J = 10, \alpha_J = 1835\}$ est un générateur non relaxé tiré de [24].

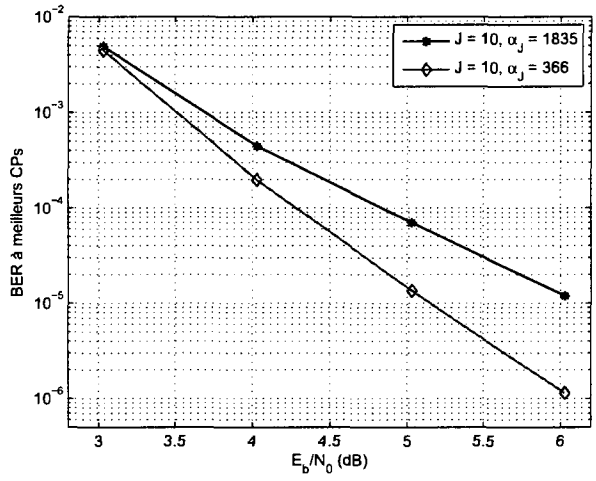

Figure 5.8: Comparaison de performances des codes PCDO de type quasi-EEP $\{J = 10, \alpha_J = 366\}$ avec les codes PCDO de type UEP $\{J = 10, \alpha_J = 1835\}$, 6 itérations, $R = 3/4$, $RInt = 5$ bits

Tableau 5.7: Codes PCDO de types quasi-EEP et UEP, $J = 10$, $R = 3/4$

Type	Spectre de perforation	δ_{max} (R=3/4)	P_c	Ensemble des connexions $\{\alpha_j\}$
quasi-EEP	(3,3,4)	0.292	12	{0, 9, 34, 107, 127, 208, 311, 330, 353, 366}
UEP	(2,4,4)	0.217	13	{0, 10, 121, 189, 588, 1235, 1332, 1742, 1808, 1835}

Bien que les codes PCDO $\{J = 10, \alpha_J = 366\}$ sont plus simplifiés, on constate qu'ils sont plus performants. Il s'ensuit donc que la protection quasi-EEP avantage une implémentation plus performante du DSI des codes PCDO.

5.7 Comparaison des codes doublement orthogonaux

5.7.1 Comparaison des codes doublement orthogonaux pour $R = 1/2$

Dans cette section, les décodeurs des différents codes doublement orthogonaux, $R = 1/2$, $J = 10$, sont abordés. En particulier, les codes CDO, les codes M-CDO et les codes M-CPDO. Le tableau 5.8 liste les codes considérés et compare la complexité et le débit des décodeurs correspondants. Les BERs à la $6^{ième}$ itération sont montrés à la figure 5.9.

Tableau 5.8: Comparaison de la complexité et du débit par itération des décodeurs à seuil des codes doublement orthogonaux, 12 étages de pipeline insérés, $R = 1/2$, $J = 10$ et $RInt = 5$ bits

Codes	R	α_J	δ_{max} (R=1/2)	C_{it} (kG)	D_{it} (Mbps)	RDC_{it} (Mbps/kG)	Latence$_{it}$	
							Cycles	μs
CDO	1/2	366	0.832	46.948	265.96	5.66	366	1.38
2-CDO	2/4	577	0.583	121.651	534.76	4.39	1154	2.16
5-CDO	5/10	462	0.43	265.670	1304.8	4.91	2310	1.77
2-CPDO	2/4	145	0.864	63.077	523.83	8.30	290	0.55
5-CPDO	5/10	128	0.803	136.824	1322.4	9.66	640	0.48

Les résultats expérimentaux montrent que les DSIs des codes M-CDO ont les meilleures performances d'erreur, ils sont parmi les plus rapides, mais aussi les plus complexes. Par exemple, pour un BER de 10^{-5}, les codes 5-CDO ont un gain de codage de plus de 0.6 dB par rapport aux codes CDO. Cependant, le décodeur des codes CDO est moins complexe bien que son débit soit limité à 265.96 Mbps seulement. Cette complexité réduite rend l'implémentation du décodeur des codes CDO plus efficace que celle des codes M-CDO comme paru en comparant les valeurs de RDC_{it}.

Toutefois, les codes M-CPDO obtenus en simplifiant la deuxième et la troisième conditions de la définition 4 et en particulier le codes 5-CPDO $\{R = 5/10,\ J = 10,\ \alpha_J = 128\}$, forment le meilleurs compromis entre complexité, débit et performances

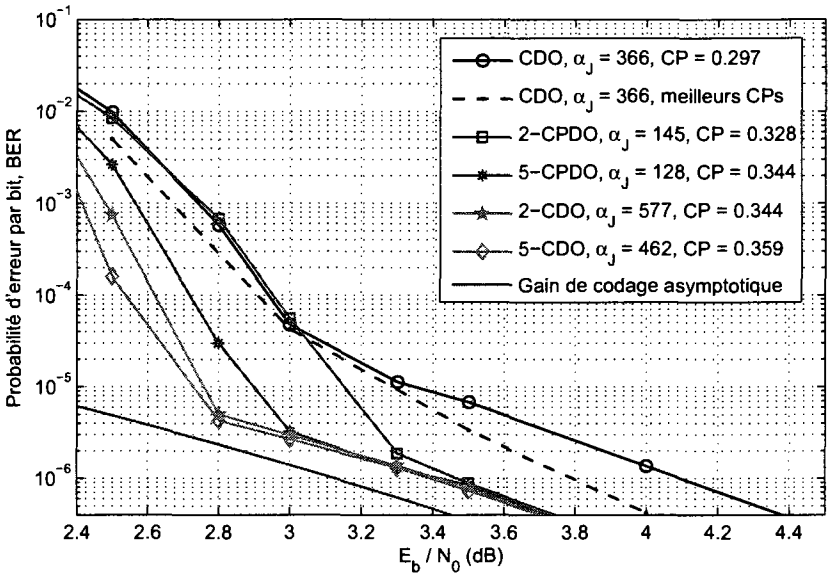

Figure 5.9: Comparaison de performances des codes doublement orthogonaux, 6 itérations, $J = 10$, $R = 1/2$, $RInt = 5$ bits

d'erreur. En effet, les performances des codes 5-CPDO $\{R = 5/10,\ J = 10,\ \alpha_J = 128\}$ rejoignent les performances des codes 5-CDO $\{R = 5/10,\ J = 10,\ \alpha_J = 462\}$ pour $E_b/N_0 \geq 3$ dB et perdent 0.2 dB seulement pour $E_b/N_0 < 3$ dB. Ces performances remarquables sont offertes par un décodeur ayant le débit le plus important de tous les codes et une complexité moyenne et plus abordable par rapport à celles du décodeur des codes M-CDO. Cette combinaison débit/complexité se traduit par une implémentation plus efficace que celle des deux autres types de codes. De plus, la réduction de la longueur de mémoire, α_J, par simplification des codes a permis au décodeur des codes 5-CPDO $\{R = 5/10,\ J = 10,\ \alpha_J = 128\}$ d'avoir la latence la moins importante d'une valeur de 0.48 μs seulement par itération.

Les résultats expérimentaux montrent aussi que les performances de tous les codes M-CDO et M-CPDO sont dictées par le facteur de simplification δ_{max}. En effet, les codes les moins simplifiés sont les plus performants et vice-versa. Cependant, bien que les codes 2-CPDO $\{R = 2/4,\ J = 10,\ \alpha_J = 145\}$ soient plus simplifiés que les codes

CDO $\{J = 10, \alpha_J = 366\}$, ils sont plus performants et convergent plus rapidement vers le gain asymptotique de codage de ces codes. Cela peut être due au fait que le nombre de différences doubles inévitables[3] croit selon un polynome en J_{RA}^4 et que ce nombre devient très important pour les codes CDO $\{J = 10, \alpha_J = 366\}$ où $J_{RA} = J = 10$.

5.7.2 Comparaison des codes doublement orthogonaux pour $R = 2/3$

Dans cette sous section, le décodeur des codes PCDO $\{J = 14, \alpha_J = 1359\}$, $R = 2/3$ [30], est comparé au décodeur des codes 2-CDO $\{R = 2/3, J = 7, \alpha_J = 613\}$. Les deux décodeurs offrent une protection d'erreur égale EEP pour $R = 2/3$ où 7 équations de parité sont utilisées pour décoder tous les bits d'information. Soit B le nombre minimal d'équations de parité utilisées pour décoder un bit à l'intérieur du décodeur à seuil des codes PCDO ou M-CDO. Alors, pour les codes PCDO de type EEP, $B = J/b$. D'autre part, pour les codes M-CDO de type EEP ayant $R = M/(M+1)$, $B = J_{RA}$. Les paramètres des deux décodeurs sont présentés au tableau 5.9. En outre, les performances d'erreur sont montrées à la figure 5.10.

Tableau 5.9: Comparaison du décodeur des codes PCDO $\{J = 14, \alpha_J = 1359\}$, 12 étages de pipeline insérés, avec le décodeur des codes 2-CDO $\{R = 2/3, J = 7, \alpha_J = 613\}$, 11 étages de pipeline insérés, $R = 2/3$ et $RInt = 5$ bits

Ensemble des connexions	δ_{max} (R=2/3)	C_{it} (kG)	D_{it} ($Mbps$)	RDC_{it} ($Mbps/kG$)	Latence$_{it}$	
					Cycles	μs
{0, 9, 40, 66, 102, 125, 390, 663, 813, 916, 1019, 1176, 1321, 1359}	0.555	125.106	264.76	2.12	1359	5.13
$\begin{bmatrix} 0 & 12 & 38 & 67 & 112 & 295 & 579 \\ 0 & 25 & 53 & 85 & 189 & 426 & 613 \end{bmatrix}$	0.606	101.625	538.21	5.30	1226	2.28

Les résultats de la figure 5.10 montrent que les performances des codes considérés sont équivalentes. Cependant, le décodeur des codes 2-CDO $\{R = 2/3, J = 7, \alpha_J = 613\}$

[3]Voir annexe II

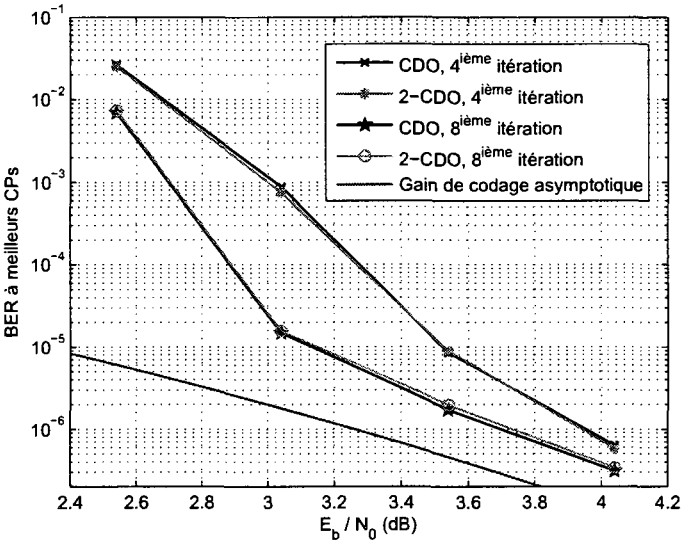

Figure 5.10: Comparaison de performances des codes PCDO $\{J = 14, \alpha_J = 1359\}$ avec les codes 2-CDO $\{R = 2/3, J = 7, \alpha_J = 613\}$, $R = 2/3$ et $RInt = 5$ bits

est deux fois plus rapide et deux fois moins complexe. La complexité réduite est due à la longueur de mémoire réduite de l'encodeur 2-CDO $\{R = 2/3, J = 7, \alpha_J = 613\}$ par rapport à celle de l'encodeur PCDO $\{J = 14, \alpha_J = 1359\}$, ce qui entraîne une réduction de la complexité engendrée par les registres à décalages du décodeur. Par conséquent, l'implémentation du décodeur des codes 2-CDO $\{R = 2/3, J = 7, \alpha_J = 613\}$ est 2.5 fois plus efficace que celle du décodeur des codes PCDO considéré.

En général, les résultats expérimentaux ont montré qu'un décodeur des codes M-CDO, $R = M/(M+1)$, garde les mêmes performances d'erreur du décodeur des codes PCDO, $R = b/(b+1)$ avec $M = b$, si les deux décodeurs utilisent le même nombre d'équations de parité B pour décoder les bits d'information. Cependant, le décodeur des codes M-CDO est M fois plus rapide. De plus, la complexité des décodeurs varie en fonction de la latence exprimée en nombre de cycles. La comparaison de la latence des encodeurs 2-CDO avec celle des encodeurs PCDO [30], $R = 2/3$, à la figure 5.11 montre que pour les valeurs de $B \geq 7$, les décodeurs des codes 2-CDO sont moins complexes. Toutefois, pour $B < 7$, la latence des codes PCDO est moins importante car l'auteur de [30] n'a

pas considéré la capacité de pipelinage dans la recherche de ses codes.

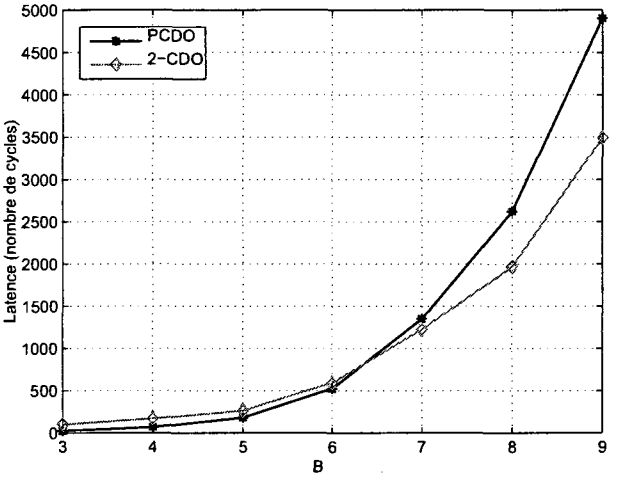

Figure 5.11: Comparaison de la latence de décodeurs à seuil des codes 2-CDO et PCDO, $R = 2/3$

5.8 Conclusion

Dans ce chapitre, nous avons présenté les résultats expérimentaux de l'implé-mentation matérielle des décodeurs à seuil itératifs conçus dans le cadre de ce projet. La technique de pipelinage appliquée au décodeur à seuil utilisant les nouvelles architectures du registre à décalage élémentaire et du pondérateur a augmenté le débit du décodeur jusqu'aux limites permises par la technologie. Les performances d'erreur des codes PCDO à taux compatibles qui supportent plusieurs taux de codage ont été également présentées. D'ailleurs, les résultats expérimentaux des nouveaux codes M-CDO introduits dans le cadre de ce projet ont montré que ces codes offrent un meilleur compromis entre complexité, débit et performances d'erreur que les codes CDO connus. En effet, pour des complexités matérielles abordables, les décodeurs de ces codes sont les plus performants et les plus rapides.

CHAPITRE 6

CONCLUSION

6.1 Bilan de la recherche réalisée

Ce mémoire, qui s'ajoute aux travaux précédemment entamés sur les codes correcteurs d'erreur dits convolutionnels doublement orthogonaux, a permis de réaliser la conception et le prototypage de décodeurs à seuil itératifs à haut débit pour ce type de codes.

Tout d'abord, nous avons exposé l'architecture du décodeur à seuil pour attirer l'attention du lecteur là où des améliorations du décodeur sont nécessaires afin d'augmenter le débit. Par la suite, nous avons introduit une technique de pipelinage efficace et développé une version parallèle du DSI des codes CDO :

- Réduction des délais engendrés par les composants du décodeur et élaboration de la technique de pipelinage.
- Introduction de la capacité de pipelinage comme un nouveau critère de sélection de générateurs des codes CDO.
- Implémentation de la première version du DSI qui supporte les codes CDO à taux compatibles opérant à haut débit.
- Introduction d'un nouveau type de codes CDO à multi-registres à décalage qui permet un décodage parallèle et qui a conduit à augmenter significativement le débit du décodeur (1.3 Gbps pour certains DSI).
- Génération et comparaison des résultats expérimentaux des codes.

Il en résulte que les architectures du DSI proposées répondent de façon satisfaisante aux exigences des systèmes pratiques de communication à haut débit.

6.2 Améliorations envisageables

Dans un premier temps, nous pourrions étudier la perforation des codes M-CDO, ce qui permet d'obtenir des codes M-CDO à taux compatibles ayant un DSI performant à haut débit et à faible complexité.

Deuxièmement, nous pourrions explorer la possibilité de construire des codeurs M-CDO récursifs (avec rétroaction sur la parité) et étudier la double orthogonalité des codes ainsi générés. La recherche de ces codes pourrait être avantageuse à cause de la complexité réduite du décodeur, son débit élevé et l'assurance de meilleures performances d'erreur.

Nous pourrions aussi intégrer le décodeur dans une application pratique de communication. Par exemple, nous pourrions envisager la construction d'un récepteur WiMAX utilisant le DSI et le comparer ainsi avec un récepteur utilisant les autres décodeurs adoptés par la norme.

Enfin, nous pourrions considérer l'implémentation d'un décodeur à seuil itératif formé d'un petit nombre d'itérations (une ou deux) et qui seront réitérées plusieurs fois. Un tel décodeur occupe peu d'espace sur un FPGA et nous pourrions ainsi vérifier l'effet de l'augmentation du nombre d'itérations sur les performances d'erreur à des faibles valeurs de E_b/N_0.

6.3 Ouverture

Les résultats remarquables des prototypes du DSI présentés motivent à aller plus loin dans ce domaine. Ainsi, la réalisation d'une version ASIC du DSI, dans le cadre de recherches futures, pourrait s'avérer très avantageuse. Pour une telle solution, il faut remplacer tous les composants propres au FPGA (LUT, Mult18x18 ...). Les librairies de la technologie ASIC peuvent former un bon point de départ dans ce cas. Le pondérateur pourrait être remplacé par un "Shifter" qui permet la multiplication par les puissances négatives de 2. À partir d'une version ASIC du DSI, nous pourrions envisager des débits beaucoup plus élevés que ceux obtenus par une solution FPGA. De plus, la consommation de la puissance pourrait être étudiée de façon précise en fonction du débit (fréquence d'opération) utilisé.

RÉFÉRENCES

[1] C. Berrou, A. Glavieux, et P. Thitimajshima, "Near shannon limit error-correcting coding and decoding: Turbo-codes. 1," *IEEE Int. Conf. on Communications, Geneva, Switzerland*, vol. 2, pp. 1064–1070, 1993.

[2] C. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, no. 1, pp. 379–423, Juillet 1948.

[3] D. MacKay et R. Neal, "Good codes based on very sparse matrices," *Proceedings of the 5th IMA Conference on Cryptography and Coding*, pp. 100 – 111, 1995.

[4] C. Cardinal, D. Haccoun, et F. Gagnon, "Iterative threshold decoding without interleaving for convolutional self-doubly orthogonal codes," *IEEE Transaction on Communications*, vol. 51, no. 8, pp. 1274–1282, Août 2003.

[5] ..., "Part 16: Air interface for fixed and mobile broadband wireless access systems amendment 2: Physical and medium access control layers for combined fixed and mobile operation in licensed bands and corrigendum 1," *IEEE Std 802.16e*, 2006.

[6] R. Hamming, "Error-detecting and error-correcting codes," *Bell System Technical Journal*, vol. 26, no. 2, pp. 147–160, Avril 1950.

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

[8] D. MacKay et R. Neal, "Near shannon limit performance of low density parity check codes," *Electronics Letters*, vol. 32, no. 18, pp. 1645–, Août 1996.

[9] ——, "Near shannon limit performance of low density parity check codes," *Electronics Letters*, vol. 33, no. 6, pp. 457–458, Mars 1997.

[10] J. Pearl, *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*. Morgan Kaufmann, Septembre 1988.

[11] F. Kschischang, B. Frey, et H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 498–519, Février 2001.

[12] M. Luby, M. Mitzenmacher, M. Shokrollahi, et D. Spielman, "Improved low-density parity-check codes using irregular graphs," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 585–598, Février 2001.

[13] P. Robertson, "Illuminating the structure of code and decoder of parallel concatenated recursive systematic (turbo) codes," in *IEEE Global Telecommunications Conference, GLOBECOM '94*, vol. 3, 1994, pp. 1298–1303 vol.3.

[14] S. Benedetto et G. Montorsi, "Unveiling turbo codes: some results on parallel concatenated coding schemes," *IEEE Transactions on Information Theory*, vol. 42, no. 2, pp. 409–428, Mars 1996.

[15] A. Amat, G. Montorsi, et S. Benedetto, "Design and decoding of optimal high-rate convolutional codes," *IEEE Transactions on Information Theory*, vol. 50, no. 5, pp. 867–881, Mai 2004.

[16] J. Hagenauer et P. Hoeher, "A viterbi algorithm with soft-decision outputs and its applications," in *IEEE Global Telecommunications Conference and Exhibition. GLOBECOM '89*, vol. 3, Novembre 1989, pp. 1680–1686.

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

[18] S. Bates, Z. Chen, L. Gunthorpe, A. Pusane, K. Zigangirov, et D. Costello, "A low-cost serial decoder architecture for low-density parity-check convolutional codes,"

IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 55, no. 7, pp. 1967–1976, Août 2008.

[19] C. Anghel, A. Enescu, O. Bugiugan, et C. Cacoveanu, "Fpga implementation of a ctc decoder for h-arq compliant wimax systems," *Proc. Int. Conf. on Design & Technology of Integrated Systems in Nanoscale Era*, pp. 82–86, 2007.

[20] T. Brack, M. Alles, T. Lehnigk-Emden, F. Kienle, N. Wehn, N. L'Insalata, F. Rossi, M. Rovini, et L. Fanucci, "Low complexity ldpc code decoders for next generation standards," *Proc. Design Autom. Test Eur., Nice, France*, pp. 16–21, 2007.

[21] J.-H. Kim et I.-C. Park, "Duo-binary circular turbo decoder based on border metric encoding for wimax," *Asia and South Pacific Design Automation Conference, ASPDAC 2008*, pp. 109–110, Mars 2008.

[22] G. Masera, F. Quaglio, et F. Vacca, "Implementation of a flexible ldpc decoder," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 54, no. 6, pp. 542–546, Juin 2007.

[23] F. Gagnon, D. Haccoun, N. Batani, et C. Cardinal, "Apparatur for convolutional self-doubly orthogonal encoding and decoding." U.S. Patent US6 167 552-A, Decembre 26, 2000.

[24] C. Cardinal, "Décodage à seuil itératif des codes convolutionnels doublement orthogonaux," Thèse de doctorat, École Polytechnique de Montréal, Juin 2001.

[25] D. Haccoun, C. Cardinal, et F. Gagnon, "Search and determination of convolutional self-doubly orthogonal codes for iterative threshold decoding," *IEEE Transactions on Communications*, vol. 53, no. 5, pp. 802–809, Mai 2005.

[26] J. Massey, *Threshold Decoding*. Cambridge, MA: M.I.T. Press, 1963.

[27] G. Provost, M. Cantin, M. Sawan, C. Cardinal, Y. Savaria, et D. Haccoun, "Fast parameters optimization of an iterative decoder using a configurable hardware ac-

celerator," *IEEE International Symposium on Circuits and Systems, ISCAS*, pp. 4159–4162 Vol. 4, Mai 2005.

[28] C. Cardinal, D. Haccoun, et Y.-C. He, "Reduced-complexity convolutional self-doubly orthogonal codes for efficient iterative decoding," in *IEEE 63rd Vehicular Technology Conference. VTC 2006-Spring.*, vol. 3, Mai 2006, pp. 1372–1376.

[29] C. Cardinal, E. Roy, et D. Haccoun, "Simplified convolutional self-doubly orthogonal codes: search algorithms and codes determination," *IEEE Transactions on Communications*, vol. 57, no. 6, pp. 1674–1682, Juin 2009.

[30] E. Roy, "Recherche et analyse de codes convolutionnels doublement orthogonaux simplifiés au sens large," Mémoire de maîtrise, École Polytechnique de Montréal, Août 2006.

[31] D. Haccoun et C. Cardinal, "High-rate punctured convolutional self-doubly orthogonal codes for iterative threshold decoding," *IEEE Transactions on Communications*, vol. 53, no. 1, pp. 55–63, Janvier 2005.

[32] E. Roy, C. Cardinal, et D. Haccoun, "Simplified high-rate punctured convolutional self-doubly orthogonal codes," *IEEE International Symposium on Information Theory, ISIT*, pp. 2696–2699, Juin 2007.

[33] La compagnie Xilinx 2009, en ligne : www.xilinx.com.

[34] C. E. Leiserson et J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, no. 1, pp. 5–35, 1991.

[35] G. Provost, M. Sawan, C. Cardinal, et D. Haccoun, "Implementation and error performance evaluation of an iterative decoding algorithm," *Proc. 3rd Int. IEEE-NEWCAS Conf.*, pp. 263–266, 2005.

[36] G. Provost, "Implémentation et optimisation d'un décodeur à seuil itératif de codes convolutionnels doublement orthogonaux," Mémoire de maîtrise, École Polytechnique de Montréal, Avril 2005.

[37] La compagnie ARM 2009, en ligne : www.arm.com.

[38] A. Nemr, C. Cardinal, M. Sawan, et D. Haccoun, "Very high throughput iterative threshold decoder for convolutional self-doubly orthogonal codes," *Proc. Joint 6th Int. IEEE Northeast Workshop on Circuits and Systems and TAISA Conf.*, pp. 257–260, 2008.

[39] M. Dubois, Y. Savaria, D. Haccoun, et N. Belanger, "Low-power configurable and generic shift register hardware realisations for convolutional encoders and decoders," *IEEE Proceedings -Circuits, Devices and Systems*, vol. 153, no. 3, pp. 207–213, Juin 2006.

[40] Y.-C. He et D. Haccoun, "An analysis of the orthogonality structures of convolutional codes for iterative decoding," *IEEE Transactions on Information Theory*, vol. 51, no. 9, pp. 3247–3261, Septembre 2005.

ANNEXE I

ARCHITECTURE DES COMPOSANTS DU DÉCODEUR À SEUIL DES CODES CDO

I.1 Opérateurs élémentaires

Les opérateurs élémentaires sont utilisés implicitement dans le décodeur afin d'implémenter d'autres opérateurs ou composants. Nous distinguons dans le design du décodeur deux opérateurs élémentaires : l'additionneur à deux entrées et l'opérateur Addmin à deux entrées. L'additionneur à deux entrées est utilisé afin d'implémenter l'arbre de l'additionneur à $(J + 1)$ entrées qui additionne les J équations de parité, $\psi_{(i,j)}^{(\mu)}$, avec le symbole d'information, y_i^u. L'opérateur Addmin à deux entrées est utilisé dans la implémentation de l'opérateur Addmin global (AG) et le registre à décalage de rétroaction (RD3) qui servent à calculer les équations de parité.

I.1.1 Additionneur à deux entrées

L'additionneur à deux entrées effectue l'addition des deux opérandes présents à ses entrées. Les deux opérandes sont signés et représentés en format C2. À la sortie, l'additionneur à deux entrées ajoute un bit de précision pour tenir compte de tout dépassement (overflow) qui peut se produire. Ainsi, comme montré à la figure I.1, si les entrées ont une résolution de k bits, alors la sortie de l'additionneur à deux entrées a une résolution de $k + 1$ bits.

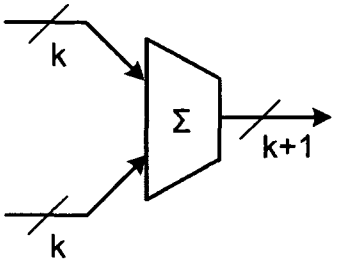

Figure I.1: Additionneur à deux entrées

I.1.2 Opérateur Addmin à deux entrées

L'opérateur Addmin à deux entrées applique l'opération "addmin" exprimée par (2.14) sur les deux opérandes représentés en format SA présents à ses entrées. L'amplitude et le signe de la sortie de cet opérateur sont calculés séparément. L'amplitude à la sortie de l'opérateur Addmin est égale au minimum des deux amplitudes des opérandes et elle s'obtient en utilisant un comparateur et un mulyiplexeur (MUX). Le signe du résultats obtenu à la sortie de l'opérateur Addmin est déterminé par un opérateur XNOR dont les entrées sont les signes des opérandes. Ainsi, la sortie de l'opérateur Addmin à deux entrées a la même résolution binaire que ses entrées. Son architecture est illustrée à la figure I.2.

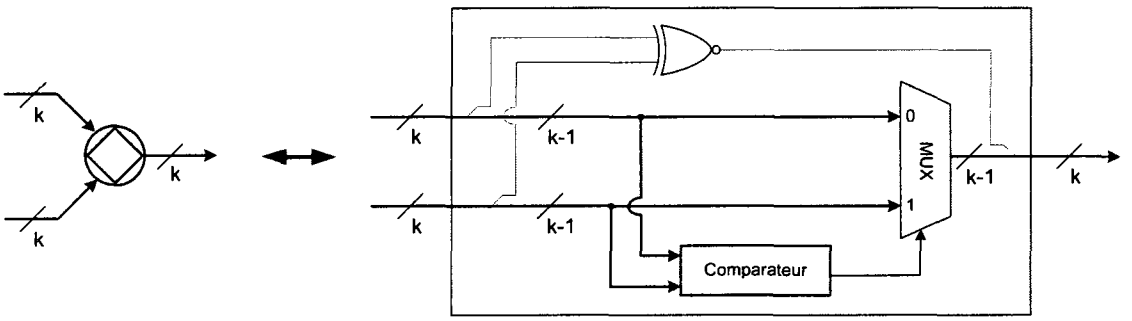

Figure I.2: Architecture de l'opérateur Addmin à deux entrées

I.2 Opérateur de conversion binaire

L'implémentation des différents opérateurs du décodeur nécessite pour chacun d'eux l'adoption d'une représentation binaire convenable. Afin d'implémenter l'additionneur, la représentation binaire "Complément à deux" (C2) est adoptée puisque l'addition en C2 peut s'effectuer directement sans avoir recours à aucune conversion des nombres négatives. Ainsi, l'implémentation d'un additionneur utilisant ce format possède une complexité matérielle moins élevée et un délai combinatoire réduit.

D'autre part, l'opération "addmin" exprimée par (2.14) est basée sur la comparaison des amplitudes (valeurs absolues) des opérandes ainsi que sur un calcul indépendant des signes. Par conséquent, la représentation binaire "Signe-Amplitude" (SA) est adoptée pour implémenter les opérateurs Addmin car avec ce format, l'amplitude de chaque opérande est directement disponible. La représentation SA favorise une implémentation plus rapide du pondérateur comme montré à la sous-section 3.2.1. Le saturateur qui suit directement le pondérateur est aussi implémenté en utilisant la même représentation.

En somme, la représentation SA est adoptée pour implémenter tous les composants du décodeur sauf l'additionneur où la représentation C2 est utilisée. Afin d'avoir les entrées de l'additionneur en format C2, un opérateur de conversion binaire (SA/C2) est inséré avant l'additionneur. Cet opérateur fait passer les nombres positifs inchangés et effectue la conversion complément à deux des nombres négatifs. La conversion complément à deux consiste à inverser tous les bits de l'amplitude d'un nombre négatif et à additionner 1 à celui-ci. Évidemment, la conversion inverse de C2 vers SA consiste à effectuer une autre conversion complément à deux. Alors, un autre opérateur de conversion binaire (C2/SA) est inséré après l'additionneur afin de convertir sa sortie en format signe amplitude. Il importe de noter que l'opérateur de conversion binaire ne change pas le nombre de bits sur lesquels les symboles sont représentés. Ainsi, chaque sortie de l'opérateur a

le même nombre de bits que l'entrée correspondante.

I.3 Le détecteur de zéros

L'utilisation des deux représentations binaires SA et C2 dans un circuit numérique conduit au problème commun de l'interprétation du nombre binaire "$10\cdots0$". Afin de bien comprendre ce problème, considérons des nombres entiers représentés par des mots binaires ayant une largeur de 4 bits. En C2, le mot "$10\cdots0$" est le plus petit nombre négatif représentable sur un nombre donné de bits, soit "$1000 = -8$" sur 4 bits, tandis que le même mot représente en SA le nombre zéro ($1000 = -0 = 0$). Toutefois, la conversion (SA/C2) du nombre "1000" donne le même nombre ($1000 \rightarrow 0111 \rightarrow 1000$). En d'autres termes, la conversion à l'entrée de l'additionneur d'un zéro "négatif" ($1000 = -0$) en SA donne ($1000 = -8$) en C2. De même, la conversion à la sortie de l'additionneur de -8 en C2 génère un zéro négatif en SA. Bref, afin d'éviter toute sorte d'interprétation erronée de ces zéros négatifs à l'entrée de l'additionneur, un opérateur "détecteur de zéros" est inséré directement après l'opérateur de conversion binaire (SA/C2). Cet opérateur détecte la présence des zéros négatifs (1000) pour ensuite les transformer en des zéros réguliers (0000).

D'autre part, l'absence des plus petits nombres négatifs à l'entrée de l'additionneur empêche la génération des plus petits nombres négatifs à sa sortie. En effet, l'additionneur produit à sa sortie le plus petit nombre négatif si et seulement si les opérandes étaient des plus petits nombres négatifs : $10000 = 1000 + 1000$ ou $-16 = -8 + (-8)$. Par conséquent, l'introduction d'un détecteur de zéros à la sortie de l'additionneur ne s'avère pas nécessaire (voir figure 3.1). Ainsi, l'élimination de cette opération contribue à réduire la complexité globale du décodeur ainsi que les délais combinatoires qui limitent son débit.

I.4 Registres à décalage

I.4.1 Registres à décalage de l'information et de parité

Le registre à décalage d'information (RD1) mémorise les symboles d'information et fournit le symbole y_i^u à l'additionneur et à la sortie correspondante. À son tour, le registre à décalage de parité (RD4) mémorise les symboles de parité afin de les synchroniser avec y_i^u et $\lambda_i^{(\mu)}$. La longueur de ces registres à décalage est égale à la mémoire de l'encodeur, α_J. Puisqu'une quantification douce sur 3 bits est utilisée pour quantifier les symboles y^u et y^p reçus du canal, la largeur de RD1 et RD4 est aussi fixée à 3 bits. Ainsi, un registre à décalage élémentaire de longueur $L = \alpha_J$ et de largeur $W = 3$ est utilisé pour implémenter chacun de RD1 et RD4.

I.4.2 Registre à décalage de $\lambda^{(\mu-1)}$

À l'itération μ, le registre à décalage RD2 conserve les valeurs de sortie $\lambda^{(\mu-1)}$. Il fournit à l'opérateur Addmin global les valeurs correspondantes aux symboles $\lambda_{i+(\alpha_j-\alpha_k)}^{(\mu-1)}$, $j > k$, requises pour effectuer le calcul des équations de parité $\psi_{(i,j)^{(\mu)}}$ données par (2.13). Pour $j > k$, $\lambda_{i+(\alpha_j-\alpha_k)}^{(\mu-1)}$ inclut tous les termes impliquant les différences simples positives du code convolutionnel utilisé.

Soit $N_s = J(J-1)/2$, le nombre des différences simples positives [30] et $DS^+ = \{\beta_h, h = 1, 2, \cdots, N_s\}$, l'ensemble des différences simples positives avec $\beta_i > \beta_j$, $i > j$, β_i et $\beta_j \in DS^+$. La plus grande différence positive correspond alors à $\beta_{N_s} = \alpha_J - \alpha_1 = \alpha_J$ ($\alpha_1 = 0$). À chaque différence positive $\beta_h \in DS^+$ correspond une sortie R_h dans RD2. Pour implémenter RD2, entre chaque sorties R_h et R_{h-1} un registre à décalage élémentaire de longueur $(\beta_h - \beta_{h-1})$ et de largeur $RInt$ est inséré. À l'entrée de RD2, la sortie R_{N_s} qui correspond à la différence $\beta_{N_s} = \alpha_J$ est connectée directement

à $\lambda_{i+\alpha_J}^{(\mu-1)}$. La longueur total de RD2 est alors donnée par $\beta_{N_s} - \beta_1 = \alpha_J - min(DS^+)$ où $min(DS^+)$ représente la plus petite différence positive. Notons qu'à la première itération, l'entrée de RD2 est connectée à $y_{i+\alpha_J}^u$.

I.5 Le saturateur

Le saturateur (Sat) a été utilisé dans [36] pour limiter le nombre de bits dans le noyau de logique combinatoire du décodeur. À l'entrée de l'additionneur, les opérandes ont une largeur équivalant à la résolution binaire du décodeur, $RInt$. La largeur de la sortie de l'additionneur est équivalant à :

$$RS_{ADD} = RInt + n \tag{I.1}$$

où n est le nombre des niveaux de l'additionneur donné par (3.10). En remplaçant (I.1) dans (3.1), la largeur de la sortie du pondérateur peut s'écrire:

$$RS_{Pond} = RInt + n + R_{CP} \tag{I.2}$$

Le saturateur réduit donc la largeur de S_{Pond} et le transforme de nouveau à $RInt$ avant d'être réinjectée dans RD3 de l'itération courante ou dans RD2 de l'itération suivante. Prenons par exemple le cas où $RInt = 5$ bits. Si la valeur absolue de S_{Pond} est plus grande que "01111" alors il la sature à "s1111" où "s" représente le signe de S_{Pond}. Cette saturation est possible grâce aux propriétés de l'opérateur Addmin qui determine en valeur absolue le minimum de ses entrées. Dans le cas où la valeur absolue de S_{Pond} est plus petite que "01111", le saturateur élimine les $(n+R_{CP})$ bits de précision. Il est demontré dans [36] que la saturation n'affecte pas les performances d'erreur du décodeur.

ANNEXE II

NOMBRE DE DIFFÉRENCES DOUBLES INÉVITABLES EN FONCTION DE J_{RA}

En considérant la formule des différences doubles $(\alpha_{m,q,j} - \alpha_{l,q,k}) + (\alpha_{l,p,r} - \alpha_{n,p,s})$, les différences doubles inévitables (DDI) existent seulement si les indices $(m, q) = (l, p)$ en permutant j et r ou si $(n, p) = (l, p)$ en permutant k et s.

II.1 Cas où $(m, q) = (l, p) = (n, p)$:

Dans ce cas, il s'agit des DDI obtenues à partir de $(\alpha_{m,q,j} - \alpha_{m,q,k}) + (\alpha_{m,q,r} - \alpha_{m,q,s})$ associées à l'ensemble $A_{m,q}$ ayant un maximum de J_{RA} connexions (le même principe s'applique aux générateur des codes CDO ayant $J = J_{RA}$ connexions) :

1. Si $J_{RA} = 2$, alors les deux différences doubles $\pm 2(\alpha_{m,q,j} - \alpha_{m,q,k})$ existent seulement et le nombre de DDI, N_{DDI}, est égale à zéro.

2. Si $J_{RA} = 3$, alors pour chaque couple de connexions $(\alpha_{m,q,j}, \alpha_{m,q,r})$, un DDI se produit en permutant j et r dans $(\alpha_{m,q,j} - \alpha_{m,q,k}) + (\alpha_{m,q,r} - \alpha_{m,q,k})$ et un autre DDI se produit en permutant les mêmes indices dans $(\alpha_{m,q,k} - \alpha_{m,q,j}) + (\alpha_{m,q,k} - \alpha_{m,q,r})$. Ainsi, $N_{DDI} = 2 \times \binom{3}{2} = 6$.

3. Si $J_{RA} \geq 4$, alors ils existent 6 DDI dans chaque sous ensemble de trois connexions et quatre différences doubles égales dont trois sont des DDI pour chaque deux couples de connexions $(\alpha_{m,q,j}, \alpha_{m,q,r})$ et $(\alpha_{m,q,k}, \alpha_{m,q,s})$. Par conséquent, le

nombre total de DDI est donné par :

$$\begin{aligned} N_{DDI} &= 6 \times \binom{J_{RA}}{3} + 3 \times \binom{J_{RA}}{2} \times \binom{J_{RA}-2}{2} \\ &= \frac{(3J_{RA}^4 - 14J_{RA}^3 + 21J_{RA}^2 - 10J_{RA})}{4} \end{aligned} \tag{II.1}$$

Notons que l'équation (II.1) est valide $\forall\ J_{RA} > 0$.

II.2 Cas où $(m,q) = (l,p) \neq (n,p)$:

Dans ce cas, il s'agit des DDI obtenues à partir de $(\alpha_{m,q,j} - \alpha_{m,q,k}) + (\alpha_{m,q,r} - \alpha_{n,q,s})$:

1. Si $J_{RA} = 2$, alors le nombre de DDI est $N'_{DDI} = 0$.
2. Si $J_{RA} = 3$, alors les DDI se produisent en permutant j et r, soit $N'_{DDI} = \binom{3}{2} = 3$.
3. Si $J_{RA} \geq 4$, alors ils existent 3 DDI dans chaque sous ensemble de trois connexions. Ainsi, le nombre total de DDI est donné par :

$$\begin{aligned} N'_{DDI} &= 3 \times \binom{J_{RA}}{3} \\ &= \frac{(J_{RA}^3 - 3J_{RA}^2 + 2J_{RA})}{2} \end{aligned} \tag{II.2}$$

ce qui reste valide $\forall\ J_{RA} > 0$ ainsi que dans le cas où $(m,q) \neq (l,q) = (n,p)$.

À partir de (II.1) et (II.2), nous constatons que si $J_{RA} \leq 2$ alors le nombre de différences doubles inévitables est égale à zéro pour les codes M-CDO.

ANNEXE III

CODE VHDL DU GESTIONNAIRE D'HORLOGE

```
- - ----------------------------------------------------------------
- - École Polytechnique de Montréal
- - Departement de Génie Électrique
- - ----------------------------------------------------------------
- - TITLE :    Gestionnaire d'horloge
- - DESCRIPTION :    Ce module implémente le Gestionnaire d'horloge
- -                   pour les taux de codage 1/2 < R < 8/9
- -
- - FILE :    clkManager.vhd
- - ----------------------------------------------------------------
- - CREATION
- - DATE    AUTHOR    PROJECT    REVISION
- - 28/04/2008    Abbas NEMR    DSI des codes PCDO à taux compatibles    v1.0
- - ----------------------------------------------------------------
- - MODIFICATION    HISTORY
- - DATE    AUTHOR    PROJECT    REVISION    COMMENTS
- - ----------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_unsigned.all;
library UNISIM;
use UNISIM.Vcomponents.ALL;
```

```
entity clkManager is
    generic (
        CLKIN_PERIOD : real := 14.000    - - clk period in ns
    );
    Port (
        - - Inputs
        CLK_IN : in std_logic;    - - Input clk
        RST_IN : in std_logic;    - - Input rst
        RSelect : in std_logic_vector(2 downto 0);    - - Coding Rate Selector
        - - Outputs
        dCLK : out std_logic;    - - Encoder/decoder clk
        dLoad : out std_logic;    - - Encoder/decoder load
        cCLK : out std_logic;    - - Channel clk
        cLoad : out std_logic;    - - Channel load
        synchDone : out std_logic    - - Outputs are Ready
    );
end clkManager;

architecture BEHAVSTRUCT of clkManager is
    signal GND_BIT : std_logic;
    signal CLKIN_IBUFG : std_logic;
    - - ————
    - -RST MSA
    signal iRST : std_logic;
    signal iRSTCurrentState , iRSTFutureState : std_logic_vector(2 downto 0);
    - - ————
    - -dLoad MSA
```

```
    signal idload : std_logic;
    signal idLoadCurrentState , idLoadFutureState : std_logic_vector(3 downto 0);
    - - ————————
    - -cLoad MSA
    signal icload : std_logic;
    signal icLoadCurrentState , icLoadFutureState : std_logic_vector(3 downto 0);
    - - ————————
    - -synchDone MSA
    signal iSynch : std_logic;
    signal isynchCurrentState , isynchFutureState : std_logic_vector(1 downto 0);
    - - ————————
    signal cCLK1 , cCLK2, cCLK3, cCLK4, cCLK5, cCLK6, cCLK7, cCLK8 : std_logic;
    signal icCLK, CLKTmp1, CLKTmp2, CLKTmp3 : std_logic;
    signal CLKTmp4, CLKTmp5, CLKTmp6 : std_logic;

begin
GND_BIT <= '0';
- - ——————————————————————
- - IBUFG input clk
CLKIN_IBUFG_INST : IBUFG
    port map (
        I=>CLK_IN,
        O=>CLKIN_IBUFG);
- - ——————————————————————
- - iRES MSA
- - iRES doit être maintenu actif au moins 3 cycles afin de remettre à zéro tous les DCM
iRES_synch : process(RST_IN , CLKIN_IBUFG)
begin
```

```
    if (RST_IN = '1') then
        iRSTCurrentState <= "000";
    elsif CLKIN_IBUFG'event and CLKIN_IBUFG = '1' then
        iRSTCurrentState <= iRSTFutureState;
    end if;
end process;
iRES_asynch : process(iRSTCurrentState)
begin
    case iRSTCurrentState is
        when "000" =>
            iRSTFutureState <= "001";
            iRST <= '1';
        when "001" =>
            iRSTFutureState <= "011";
            iRST <= '1';
        when "011" =>
            iRSTFutureState <= "010";
            iRST <= '1';
        when "010" =>
            iRSTFutureState <= "110";
            iRST <= '1';
        when "110" =>
            iRSTFutureState <= "111";
            iRST <= '1';
        when "111" =>
            iRSTFutureState <= "101";
            iRST <= '1';
        when "101" =>
```

```
            iRSTFutureState <= "100";
            iRST <= '1';
        when "100" =>
            iRSTFutureState <= "100";
            iRST <= '0';
        when others =>
            iRSTFutureState <= "100";
            iRST <= '0';
    end case;
end process;
- - ----------------------------
- - Signaux de synchronisation : dLoad = idLoad
- - dLoad = '1' chaque (RSelect) cycles de dCLK
idload_synch : process(iRST , CLKIN_IBUFG)
begin
    if (iRST = '1') then
        idLoadCurrentState <= "0000";
    elsif CLKIN_IBUFG'event and CLKIN_IBUFG = '1' then
        idLoadCurrentState <= idLoadFutureState;
    end if;
end process;
idload_asynch : process(idLoadCurrentState, RSelect)
begin
    case idLoadCurrentState is
        when "0000" =>
            if (RSelect = "001") then - - R = 1/2
                idLoadFutureState <= "1100";
            else
```

```
            idLoadFutureState <= "0001";
        end if;
        idload <= '0';
    when "0001" =>
        if (RSelect = "010") then - - R = 2/3
            idLoadFutureState <= "1100";
        else
            idLoadFutureState <= "0011";
        end if;
        idload <= '0';
    when "0011" =>
        if (RSelect = "011") then - - R = 3/4
            idLoadFutureState <= "1100";
        else
            idLoadFutureState <= "0010";
        end if;
        idload <= '0';
    when "0010" =>
        if (RSelect = "100") then - - R = 4/5
            idLoadFutureState <= "1100";
        else
            idLoadFutureState <= "0110";
        end if;
        idload <= '0';
    when "0110" =>
        if (RSelect = "101") then - - R = 5/6
            idLoadFutureState <= "1100";
        else
```

```
            idLoadFutureState <= "0111";
        end if;
        idload <= '0';
    when "0111" =>
        if (RSelect = "110") then - - R = 6/7
            idLoadFutureState <= "1100";
        else
            idLoadFutureState <= "0101";
        end if;
        idload <= '0';
    when "0101" =>
        if (RSelect = "111") then - - R = 7/8
            idLoadFutureState <= "1100";
        else
            idLoadFutureState <= "0100";
        end if;
        idload <= '0';
    when "0100" =>
        if (RSelect = "000") then - - R = 8/9
            idLoadFutureState <= "1100";
        else
            idLoadFutureState <= "0000";
        end if;
        idload <= '0';
    when "1100" =>
        if (RSelect = "001") then
            idLoadFutureState <= "1100";
        else
```

```
                idLoadFutureState <= "0001";
            end if;
            idload <= '1';
        when others =>
            idLoadFutureState <= "0001";
            idload <= '0';
    end case;
end process;
-- ----------------------
-- Signaux de synchronisation : cLoad = icLoad
-- cLoad = '1' chaque (RSelect + 1) cycles de cCLK
-- idLoad est surveillé afin de séparer les fronts montant de cLoad et dLoad
icload_synch : process(iRST , icCLK )
begin
    if (iRST = '1') then
        icLoadCurrentState <= "0000";
    elsif icCLK'event and icCLK = '1' then
        icLoadCurrentState <= icLoadFutureState;
    end if;
end process;
icload_asynch : process(icLoadCurrentState, RSelect,idload)
begin
    case icLoadCurrentState is
        when "0000" =>
            if idload = '1' then
                icLoadFutureState <= "0001";
            else
                icLoadFutureState <= "0000";
```

```
        end if;
        icload <= '0';
    when "0001" =>
        icLoadFutureState <= "0100";
        icload <= '0';
    when "0100" =>
        icLoadFutureState <= "1100";
        icload <= '1';
    when "1100" =>
        if (RSelect = "001") then - - R = 1/2
            icLoadFutureState <= "0100";
        else
            icLoadFutureState <= "1101";
        end if;
        icload <= '0';
    when "1101" =>
        if (RSelect = "010") then - - R = 2/3
            icLoadFutureState <= "0100";
        else
            icLoadFutureState <= "1111";
        end if;
        icload <= '0';
    when "1111" =>
        if (RSelect = "011") then - - R = 3/4
            icLoadFutureState <= "0100";
        else
            icLoadFutureState <= "1110";
        end if;
```

```
    icload <= '0';
when "1110" =>
    if (RSelect = "100") then - - R = 4/5
        icLoadFutureState <= "0100";
    else
        icLoadFutureState <= "1010";
    end if;
    icload <= '0';
when "1010" =>
    if (RSelect = "101") then - - R = 5/6
        icLoadFutureState <= "0100";
    else
        icLoadFutureState <= "1011";
    end if;
    icload <= '0';
when "1011" =>
    if (RSelect = "110") then - - R = 6/7
        icLoadFutureState <= "0100";
    else
        icLoadFutureState <= "1001";
    end if;
    icload <= '0';
when "1001" =>
    if (RSelect = "111") then - - R = 7/8
        icLoadFutureState <= "0100";
    else
        icLoadFutureState <= "1000";
    end if;
```

```
            icload <= '0';
        when "1000" =>
            if (RSelect = "000") then - - R = 8/9
                icLoadFutureState <= "0100";
            else
                icLoadFutureState <= "1100";
            end if;
            icload <= '0';
        when others =>
            icLoadFutureState <= "0000";
            icload <= '0';
    end case;
end process;
- - ------------------------------
- - Signaux de synchronisation : synchDone = iSynch
- - iSynch = '1' si cLoad et dLoad sont prêts
iSynch_synch : process(RST_IN , icCLK)
begin
    if (RST_IN = '1') then
        iSynchCurrentState <= "00";
    elsif icCLK'event and icCLK = '1' then
        iSynchCurrentState <= iSynchFutureState;
    end if;
end process;
iSynch_asynch : process(iSynchCurrentState, icload, idload)
begin
    case iSynchCurrentState is
        when "00" =>
```

```
            if idload ='1' then
                iSynchFutureState <= "01";
            else
                iSynchFutureState <= "00";
            end if;
            iSynch <= '0';
        when "01" =>
            if icload ='1' then
                iSynchFutureState <= "11";
            else
                iSynchFutureState <= "01";
            end if;
            iSynch <= '0';
        when "11" =>
            iSynchFutureState <= "11";
            iSynch <= '1';
        when others =>
            iSynchFutureState <= "00";
            iSynch <= '0';
    end case;
end process;
- - ------------------------------
- - DCM
- - 1 DCM est utilisé pour génèrer chaque fréquence de cCLK
- - R = CLKFX_DIVIDE / CLKFX_MULTIPLY
DCM_INST1_R12 : DCM
    generic map( CLK_FEEDBACK => "NONE",
        CLKDV_DIVIDE => 2.5,
```

```
    CLKFX_DIVIDE => 1,
    CLKFX_MULTIPLY => 2,
    CLKIN_DIVIDE_BY_2 => FALSE,
    CLKIN_PERIOD => CLKIN_PERIOD,
    CLKOUT_PHASE_SHIFT => "NONE",
    DESKEW_ADJUST => "SYSTEM_SYNCHRONOUS",
    DFS_FREQUENCY_MODE => "LOW",
    DLL_FREQUENCY_MODE => "LOW",
    DUTY_CYCLE_CORRECTION => TRUE,
    FACTORY_JF => x"C080",
    PHASE_SHIFT => 0,
    STARTUP_WAIT => FALSE)
port map (CLKFB=> GND_BIT,
    CLKIN=> CLKIN_IBUFG,
    DSSEN=>GND_BIT,
    PSCLK=>GND_BIT,
    PSEN=>GND_BIT,
    PSINCDEC=>GND_BIT,
    RST=> iRST,
    CLKDV=>open,
    CLKFX=> cCLK1,
    CLKFX180=>open,
    CLK0=> open,
    CLK2X=>open,
    CLK2X180=>open,
    CLK90=>open,
    CLK180=>open,
    CLK270=>open,
```

```
        LOCKED=> open,
        PSDONE=>open,
        STATUS=>open);

DCM_INST1_R23 : DCM
    generic map( CLK_FEEDBACK => "NONE",
        CLKDV_DIVIDE => 2.5,
        CLKFX_DIVIDE => 2,
        CLKFX_MULTIPLY => 3,
        CLKIN_DIVIDE_BY_2 => FALSE,
        CLKIN_PERIOD => CLKIN_PERIOD,
        CLKOUT_PHASE_SHIFT => "NONE",
        DESKEW_ADJUST => "SYSTEM_SYNCHRONOUS",
        DFS_FREQUENCY_MODE => "LOW",
        DLL_FREQUENCY_MODE => "LOW",
        DUTY_CYCLE_CORRECTION => TRUE,
        FACTORY_JF => x"C080",
        PHASE_SHIFT => 0,
        STARTUP_WAIT => FALSE)
    port map (CLKFB=> GND_BIT,
        CLKIN=> CLKIN_IBUFG,
        DSSEN=>GND_BIT,
        PSCLK=>GND_BIT,
        PSEN=>GND_BIT,
        PSINCDEC=>GND_BIT,
        RST=> iRST,
        CLKDV=>open,
        CLKFX=> cCLK2,
```

```
        CLKFX180=>open,
        CLK0=> open,
        CLK2X=>open,
        CLK2X180=>open,
        CLK90=>open,
        CLK180=>open,
        CLK270=>open,
        LOCKED=> open,
        PSDONE=>open,
        STATUS=>open);

DCM_INST1_R34 : DCM
    generic map( CLK_FEEDBACK => "NONE",
        CLKDV_DIVIDE => 2.5,
        CLKFX_DIVIDE => 3,
        CLKFX_MULTIPLY => 4,
        CLKIN_DIVIDE_BY_2 => FALSE,
        CLKIN_PERIOD => CLKIN_PERIOD,
        CLKOUT_PHASE_SHIFT => "NONE",
        DESKEW_ADJUST => "SYSTEM_SYNCHRONOUS",
        DFS_FREQUENCY_MODE => "LOW",
        DLL_FREQUENCY_MODE => "LOW",
        DUTY_CYCLE_CORRECTION => TRUE,
        FACTORY_JF => x"C080",
        PHASE_SHIFT => 0,
        STARTUP_WAIT => FALSE)
    port map (CLKFB=> GND_BIT,
        CLKIN=> CLKIN_IBUFG,
```

```
        DSSEN=>GND_BIT,
        PSCLK=>GND_BIT,
        PSEN=>GND_BIT,
        PSINCDEC=>GND_BIT,
        RST=> iRST,
        CLKDV=>open,
        CLKFX=> cCLK3,
        CLKFX180=>open,
        CLK0=> open,
        CLK2X=>open,
        CLK2X180=>open,
        CLK90=>open,
        CLK180=>open,
        CLK270=>open,
        LOCKED=> open,
        PSDONE=>open,
        STATUS=>open);

DCM_INST1_R45 : DCM
    generic map( CLK_FEEDBACK => "NONE",
        CLKDV_DIVIDE => 2.5,
        CLKFX_DIVIDE => 4,
        CLKFX_MULTIPLY => 5,
        CLKIN_DIVIDE_BY_2 => FALSE,
        CLKIN_PERIOD => CLKIN_PERIOD,
        CLKOUT_PHASE_SHIFT => "NONE",
        DESKEW_ADJUST => "SYSTEM_SYNCHRONOUS",
        DFS_FREQUENCY_MODE => "LOW",
```

```
        DLL_FREQUENCY_MODE => "LOW",
        DUTY_CYCLE_CORRECTION => TRUE,
        FACTORY_JF => x"C080",
        PHASE_SHIFT => 0,
        STARTUP_WAIT => FALSE)
    port map (CLKFB=> GND_BIT,
        CLKIN=> CLKIN_IBUFG,
        DSSEN=>GND_BIT,
        PSCLK=>GND_BIT,
        PSEN=>GND_BIT,
        PSINCDEC=>GND_BIT,
        RST=> iRST,
        CLKDV=>open,
        CLKFX=> cCLK4,
        CLKFX180=>open,
        CLK0=> open,
        CLK2X=>open,
        CLK2X180=>open,
        CLK90=>open,
        CLK180=>open,
        CLK270=>open,
        LOCKED=> open,
        PSDONE=>open,
        STATUS=>open);

DCM_INST1_R56 : DCM
    generic map( CLK_FEEDBACK => "NONE",
        CLKDV_DIVIDE => 2.5,
```

```
    CLKFX_DIVIDE => 5,
    CLKFX_MULTIPLY => 6,
    CLKIN_DIVIDE_BY_2 => FALSE,
    CLKIN_PERIOD => CLKIN_PERIOD,
    CLKOUT_PHASE_SHIFT => "NONE",
    DESKEW_ADJUST => "SYSTEM_SYNCHRONOUS",
    DFS_FREQUENCY_MODE => "LOW",
    DLL_FREQUENCY_MODE => "LOW",
    DUTY_CYCLE_CORRECTION => TRUE,
    FACTORY_JF => x"C080",
    PHASE_SHIFT => 0,
    STARTUP_WAIT => FALSE)
port map (CLKFB=> GND_BIT,
    CLKIN=> CLKIN_IBUFG,
    DSSEN=>GND_BIT,
    PSCLK=>GND_BIT,
    PSEN=>GND_BIT,
    PSINCDEC=>GND_BIT,
    RST=> iRST,
    CLKDV=>open,
    CLKFX=> cCLK5,
    CLKFX180=>open,
    CLK0=> open,
    CLK2X=>open,
    CLK2X180=>open,
    CLK90=>open,
    CLK180=>open,
    CLK270=>open,
```

```
        LOCKED=> open,
        PSDONE=>open,
        STATUS=>open);

DCM_INST1_R67 : DCM
    generic map( CLK_FEEDBACK => "NONE",
        CLKDV_DIVIDE => 2.5,
        CLKFX_DIVIDE => 6,
        CLKFX_MULTIPLY => 7,
        CLKIN_DIVIDE_BY_2 => FALSE,
        CLKIN_PERIOD => CLKIN_PERIOD,
        CLKOUT_PHASE_SHIFT => "NONE",
        DESKEW_ADJUST => "SYSTEM_SYNCHRONOUS",
        DFS_FREQUENCY_MODE => "LOW",
        DLL_FREQUENCY_MODE => "LOW",
        DUTY_CYCLE_CORRECTION => TRUE,
        FACTORY_JF => x"C080",
        PHASE_SHIFT => 0,
        STARTUP_WAIT => FALSE)
    port map (CLKFB=> GND_BIT,
        CLKIN=> CLKIN_IBUFG,
        DSSEN=>GND_BIT,
        PSCLK=>GND_BIT,
        PSEN=>GND_BIT,
        PSINCDEC=>GND_BIT,
        RST=> iRST,
        CLKDV=>open,
        CLKFX=> cCLK6,
```

```
        CLKFX180=>open,
        CLK0=> open,
        CLK2X=>open,
        CLK2X180=>open,
        CLK90=>open,
        CLK180=>open,
        CLK270=>open,
        LOCKED=> open,
        PSDONE=>open,
        STATUS=>open);

DCM_INST1_R78 : DCM
    generic map( CLK_FEEDBACK => "NONE",
        CLKDV_DIVIDE => 2.5,
        CLKFX_DIVIDE => 7,
        CLKFX_MULTIPLY => 8,
        CLKIN_DIVIDE_BY_2 => FALSE,
        CLKIN_PERIOD => CLKIN_PERIOD,
        CLKOUT_PHASE_SHIFT => "NONE",
        DESKEW_ADJUST => "SYSTEM_SYNCHRONOUS",
        DFS_FREQUENCY_MODE => "LOW",
        DLL_FREQUENCY_MODE => "LOW",
        DUTY_CYCLE_CORRECTION => TRUE,
        FACTORY_JF => x"C080",
        PHASE_SHIFT => 0,
        STARTUP_WAIT => FALSE)
    port map (CLKFB=> GND_BIT,
        CLKIN=> CLKIN_IBUFG,
```

```
        DSSEN=>GND_BIT,
        PSCLK=>GND_BIT,
        PSEN=>GND_BIT,
        PSINCDEC=>GND_BIT,
        RST=> iRST,
        CLKDV=>open,
        CLKFX=> cCLK7,
        CLKFX180=>open,
        CLK0=> open,
        CLK2X=>open,
        CLK2X180=>open,
        CLK90=>open,
        CLK180=>open,
        CLK270=>open,
        LOCKED=> open,
        PSDONE=>open,
        STATUS=>open);

DCM_INST1_R89 : DCM
    generic map( CLK_FEEDBACK => "NONE",
        CLKDV_DIVIDE => 2.5,
        CLKFX_DIVIDE => 8,
        CLKFX_MULTIPLY => 9,
        CLKIN_DIVIDE_BY_2 => FALSE,
        CLKIN_PERIOD => CLKIN_PERIOD,
        CLKOUT_PHASE_SHIFT => "NONE",
        DESKEW_ADJUST => "SYSTEM_SYNCHRONOUS",
        DFS_FREQUENCY_MODE => "LOW",
```

```
        DLL_FREQUENCY_MODE => "LOW",
        DUTY_CYCLE_CORRECTION => TRUE,
        FACTORY_JF => x"C080",
        PHASE_SHIFT => 0,
        STARTUP_WAIT => FALSE)
    port map (CLKFB=> GND_BIT,
        CLKIN=> CLKIN_IBUFG,
        DSSEN=>GND_BIT,
        PSCLK=>GND_BIT,
        PSEN=>GND_BIT,
        PSINCDEC=>GND_BIT,
        RST=> iRST,
        CLKDV=>open,
        CLKFX=> cCLK8,
        CLKFX180=>open,
        CLK0=> open,
        CLK2X=>open,
        CLK2X180=>open,
        CLK90=>open,
        CLK180=>open,
        CLK270=>open,
        LOCKED=> open,
        PSDONE=>open,
        STATUS=>open);
-- ------------------------------
-- BUFGMUX
-- Selection de cCLK
BUFGMUX_INST1 : BUFGMUX
```

```
    port map (I0=>cCLK8,
        I1=>cCLK1,
        S=>RSelect(0),
        O=>CLKTmp1);
BUFGMUX_INST2 : BUFGMUX
    port map (I0=>cCLK2,
        I1=>cCLK3,
        S=>RSelect(0),
        O=>CLKTmp2);
BUFGMUX_INST3 : BUFGMUX
    port map (I0=>cCLK4,
        I1=>cCLK5,
        S=>RSelect(0),
        O=>CLKTmp3);
BUFGMUX_INST4 : BUFGMUX
    port map (I0=>cCLK6,
        I1=>cCLK7,
        S=>RSelect(0),
        O=>CLKTmp4);
BUFGMUX_INST5 : BUFGMUX
    port map (I0=>CLKTmp1,
        I1=>CLKTmp2,
        S=>RSelect(1),
        O=>CLKTmp5);
BUFGMUX_INST6 : BUFGMUX
    port map (I0=>CLKTmp3,
        I1=>CLKTmp4,
        S=>RSelect(1),
```

```
        O=>CLKTmp6);
BUFGMUX_INST7 : BUFGMUX
    port map (I0=>CLKTmp5,
        I1=>CLKTmp6,
        S=>RSelect(2),
        O=>icCLK);
-- ------------------------------
-- Outputs
dCLK <= CLKIN_IBUFG;
dLoad <= idload;
cCLK <= icCLK;
cLoad <= icload;
synchDone <= iSynch;

end architecture;
```

ANNEXE IV

COMPLEXITÉ ET DÉBIT DES DÉCODEURS À SEUIL

Tableau IV.1: Complexité et débit par itération du décodeur à seuil des codes PCDO $\{J = 8, \alpha_J = 139\}$, $RInt = 5$ bits, 11 étages de pipeline insérés

$\{0, 9, 25, 46, 95, 112, 126, 139\}$							
Complexité par itération					Délai	D_{it}	RDC_{it}
LUT	FF	MP†	Slices	C_{it} (kG)	critique (ns)	(Mbps)	(Mbps/kG)
677	563	1	635 (1%)	26.514	3.776	264.83	9.99

†: MULT18x18

Tableau IV.2: Complexité et débit par itération du décodeur à seuil des codes PCDO $\{J = 15, \alpha_J = 2932\}$, $RInt = 5$ bits, 12 étages de pipeline insérés

$\{0, 135, 176, 192, 205, 222, 243, 358, 526, 791, 1243, 1889, 2069, 2789, 2932\}$							
Complexité par itération					Délai	D_{it}	RDC_{it}
LUT	FF	MP†	Slices	C_{it} (kG)	critique (ns)	(Mbps)	(Mbps/kG)
4,512	1,515	1	4,379 (13%)	231.242	4.027	248.32	1.07

†: MULT18x18

Tableau IV.3: Complexité et débit par itération du décodeur à seuil des codes 3-CDO $\{R = 3/6, J = 6, \alpha_J = 67\}$, $RInt = 5$ bits, 10 étages de pipeline insérés

$$\left[\begin{array}{cc|cc|cc} 0 & 33 & 22 & 60 & 0 & 37 \\ 11 & 65 & 11 & 46 & 12 & 51 \\ 22 & 51 & 0 & 34 & 24 & 67 \end{array}\right]$$

Complexité par itération					Délai	D_{it}	RDC_{it}
LUT	FF	MP†	Slices	C_{it} (kG)	critique (ns)	(Mbps)	(Mbps/kG)
1,206	1,047	3	1,096 (3%)	49.128	3.801	789.27	16.06

†: MULT18x18

Tableau IV.4: Complexité et débit par itération du décodeur à seuil des codes 2-CDO $\{R = 2/4,\ J = 8,\ \alpha_J = 256\}$, $RInt = 5$ bits, 11 étages de pipeline insérés

$$\left[\begin{array}{cccc|cccc} 0 & 13 & 43 & 234 & 12 & 41 & 145 & 253 \\ 0 & 25 & 79 & 256 & 0 & 26 & 86 & 204 \end{array}\right]$$

Complexité par itération					Délai	D_{it}	RDC_{it}
LUT	FF	MP†	Slices	C_{it} (kG)	critique (ns)	(Mbps)	(Mbps/kG)
1,592	1,095	2	1,501 (4%)	67.557	3.777	529.52	7.84

†: MULT18x18

Tableau IV.5: Complexité et débit par itération du décodeur à seuil des codes 2-CDO $\{R = 2/4,\ J = 9,\ \alpha_J = 383\}$, $RInt = 5$ bits, 12 étages de pipeline insérés

$$\left[\begin{array}{ccccc|ccccc} 0 & 26 & 63 & 208 & 376 & 0 & 30 & 157 & 361 & -1 \\ 13 & 41 & 112 & 314 & -1 & 14 & 46 & 96 & 296 & 383 \end{array}\right]$$

Complexité par itération					Délai	D_{it}	RDC_{it}
LUT	FF	MP†	Slices	C_{it} (kG)	critique (ns)	(Mbps)	(Mbps/kG)
2,107	1,346	2	2,010 (6%)	91.223	3.778	529.38	5.80

†: MULT18x18

Tableau IV.6: Complexité et débit par itération du décodeur à seuil des codes 2-CPDO $\{R = 2/4,\ J = 8,\ \alpha_J = 100\}$, $RInt = 5$ bits, 11 étages de pipeline insérés

$$\left[\begin{array}{cccc|cccc} 0 & 13 & 42 & 77 & 12 & 40 & 72 & 84 \\ 0 & 25 & 55 & 92 & 0 & 26 & 57 & 100 \end{array}\right]$$

Complexité par itération					Délai	D_{it}	RDC_{it}
LUT	FF	MP†	Slices	C_{it} (kG)	critique (ns)	(Mbps)	(Mbps/kG)
1,285	1,065	2	1,164 (3%)	47.577	3.776	529.66	11.13

†: MULT18x18

Tableau IV.7: Complexité et débit par itération du décodeur à seuil des codes 3-CPDO $\{R = 3/6, J = 9, \alpha_J = 107\}$, $RInt = 5$ bits, 12 étages de pipeline insérés

$$\left[\begin{array}{ccc|ccc|ccc} 0 & 27 & 73 & 13 & 57 & 106 & 0 & 28 & 89 \\ 13 & 41 & 89 & 0 & 42 & 87 & 0 & 43 & 107 \\ 0 & 57 & 107 & 0 & 27 & 73 & 13 & 58 & 71 \end{array}\right]$$

Complexité par itération					Délai	D_{it}	RDC_{it}
LUT	FF	MP†	Slices	C_{it} (kG)	critique (ns)	(Mbps)	(Mbps/kG)
2,244	1,885	3	1,967 (5%)	75.930	3.778	794.07	10.46

†: MULT18x18

Tableau IV.8: Complexité et débit par itération du décodeur à seuil des codes 2-CDO $\{R = 2/3, J = 5, \alpha_J = 132\}$, $RInt = 5$ bits, 11 étages de pipeline insérés

$$\left[\begin{array}{ccccc} 0 & 12 & 38 & 96 & 132 \\ 0 & 25 & 55 & 78 & 112 \end{array}\right]$$

Complexité par itération					Délai	D_{it}	RDC_{it}
LUT	FF	MP†	Slices	C_{it} (kG)	critique (ns)	(Mbps)	(Mbps/kG)
1,019	825	2	951 (2%)	41.519	3.944	507.10	12.21

†: MULT18x18

Tableau IV.9: Complexité et débit par itération du décodeur à seuil des codes 2-CDO $\{R = 2/3, J = 9, \alpha_J = 1746\}$, $RInt = 5$ bits, 13 étages de pipeline insérés

$$\left[\begin{array}{ccccccccc} 0 & 28 & 59 & 94 & 140 & 316 & 530 & 1057 & 1690 \\ 14 & 43 & 75 & 114 & 252 & 386 & 765 & 1223 & 1746 \end{array}\right]$$

Complexité par itération					Délai	D_{it}	RDC_{it}
LUT	FF	MP†	Slices	C_{it} (kG)	critique (ns)	(Mbps)	(Mbps/kG)
4,859	2,205	2	4,691 (14%)	232.101	4.231	472.70	2.03

†: MULT18x18

ANNEXE V

PERFORMANCES D'ERREUR DE DSI DES CODES PCDO

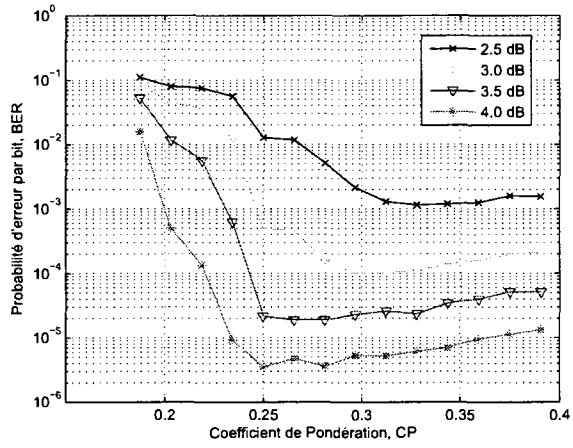

Figure V.1: BER en fonction de CP, CDO $\{J = 8, \alpha_J = 139\}$, $R = 1/2$, $RInt = 5$ bits, 8 itérations

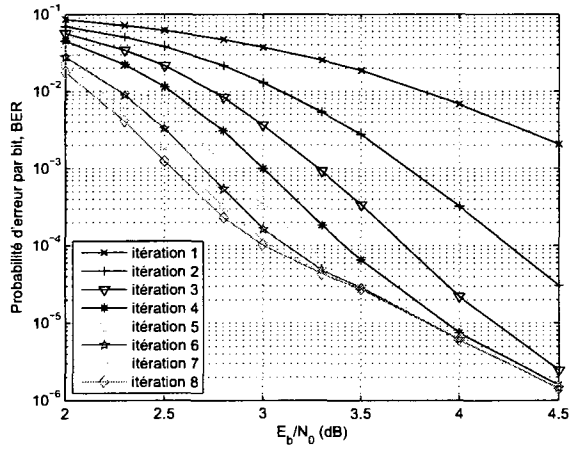

Figure V.2: BER en fonction de E_b/N_0, CDO $\{J = 8, \alpha_J = 139\}$, $R = 1/2$, $CP = 0.3125$, $RInt = 5$ bits, 8 itérations

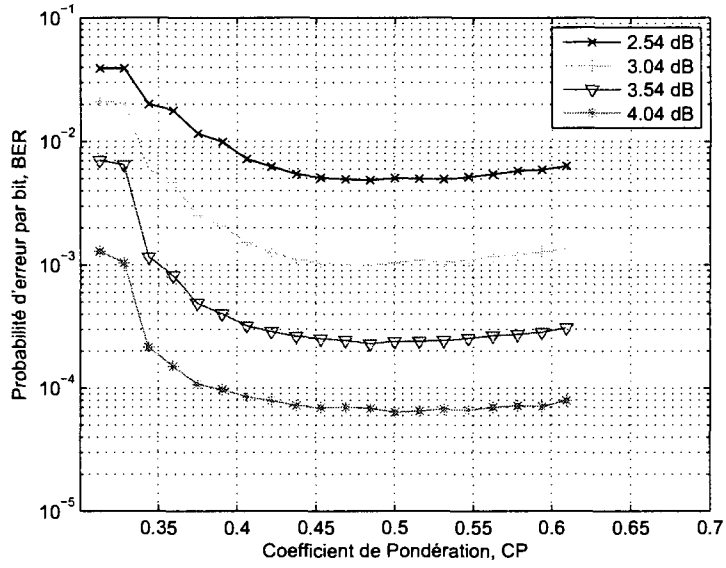

Figure V.3: BER en fonction de CP, PCDO $\{J = 8, \alpha_J = 139\}$, $R = 2/3$, $RInt = 5$ bits, 8 itérations

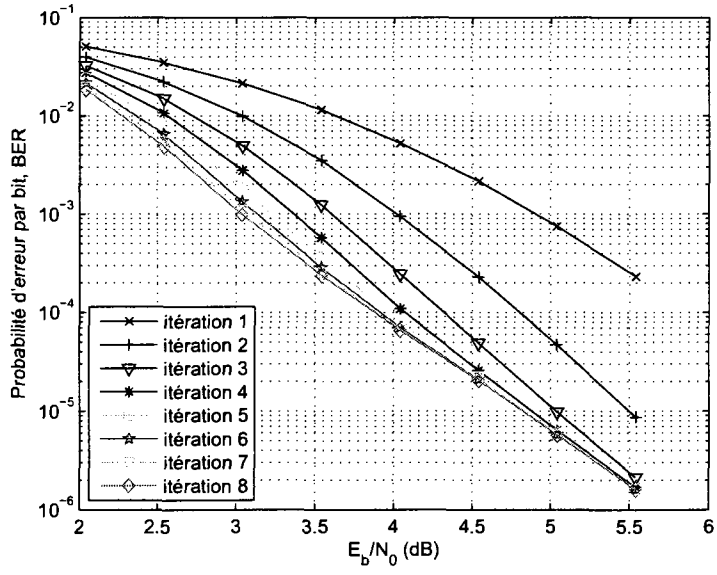

Figure V.4: BER en fonction de E_b/N_0, PCDO $\{J = 8, \alpha_J = 139\}$, $R = 2/3$, $CP = 0.484375$, $RInt = 5$ bits, 8 itérations

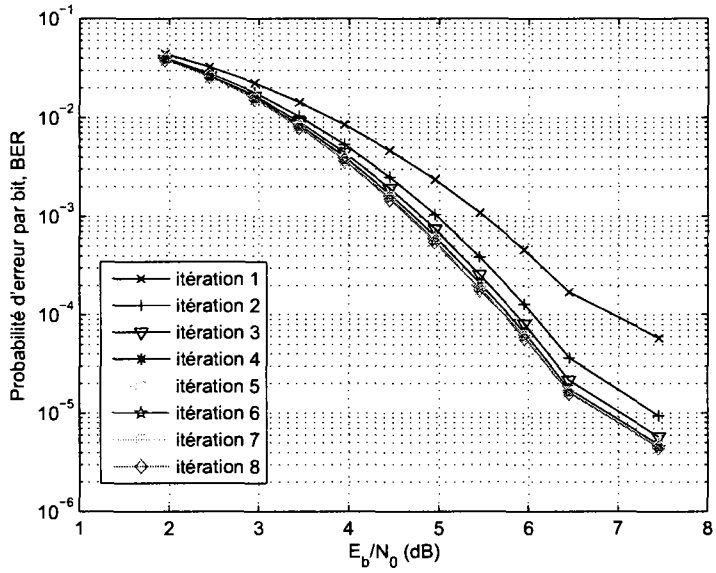

Figure V.5: BER en fonction de E_b/N_0, PCDO $\{J = 8, \alpha_J = 139\}$, $R = 4/5$, $CP = 0.75$, $RInt = 5$ bits, 8 itérations

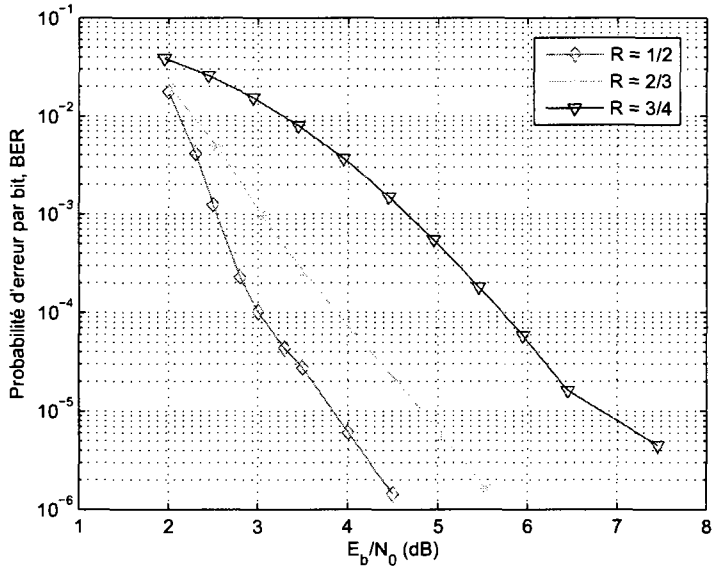

Figure V.6: BER en fonction de E_b/N_0, PCDO à taux compatibles $\{J = 8, \alpha_J = 139\}$, $R = 1/2, 2/3$ et $4/5$, $RInt = 5$ bits, 8 itérations

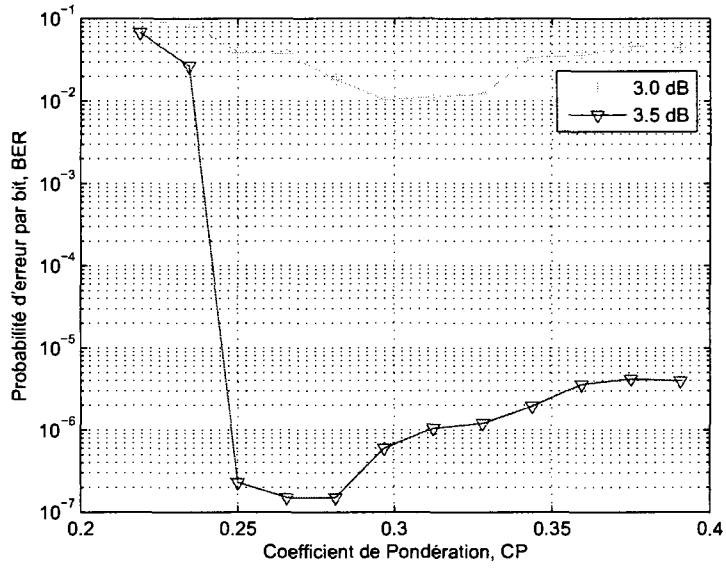

Figure V.7: BER en fonction de CP, CDO $\{J = 15, \alpha_J = 2932\}$, $R = 1/2$, $RInt = 5$ bits, 8 itérations

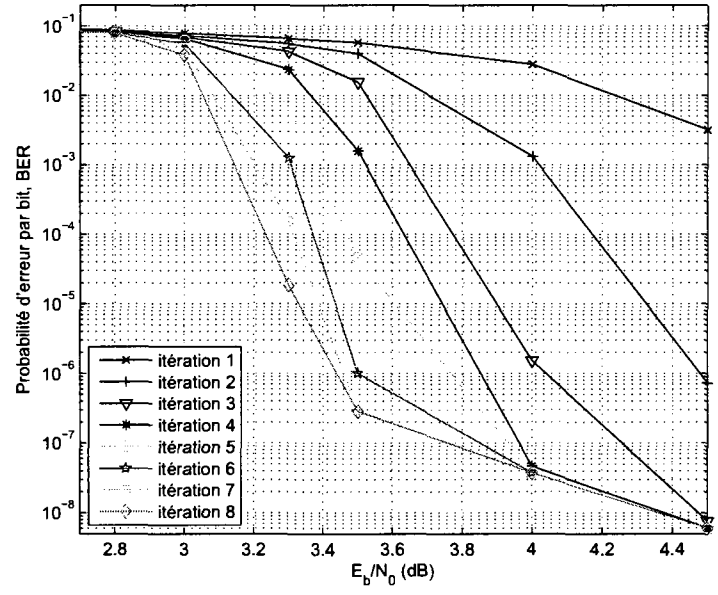

Figure V.8: BER en fonction de E_b/N_0, CDO $\{J = 15, \alpha_J = 2932\}$, $R = 1/2$, $CP = 0.25$, $RInt = 5$ bits, 8 itérations

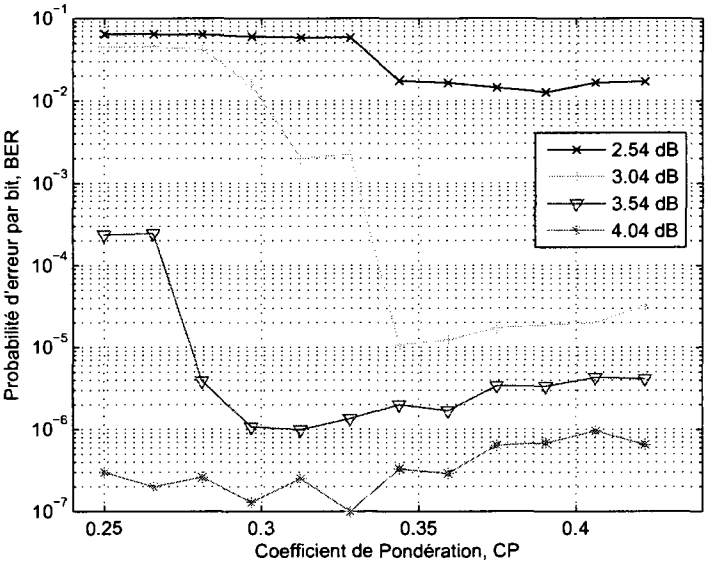

Figure V.9: BER en fonction de CP, PCDO $\{J = 15, \alpha_J = 2932\}$, $R = 2/3$, $RInt = 5$ bits, 8 itérations

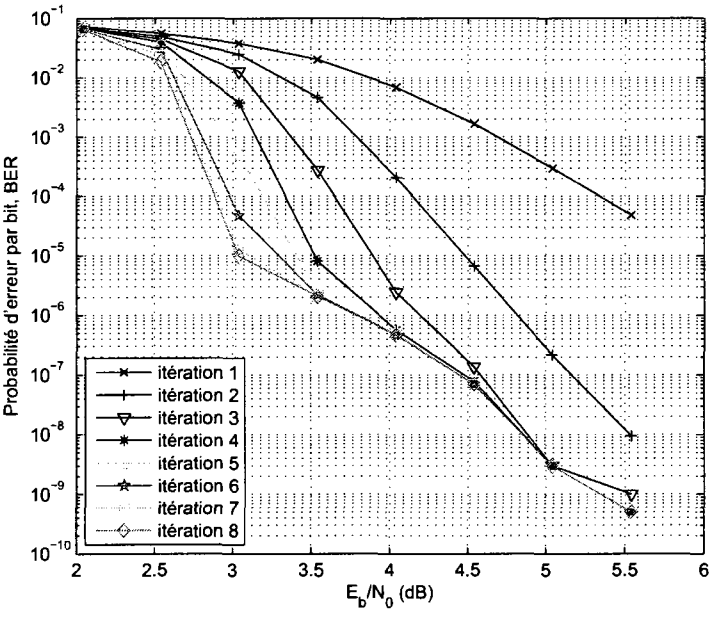

Figure V.10: BER en fonction de E_b/N_0, PCDO $\{J = 15, \alpha_J = 2932\}$, $R = 2/3$, $CP = 0.34375$, $RInt = 5$ bits, 8 itérations, 10 Gbits simulés

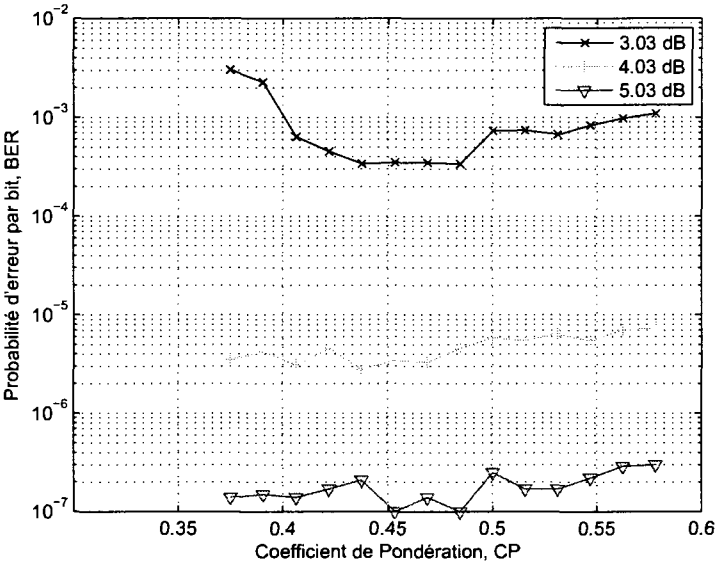

Figure V.11: BER en fonction de CP, PCDO $\{J = 15, \alpha_J = 2932\}$, $R = 3/4$, $RInt = 5$ bits, 8 itérations

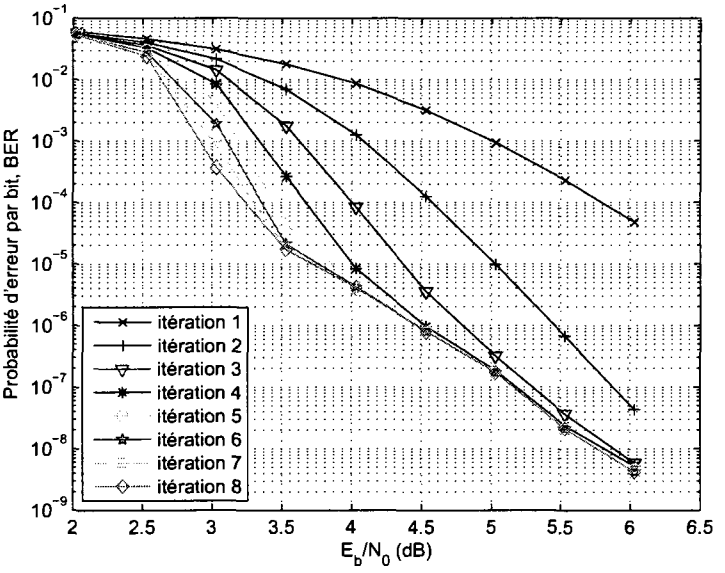

Figure V.12: BER en fonction de E_b/N_0, PCDO $\{J = 15, \alpha_J = 2932\}$, $R = 3/4$, $CP = 0.453125$, $RInt = 5$ bits, 8 itérations, 10 Gbits simulés

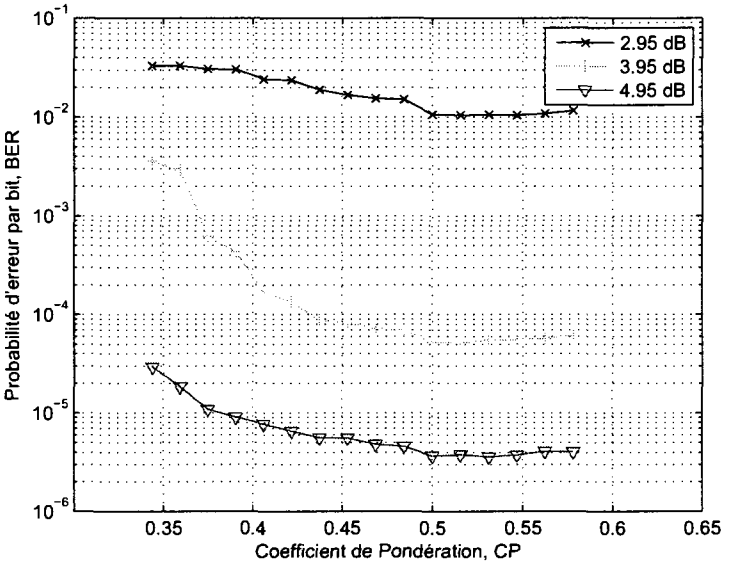

Figure V.13: BER en fonction de CP, PCDO $\{J = 15, \alpha_J = 2932\}$, $R = 4/5$, $RInt =$ 5 bits, 8 itérations

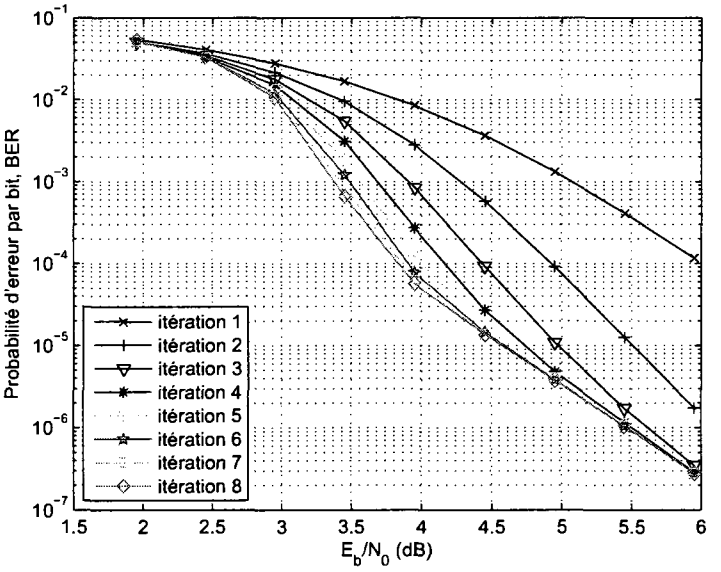

Figure V.14: BER en fonction de E_b/N_0, PCDO $\{J = 15, \alpha_J = 2932\}$, $R = 4/5$, $CP = 0.546875$, $RInt = 5$ bits, 8 itérations

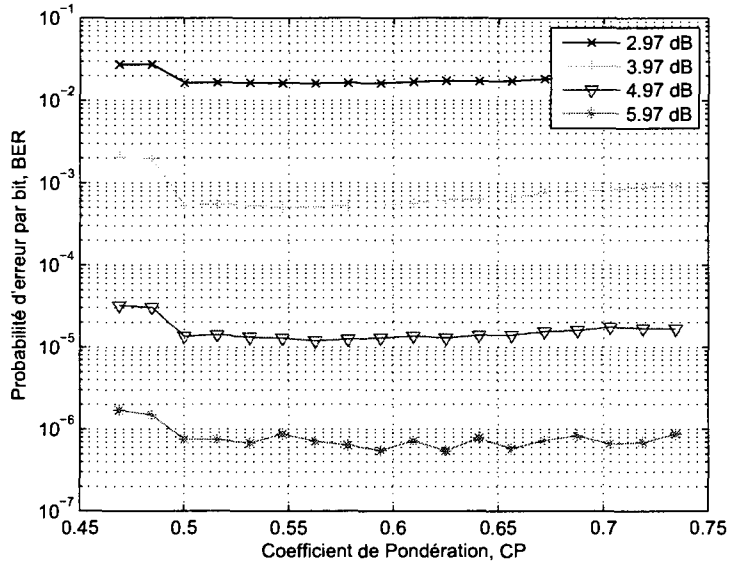

Figure V.15: BER en fonction de CP, PCDO $\{J = 15, \alpha_J = 2932\}$, $R = 5/6$, $RInt =$ 5 bits, 8 itérations

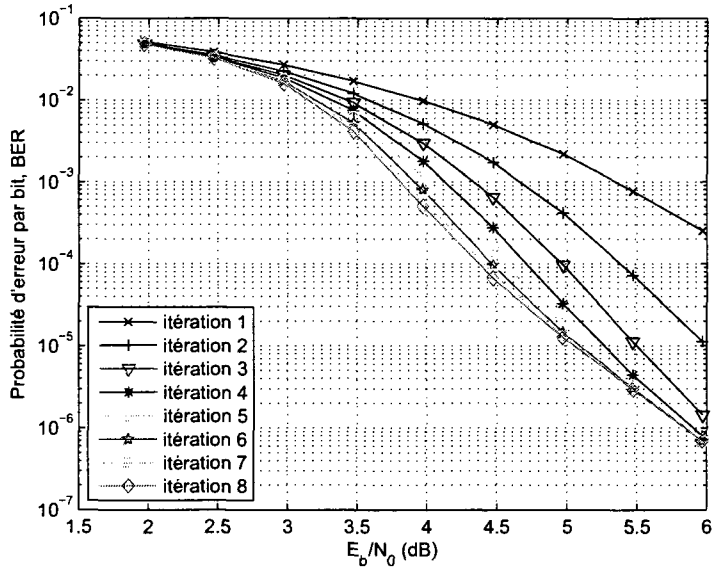

Figure V.16: BER en fonction de E_b/N_0, PCDO $\{J = 15, \alpha_J = 2932\}$, $R = 5/6$, $CP = 0.546875$, $RInt = 5$ bits, 8 itérations

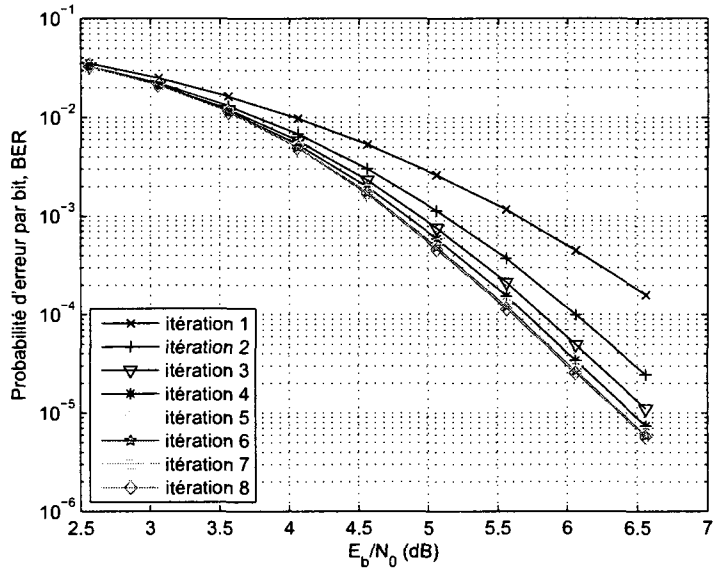

Figure V.17: BER en fonction de E_b/N_0, PCDO $\{J = 15, \alpha_J = 2932\}$, $R = 7/8$, $CP = 0.75$, $RInt = 5$ bits, 8 itérations

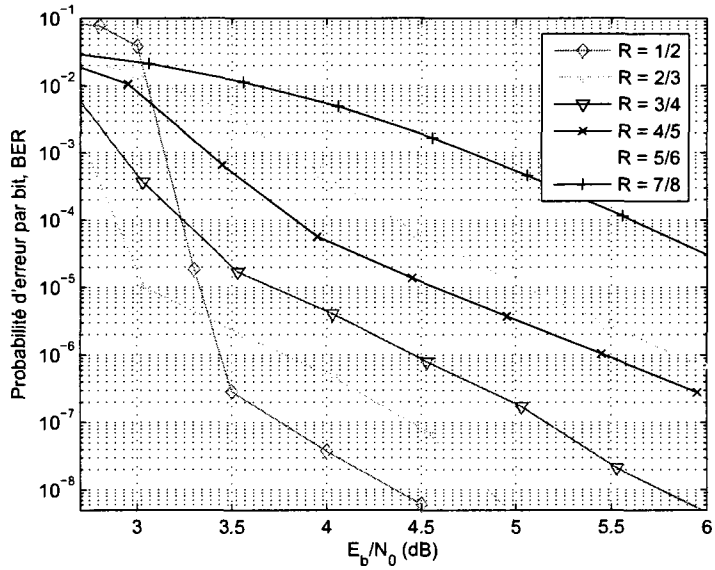

Figure V.18: BER en fonction de E_b/N_0, PCDO à taux compatibles $\{J = 15, \alpha_J = 2932\}$, $R = 1/2, 2/3, 3/4, 4/5, 5/6$ et $7/8$, $RInt = 5$ bits, 8 itérations

ANNEXE VI

PERFORMANCES D'ERREUR DE DSI DES CODES M-CDO ET M-CPDO

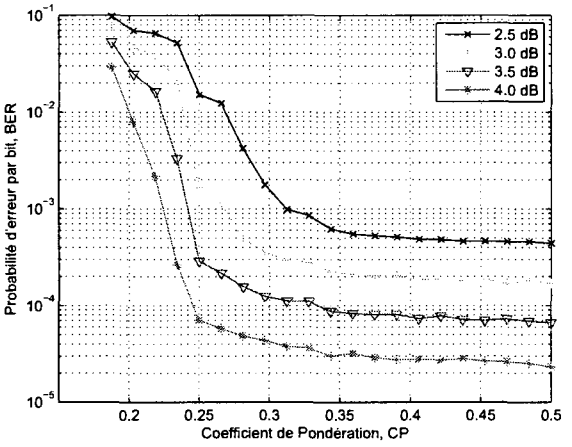

Figure VI.1: BER en fonction de CP, 3-CDO $\{R = 3/6,\ J = 6,\ \alpha_J = 67\}$, $RInt = 5$ bits, 8 itérations

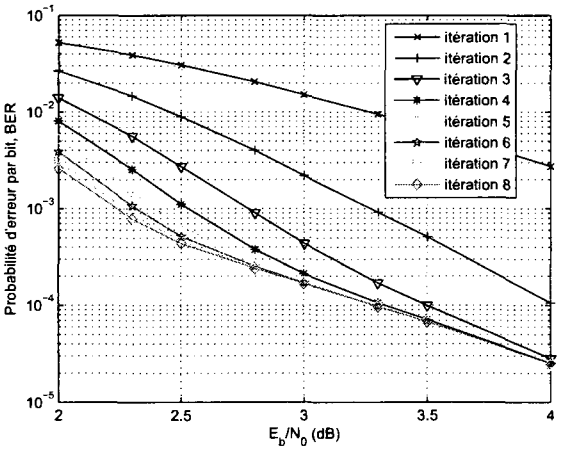

Figure VI.2: BER en fonction de E_b/N_0, 3-CDO $\{R = 3/6,\ J = 6,\ \alpha_J = 67\}$, $CP = 0.5$, $RInt = 5$ bits, 8 itérations

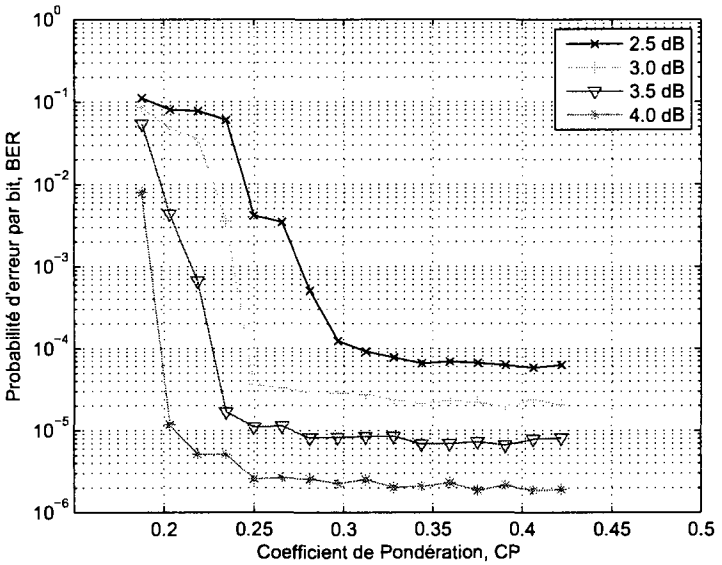

Figure VI.3: BER en fonction de CP, 2-CDO $\{R = 2/4,\ J = 8,\ \alpha_J = 256\}$, $RInt = 5$ bits, 8 itérations

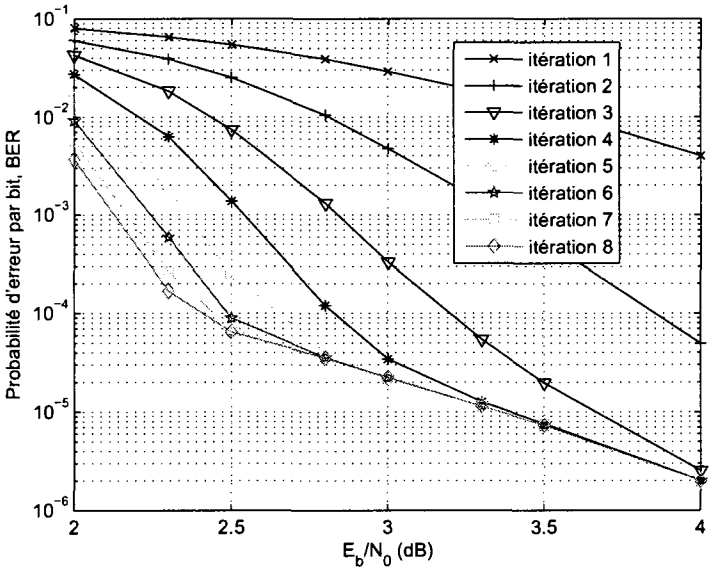

Figure VI.4: BER en fonction de E_b/N_0, 2-CDO $\{R = 2/4,\ J = 8,\ \alpha_J = 256\}$, $CP = 0.375$, $RInt = 5$ bits, 8 itérations

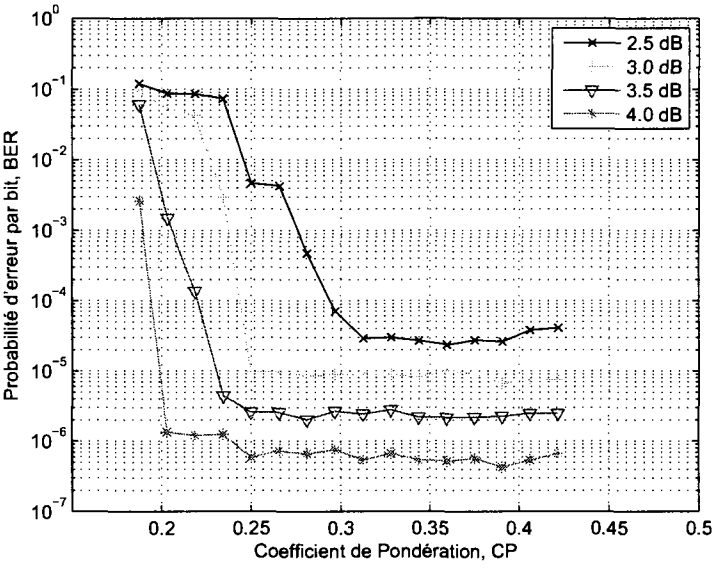

Figure VI.5: BER en fonction de CP, 2-CDO $\{R = 2/4,\ J = 9,\ \alpha_J = 383\}$, $RInt = 5$ bits, 8 itérations

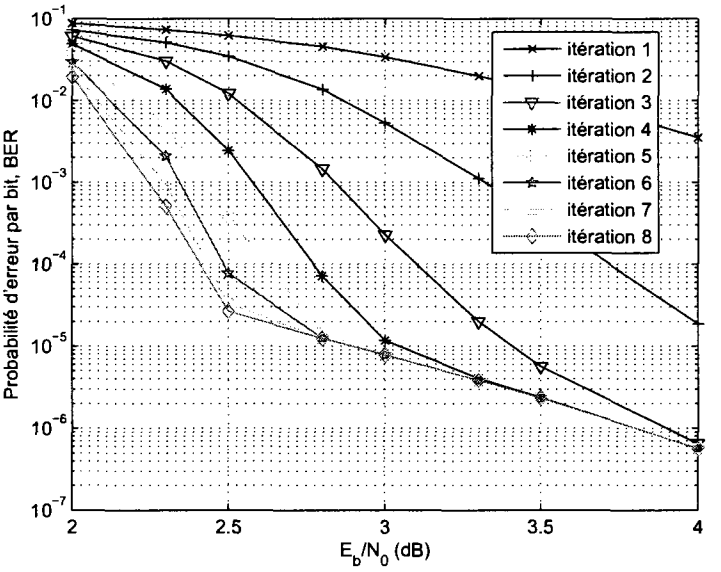

Figure VI.6: BER en fonction de E_b/N_0, 2-CDO $\{R = 2/4,\ J = 9,\ \alpha_J = 383\}$, $CP = 0.375$, $RInt = 5$ bits, 8 itérations

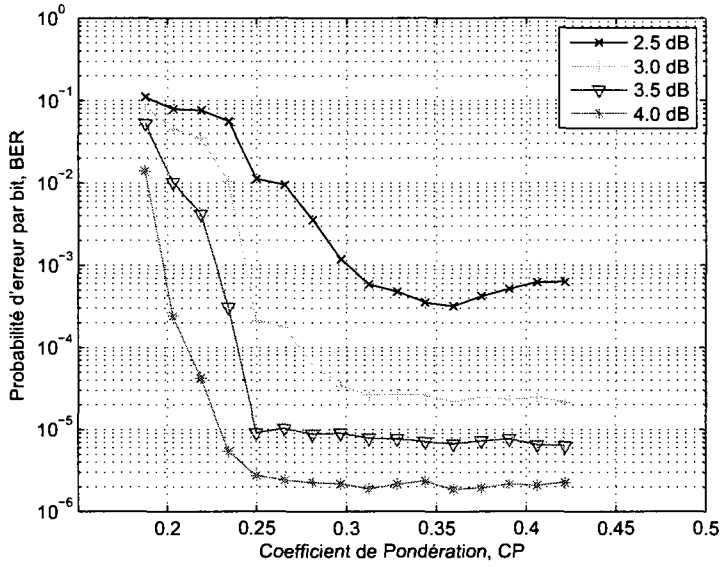

Figure VI.7: BER en fonction de CP, 2-CPDO $\{R = 2/4, J = 8, \alpha_J = 100\}$, $RInt = 5$ bits, 8 itérations

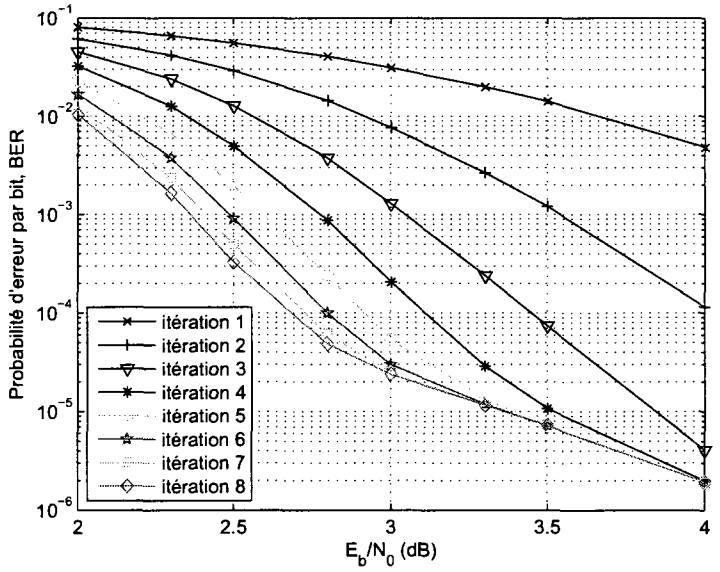

Figure VI.8: BER en fonction de E_b/N_0, 2-CPDO $\{R = 2/4, J = 8, \alpha_J = 100\}$, $CP = 0.359375$, $RInt = 5$ bits, 8 itérations

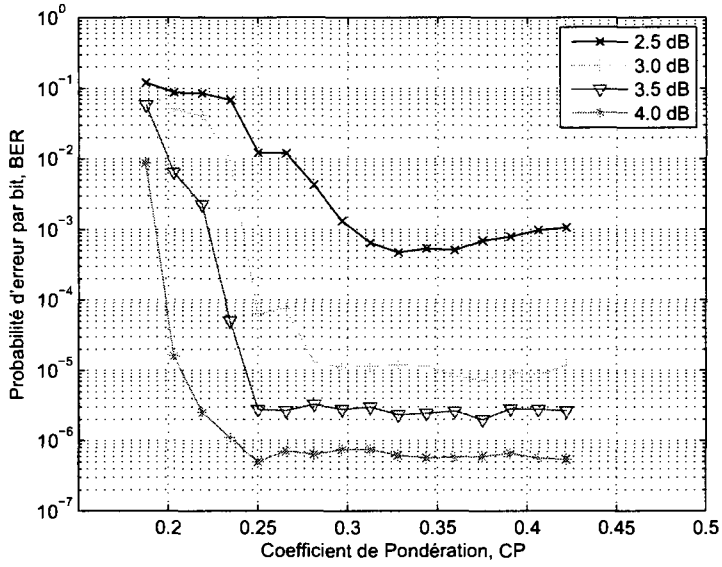

Figure VI.9: BER en fonction de CP, 3-CPDO $\{R = 3/6, J = 9, \alpha_J = 107\}$, $RInt = 5$ bits, 8 itérations

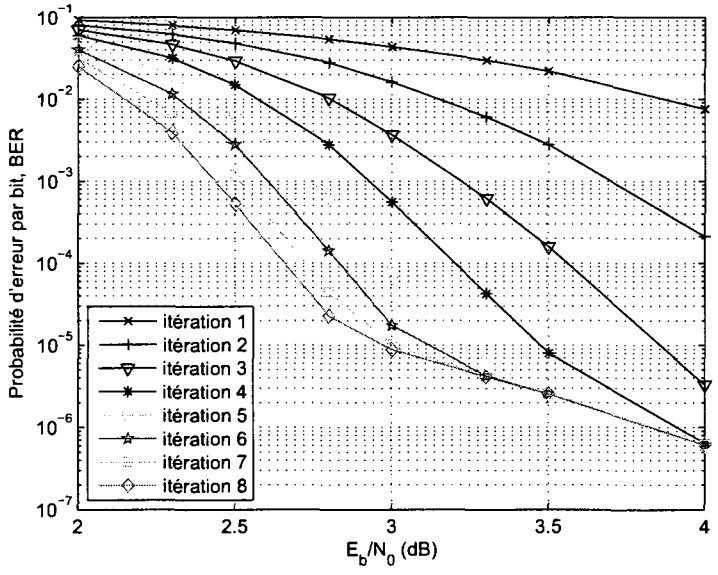

Figure VI.10: BER en fonction de E_b/N_0, 3-CPDO $\{R = 3/6, J = 9, \alpha_J = 107\}$, $CP = 0.34375$, $RInt = 5$ bits, 8 itérations

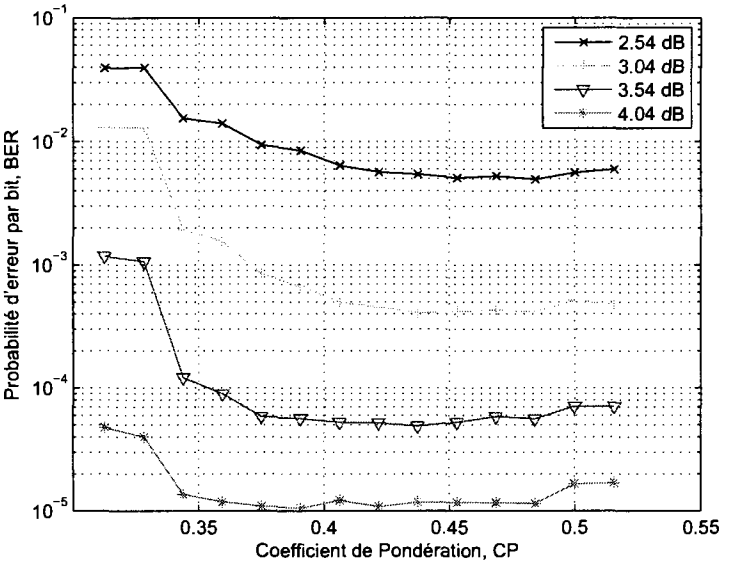

Figure VI.11: BER en fonction de CP, 2-CDO $\{R = 2/3, J = 5, \alpha_J = 132\}$, $RInt = 5$ bits, 6 itérations

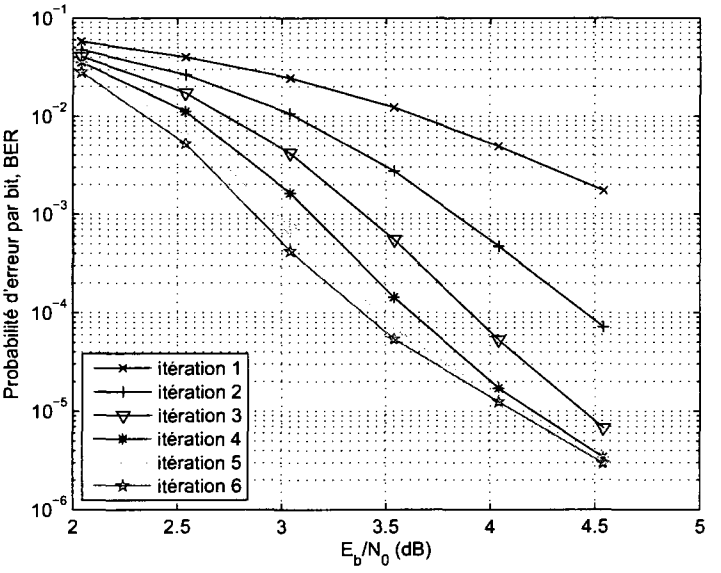

Figure VI.12: BER en fonction de E_b/N_0, 2-CDO $\{R = 2/3, J = 5, \alpha_J = 132\}$, $CP = 0.453125$, $RInt = 5$ bits, 6 itérations

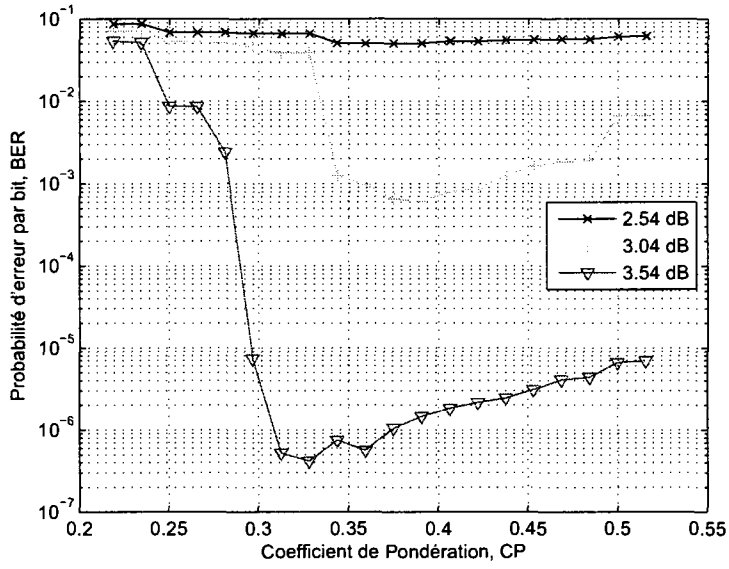

Figure VI.13: BER en fonction de CP, 2-CDO $\{R = 2/3,\ J = 9,\ \alpha_J = 1746\}$, $RInt = 5$ bits, 6 itérations

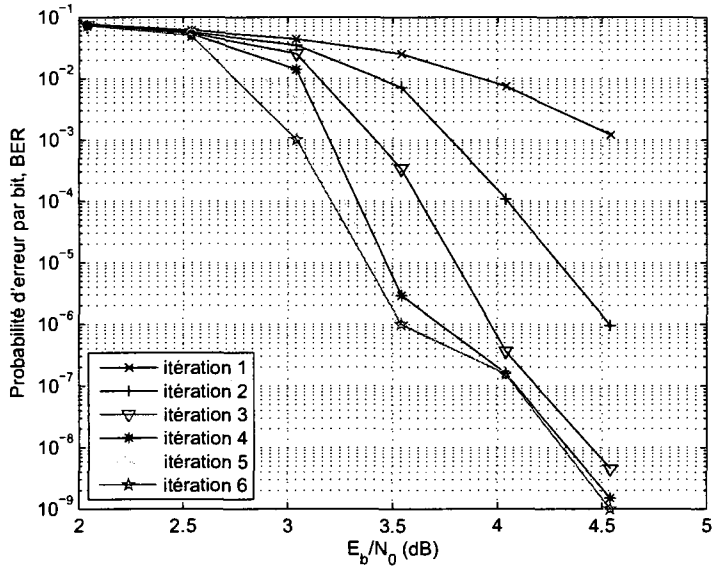

Figure VI.14: BER en fonction de E_b/N_0, 2-CDO $\{R = 2/3,\ J = 9,\ \alpha_J = 1746\}$, $CP = 0.359375$, $RInt = 5$ bits, 6 itérations, 10 Gbits simulés