

# Conception d'architectures rapides pour codes convolutifs en télécommunications : application aux turbo-codes

Mohamed El Amine M'Sir

▶ **To cite this version:**

Mohamed El Amine M'Sir. Conception d'architectures rapides pour codes convolutifs en télécommunications : application aux turbo-codes. Autre. Université Paul Verlaine - Metz, 2003. Français. NNT : 2003METZ015S . tel-01749919

**HAL Id: tel-01749919**
**https://hal.univ-lorraine.fr/tel-01749919**

Submitted on 29 Mar 2018

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

AVERTISSEMENT

Ce document est le fruit d'un long travail approuvé par le jury de soutenance et mis à disposition de l'ensemble de la communauté universitaire élargie.

Il est soumis à la propriété intellectuelle de l'auteur. Ceci implique une obligation de citation et de référencement lors de l'utilisation de ce document.

D'autre part, toute contrefaçon, plagiat, reproduction illicite encourt une poursuite pénale.

Contact : ddoc-theses-contact@univ-lorraine.fr

LIENS

Code de la Propriété Intellectuelle. articles L 122. 4
Code de la Propriété Intellectuelle. articles L 335.2- L 335.10
http://www.cfcopies.com/V2/leg/leg_droi.php
http://www.culture.gouv.fr/culture/infos-pratiques/droits/protection.htm

LABORATOIRE INTERFACES CAPTEURS
ET MICRO-ÉLECTRONIQUE

École Doctorale IAEM - Lorraine
Département de Formation Doctorale Électronique - Électrotechnique

BIBLIOTHEQUE UNIVERSITAIRE - METZ
N° inv. 20030455
Cote S/Mz 03/15
Loc

# THÈSE

Présentée à l'Université de Metz pour l'obtention du diplôme de
Docteur de l'Université de Metz
Discipline : Électronique

# CONCEPTION D'ARCHITECTURES RAPIDES POUR CODES CONVOLUTIFS EN TÉLÉCOMMUNICATIONS : APPLICATION AUX TURBO-CODES

Par

**Mohamed El Amine M'SIR**

Soutenue le 19 Septembre 2003 devant le jury composé de :

| | | |
|---|---|---|
| A. DANDACHE | Professeur à l'Université de Metz | Directeur de thèse |
| S. J. PIESTRAK | Professeur à l'Université de Wroclaw (Pologne) | Rapporteur |
| O. SENTIEYS | Professeur à l'ENSSAT de Lannion | Rapporteur |
| B. LEPLEY | Professeur à l'Université de Metz | Examinateur |
| F. MONTEIRO | Maître de conférence à l'Université de Metz | Examinateur |
| P. ADDE | Directeur d'études à l'ENST de Bretagne | Membre invité |
| P. JEAN | Ingénieur 2MG Communication | Membre invité |

BIBLIOTHEQUE UNIVERSITAIRE DE METZ
022 486107 2

LABORATOIRE INTERFACES CAPTEURS
ET MICRO-ÉLECTRONIQUE

École Doctorale IAEM - Lorraine
Département de Formation Doctorale Électronique - Électrotechnique

# THÈSE

Présentée à l'Université de Metz pour l'obtention du diplôme de
Docteur de l'Université de Metz
Discipline : Électronique

# CONCEPTION D'ARCHITECTURES RAPIDES POUR CODES CONVOLUTIFS EN TÉLÉCOMMUNICATIONS : APPLICATION AUX TURBO-CODES

Par

**Mohamed El Amine M'SIR**

Soutenue le 19 Septembre 2003 devant le jury composé de :

| | | |
|---|---|---|
| A. DANDACHE | Professeur à l'Université de Metz | Directeur de thèse |
| S. J. PIESTRAK | Professeur à l'Université de Wroclaw (Pologne) | Rapporteur |
| O. SENTIEYS | Professeur à l'ENSSAT de Lannion | Rapporteur |
| B. LEPLEY | Professeur à l'Université de Metz | Examinateur |
| F. MONTEIRO | Maître de conférence à l'Université de Metz | Examinateur |
| P. ADDE | Directeur d'études à l'ENST de Bretagne | Membre invité |
| P. JEAN | Ingénieur 2MG Communication | Membre invité |

LICM - 07 Rue Marconi, Technopôle, 57070 Metz, France
Tel : +33.(0)387.54.73.06/11 - Fax : +33.(0)387.54.73.07

# Avant-propos

Cette thèse a été effectuée à l'occasion d'un contrat C.I.F.R.E (Conventions Industrielles de Formation par la REcherche) entre le laboratoire L.I.C.M (Laboratoire Interface Capteurs et Micro-électronique) de l'Université de Metz et la société 2MG Communication avec l'aide du Conseil Régional de Lorraine. Nous Remercions tout particulièrement l'A.N.R.T (Association National de la Recherche Technique) et la Région Lorraine pour leur soutien financier.

Je dédie ce mémoire à :

- *Ma tendre mère.*
- *Mon cher père.*
- *Mon épouse bien aimée.*
- *Mes deux chères soeurs.*

# Remerciements

Je tiens avant tout à remercier :

- Monsieur **Bernard LEPLEY**, Directeur du Laboratoire LICM pour m'avoir accueilli au sein de son laboratoire. Qu'il trouve dans ces quelques lignes toute ma reconnaissance.
- Monsieur **Stanislaw J. PIESTRAK**, Professeur à l'Université de Wroclaw (Pologne) pour avoir accepté d'être rapporteur de ce travail.
- Monsieur **Olivier SENTIEYS**, Professeur à l'ENSSAT de Lannion pour avoir accepté d'être rapporteur de ce travail.
- Monsieur **Patrick ADDE**, Directeur d'études à l'ENST de Bretagne pour avoir accepté d'être membre de ce jury.
- Monsieur **Abbas DANDACHE**, Professeur à l'Université de Metz pour avoir encadré mes travaux, facilité mon intégration ainsi que pour son soutien moral et logistique. Qu'il trouve dans ces quelques lignes ma profonde gratitude.
- Monsieur **Fabrice MONTEIRO**, Maitre de conférence à l'Université de Metz pour avoir encadré mes travaux ainsi que pour sa gentillesse, ses conseils avisés et tout le temps qu'il m'a consacré lors de ce travail. Qu'il trouve dans ces quelques lignes ma profonde sympathie et toute ma reconnaissance.
- Monsieur **Philippe JEAN**, pour m'avoir accueilli au sein de sa société 2MG Communication. Je tiens à le remercier pour sa sympathie et toute l'aide apportée lors de ce travail.
- Enfin je n'oublierai pas toutes les personnes, qui par leurs actions, gestes, paroles ou écoutes m'ont soutenu tout au long de cette thèse. Je citerai tout particulièrement : **Camille**, **Abbas**, **Etienne**, **Jean-François**, **Rémy**, **Hervé** et **Gérald**.

# Résumé

En télécommunications, l'intégration des services et la diversité des données échangées (voix, vidéo, cellules ATM, etc.) exigent des systèmes de plus en plus rapides pour traiter des volumes d'information en augmentation constante, tout en conservant des délais d'attente les moins contraignants possibles pour l'utilisateur. ces deux contraintes liées au traitement de l'information s'ajoute la nécessité de protéger les données émises dans des canaux soumis à des bruits perturbateurs. De la qualité du service (QoS) rendu en termes de communication, quelles que soient les conditions de transmission, se dégagent deux mots clés : fiabilité et rapidité. Le travail de cette thèse fait partie d'un projet global mené dans le laboratoire LICM concernant la conception architecturale d'une chaîne de transmission à haut débit et à faible coût. Ce travail se situe dans le cadre d'un contrat CIFRE entre le laboratoire LICM et la société 2MG Communication. L'objectif est de concevoir une architecture rapide pour un système de codage de faible coût et à haut débit permettant une protection optimale des données.

Une étude approfondie sur les stratégies de codage canal dans le domaine des télécommunications a été effectuée. Il en ressort que les stratégies FEC (*Forward Error Correction*) offrent les meilleurs compromis débit/protection. L'emploi de codes convolutifs est l'une des approches FEC les plus répandues : la possibilité de les utiliser, soit comme entités indépendantes, soit dans des structures concaténées avec d'autres codes, soit encore comme éléments constituants de turbo-codes (techniques de codage offrant actuellement les meilleurs niveaux de protection), les rend très attractifs.

A partir de ce constat, quatre types d'architecture de codeurs convolutifs ainsi qu'une architecture de turbo-codeur ont été développés au niveau RTL et validés sur FPGA. Les résultats obtenus sont compris, pour le codeur convolutif haut débit, entre 1 et 7.45 Gbits/s pour des degrés de parallélisme 8 et 32, respectivement. Dans le cas du turbo-code, l'implantation d'un turbo-codeur à deux niveaux avec un rendement $1/3$ a permis d'atteindre des débits compris entre 1 et 2 Gbits/s et entre 3 et 5 Gbits/s pour des degrés de parallélisme de 8 et 32, respectivement. L'étude a été complétée par le développement d'un décodeur de syndrome pour codes convolutifs, exploitant l'architecture parallèle du codeur haut débit, qui a permis d'atteindre des débits de 4.78 Gbits/s pour un rendement de $1/2$.

**Mots clés :** transmission haut débit, codage canal, codes convolutifs, turbo-codes, architectures parallèles, modélisation RTL.

# Abstract

In the telecommunication area, integrating services, exchanging ever increasing volumes of heterogeneous data and preserving acceptable delays to end users, are requisites setting the need for faster and faster systems. Moreover, data must be protected against modifications due to the noise affecting the transmission channels. From the Quality of Service (QoS) of the communication, whatever the transmition conditions are, two keywords can be highlighted: reliability and throughput. The work presented here is a part of a larger project realized within the LICM laboratory concerning the architectural conception of a high-throughput and low-cost data transmission system. It has been carried out as a collaborative work between the LICM laboratory and the 2MG Communication company. The goal is to design a fast architecture for a low-cost and high-speed encoding and decoding system allowing for optimal data protection.

A thorough study of channel coding strategies in the telecommunication domain has been done. FEC strategies (Forward Error Correction) emerged as those providing the best troughput/protection trade-off. Within FEC, convolutional codes are among the most employed: the possibility to use them, either as independent codes, or in concatenated structures with other codes, or even as a constituent parts of turbo-codes (offering presently the highest levels of data protection) makes them very attractive, which motivates this research work.

Hence, four architectural types of convolutional encoders and one turbo-encoder architecture have been designed in VHDL at the RTL level and validated on FPGAs. The results obtained for the convolutional encoders are within 1 and 7.45 Gbits/s with parallelism levels of 8 and 32, respectively. Concerning the turbo-code, a 1/3-rate split-level parallel turbo-encoder has been implemented, allowing rates within 1 and 2 Gbits/s and within 3 and 5 Gbits/s to be achieved with parallelism levels of 8 and 32, respectively. The study was completed with the implementation of a syndrome decoder based on the parallel architecture of the high-speed convolutional encoder. Data rates up to 4.78 Gbits/s have been achieved with a 1/2-rate decoder.

**Key words:** high speed transmission, channel encoding, convolutional codes, turbo-codes, parallel architectures, RTL modeling.

# Table des matières

**Résumé** v

**Abstract** vii

**INTRODUCTION GÉNÉRALE** 9

**I. ETAT DE L'ART ET ÉTUDE THÉORIQUE** 15

**1 Introduction au codage canal** 15

1.1 Introduction . . . 15
1.2 Mesure quantitative de l'information . . . 16
1.3 Définition de l'entropie et de l'information mutuelle . . . 17
1.3.1 Définition de l'entropie . . . 17
1.3.2 Définition de l'information mutuelle . . . 18
1.3.3 Débit d'information . . . 19
1.4 Canal de transmission . . . 19
1.4.1 Canal BSC . . . 20
1.4.2 Canal AWGN . . . 21
1.4.3 Modélisation des erreurs . . . 23
1.5 Théorème fondamental du codage canal . . . 24
1.6 Chaîne de transmission . . . 25
1.6.1 Source d'information . . . 25
1.6.2 Codage source . . . 26
1.6.3 Codage canal . . . 26
1.6.4 Modulation . . . 26
1.6.5 Canal de transmission . . . 27
1.6.6 Démodulation . . . 27
1.6.7 Décodage canal . . . 29

1.6.8 Décodage source . . . . . 29
1.7 Stratégies de codage canal . . . . . 29
1.7.1 Stratégies ARQ . . . . . 29
1.7.2 Stratégies FEC . . . . . 31
1.7.3 Stratégies hybride ARQ/FEC . . . . . 31
1.7.4 Comparaison des stratégies de codage canal . . . . . 31
1.8 Techniques de codage canal FEC . . . . . 32
1.8.1 Codes blocs . . . . . 32
1.8.2 Codes convolutifs . . . . . 33
1.8.3 Modulation codée en treillis . . . . . 36
1.8.4 Codes entrelacés . . . . . 37
1.8.5 Codes concaténés . . . . . 37
1.8.6 Turbo-codes . . . . . 38
1.9 Comparatifs de quelques systèmes de codage FEC . . . . . 40
1.10 Conclusion . . . . . 41

**2 Codes convolutifs : application aux turbo-codes** **43**
2.1 Introduction . . . . . 43
2.2 Codes blocs . . . . . 44
2.2.1 Capacité de détection et correction d'un code . . . . . 44
2.2.2 Codes cycliques . . . . . 45
2.2.3 Code systématique . . . . . 48
2.3 Codes convolutifs . . . . . 50
2.3.1 Représentation des codes convolutifs . . . . . 51
2.3.2 Distance minimale d'un code convolutif . . . . . 54
2.3.3 Période maximale du codeur . . . . . 54
2.3.4 Codeur convolutif de rendement 1/n . . . . . 54
2.3.5 Poinçonnage (ou perforation) . . . . . 55
2.3.6 Codes convolutifs récursifs et systématiques . . . . . 56
2.4 Turbo-codes . . . . . 58
2.4.1 Turbo-codes de type série . . . . . 58
2.4.2 Turbo codes de type parallèle . . . . . 59
2.5 Performances d'un turbo-code . . . . . 60
2.5.1 Evaluation des performances par la borne de l'union . . . . . 60
2.5.2 Calcul du paramètre $A^{\mathbf{C}}_{w,d}$ . . . . . 61
2.6 Rôle des codes constituants et de l'entrelaceur . . . . . 64
2.7 Choix des codes constituants . . . . . 65

2.8 Matrices d'entrelacements . . . . . 66
2.8.1 Entrelacement périodique . . . . . 67
2.8.2 Entrelacement aléatoire . . . . . 68
2.8.3 Entrelacement pseudo-aléatoire . . . . . 69
2.9 Terminaison du treillis d'un turbo-code . . . . . 70
2.10 Perforation de la sortie d'un turbo-code . . . . . 70
2.11 Conclusion . . . . . 71

**3 Algorithmes de décodage itératif des turbo-codes 73**
3.1 Introduction . . . . . 73
3.2 Décodage SISO . . . . . 74
3.3 Algorithme de décodage SISO . . . . . 76
3.3.1 Algorithme de Viterbi . . . . . 77
3.3.2 Algorithme de Viterbi à sortie souple SOVA . . . . . 80
3.3.3 Algorithme SOVA amélioré . . . . . 82
3.3.4 Algorithme MAP . . . . . 82
3.3.5 Algorithme Log MAP . . . . . 85
3.3.6 Algorithme Max-Log MAP . . . . . 87
3.4 Comparaison des algorithmes de décodage SISO . . . . . 88
3.5 Etude architecturale d'un décodeur de Viterbi . . . . . 89
3.6 Quelques réalisations de turbo-codeur . . . . . 94
3.7 Conclusion . . . . . 98

**II. ETUDE ARCHITECTURALE ET IMPLANTATION 101**

**4 Architectures systoliques pour la convolution 101**
4.1 Introduction . . . . . 101
4.2 Avantages et inconvénients des architectures systoliques . . . . . 102
4.2.1 Principaux avantages . . . . . 102
4.2.2 Principaux inconvénients . . . . . 103
4.3 Domaine d'applications des réseaux systoliques . . . . . 103
4.4 Convolution non récursive . . . . . 104
4.4.1 Calcul de la convolution non récursive . . . . . 104
4.4.2 Description de la cellule élémentaire . . . . . 105
4.4.3 Description du réseau systolique . . . . . 107
4.5 Convolution récursive . . . . . 110
4.5.1 Description du réseau systolique . . . . . 111

4.6 Convolution récursive générale . . . . . 113
4.7 Optimisation des structures systoliques . . . . . 115
4.7.1 Réseau systolique unidirectionnel pour la convolution . . . . . 115
4.7.2 Technique « Diviser pour Régner » . . . . . 117
4.8 Résistance aux pannes . . . . . 122
4.8.1 Réseaux linéaires unidirectionnels . . . . . 122
4.8.2 Réseaux linéaires bidirectionnels . . . . . 123
4.9 Réseaux orthogonaux pour le calcul de la convolution . . . . . 123
4.10 Conclusion . . . . . 126

**5 Codeur convolutif et turbo-codeur haut débit** **127**
5.1 Introduction . . . . . 127
5.2 Principe . . . . . 130
5.3 Description fonctionnelle du codeur convolutif série . . . . . 134
5.3.1 Architecture *Many To One* (MTO) du codeur convolutif . . . . . 135
5.3.2 Architecture *One To Many* (OTM) du codeur convolutif . . . . . 138
5.4 Diviseur parallèle . . . . . 140
5.4.1 Description fonctionnelle du diviseur parallèle . . . . . 140
5.4.2 Implantation du diviseur parallèle . . . . . 143
5.5 Etude du codeur convolutif haut débit . . . . . 147
5.5.1 Présentation du codeur convolutif haut débit . . . . . 147
5.5.2 Description fonctionnelle de l'architecture parallèle-pipeline . . . . . 148
5.6 Implantation du codeur convolutif parallèle . . . . . 150
5.6.1 Implantation du codeur parallèle type MTO ($CP_{mto}$) . . . . . 150
5.6.2 Implantation du codeur parallèle type OTM ($CP_{otm}$) . . . . . 152
5.7 Résultats expérimentaux . . . . . 153
5.7.1 Synthèse de l'architecture parallèle . . . . . 153
5.7.2 Performances du codeur parallèle $CP_{mto}$ . . . . . 157
5.7.3 Comparaison des performances des codeurs OTM et MTO . . . . . 160
5.8 Simulation des codeurs parallèles $CP_{mto}$ et $CP_{otm}$ . . . . . 166
5.8.1 Simulation du codeur *Many To One* . . . . . 168
5.8.2 Simulation du codeur *One To Many* . . . . . 169
5.9 Application aux turbo-codes . . . . . 170
5.10 Conclusion . . . . . 173

**6 Décodeur de syndrome haut débit** **175**
6.1 Introduction . . . . . 175
6.2 Principe du décodage à logique majoritaire . . . . . 178

6.2.1 Somme des contrôles de parité . . . . . 178
6.2.2 Règle à logique majoritaire pour le décodage . . . . . 178
6.3 Architecture série du décodeur de syndrome . . . . . 178
6.4 Architecture parallèle proposée . . . . . 180
6.4.1 Codeur parallèle . . . . . 180
6.4.2 Réseau de correction . . . . . 181
6.5 Architecture parallèle du bloc de correction . . . . . 183
6.5.1 Cas $\wp = m$ . . . . . 183
6.5.2 Cas $\wp < m$ . . . . . 184
6.5.3 Cas $\wp > m$ . . . . . 185
6.6 Réseau de correction dans le cas de rendement $1/n$ . . . . . 188
6.7 Matrice d'interconnexion . . . . . 189
6.8 Résultats expérimentaux . . . . . 190
6.8.1 Analyse du réseau de correction . . . . . 192
6.8.2 Implantation du décodeur . . . . . 194
6.9 Conclusion . . . . . 194

**CONCLUSION GÉNÉRALE** **199**

**Bibliographie** **202**

**A Listes des Abréviations** **213**

**B Notations** **217**

**C Résultats Expérimentaux Diviseurs** **221**

**D Résultats Expérimentaux Codeurs convolutifs** **223**

**E Résultats Expérimentaux Turbo-codeurs convolutifs** **227**

**F Résultats Expérimentaux Décodeurs de syndromes** **229**

**G Listes des publications** **231**

# INTRODUCTION GÉNÉRALE

# Introduction générale

## Contexte et objectifs de la thèse

Avec l'apparition de nouvelles technologies en télécommunications, les normes actuelles deviennent de plus en plus strictes en termes de qualité du service (QoS) rendu aux clients. De plus, l'integration des services et la diversité des données échangées (voix, vidéo, cellules ATM, etc.) exigent des systèmes de plus en plus rapides pour traiter des volumes d'information en augmentation constante, tout en conservant des délais d'attente les moins contraignants possibles pour l'utilisateur. A ces deux contraintes liées au traitement de l'information s'ajoute la nécessité de protéger les informations émises dans des environnements où se côtoient une multitude de systèmes susceptibles de se parasiter mutuellement, sans oublier les diverses autres sources de pollution ( orage, bruit galactique, etc,). En bref, de la qualité du service rendu en termes de communication, quelles que soient les conditions d'émission et de réception, se dégagent deux mots clés : fiabilité et rapidité. D'autre part, l'aspect financier imposé à la réalisation d'un projet peut être un facteur déterminant dans le choix d'une technologie donnée, le but étant de satisfaire le cahier des charges avec un coût minimum afin d'être compétitif sur le marché en assurant la meilleure rentabilité et en proposant des solutions originales et efficaces.

Le contexte de cette thèse se situe au niveau des applications modem haut débit. En effet, les modems actuels requièrent des débits de transmission très élevés, des techniques de modulation variées et des protocoles de communications complexes. Ces applications sont caractérisées par l'emploi, entre autres, de fonctions complexes et coûteuses en temps de traitement. Certaines de ces fonctions, telles que la modulation et démodulation numériques, le filtrage, concernent l'adaptation du signal à la nature du support de transmission. D'autre fonctions, telles que le codage canal, ont pour but de renforcer la robustesse des informations transmises.

Cette thèse s'inscrit dans deux axes de recherche menés au sein du laboratoire LICM. Le premier concerne la conception architecturale d'une chaîne de transmission à haut débit et faible coût, dans le cadre d'un contrat CIFRE entre le laboratoire LICM et la société 2MG Communication : L'objectif principal de ce travail donc est de concevoir une architecture rapide et configurable pour un système de codage de faible coût et à haut débit permettant une protection optimale des données. Ce système pourra ensuite être intégré sous forme de module ADM (*Auxiliary Dedicated Module*) dans un

processeur modulaire et hautement parallèle construit autour d'un cœur de DSP [1].

L'objectif imposé par le cahier des charges est de proposer un système de codage qui satisfasse les quatres points suivants :

- protection optimale des données ;
- traitement rapide (haut débit) ;
- architecture configurable ;
- utilisation de technologies de faible coût.

Le deuxième vise à développer une méthodologie de conception architecturale pouvant être ultérieurement implantée sous forme d'outils informatiques d'aide à la conception et de génération automatique ou semi-automatique de modèles IP synthétisables.

Dans le contexte des télécommunications les codes convolutifs restent très utilisés, soit comme entité indépendante, soit dans une structure concaténée avec un autre code, soit plus récemment sous forme d'élément constituant dans un turbo-code. De plus différentes études [2, 3, 4, 5, 6, 7] ont démontré que les performances de codage atteintes par les structures turbo-codes surpassent celles des codes classiques de protection utilisés dans le codage canal. C'est pourquoi la technique de codage adoptée a porté sur les codes convolutifs et leurs applications aux turbo-codes. Quant à l'augmentation de la vitesse de traitement, deux solutions sont envisageables : une solution directe qui consiste à choisir une technologie cible en fonction des besoins sans prendre en compte le facteur coût, solution qui ne rentre pas dans le cadre des objectifs globaux. Une deuxième solution consiste à trouver de nouvelles architectures permettant d'atteindre de hauts débits sur des cibles technologiques à faible coût (FPGA, ASIC-CMOS), ce qui est en adéquation avec les objectifs fixés. La solution proposée consiste à développer un modèle RTL (*Register Transfert Level*) d'une architecture rapide de turbo-codeur convolutif indépendamment d'une technologie cible et d'en évaluer les performances en terme de débit de traitement et de surface consommée après synthèse sur FPGA.

## Structure de la thèse

Un turbo-code convolutif est une concaténation de codes de type récursif systématique convolutif (RSC) séparés par des générateurs de permutations. Cette structure concaténée du turbo-code permet de traiter individuellement chaque code constituant. En effet, l'augmentation du débit du turbo-codeur revient à augmenter le débit de ces codes RSC constituants. Il s'agit donc d'explorer dans un premier temps les possibilités d'augmentation du débit d'un codeur RSC en utilisant des techniques de parallélisme temporel ou spatial et de proposer une architecture la plus efficace possible. La deuxième étape est l'élaboration d'un modèle RTL et l'évaluation de ses performances (débit, surface). La troisième étape consiste enfin à évaluer l'architecture globale du turbo-codeur.

Ce mémoire de thèse est structuré en 2 parties.

◇ **La première partie** s'intitule « Etat de l'art et étude théorique ». Elle est divisée en 3 chapitres.

▷ **Chapitre 1**

• La première partie de ce chapitre introduit succinctement des notions de la théorie de l'information et décrit l'aspect théorique du codage canal. On n'y voit qu'un codage adéquat de l'information permet d'effectuer une transmission fiable à travers un canal soumis à des bruits perturbateurs.

• La seconde partie s'intéresse à la chaîne de transmission et à la fonction de chaque élément la constituant.

• Enfin la dernière partie détaille les différentes techniques utilisées pour réaliser la fonction de codage canal et présente un comparatif de différents systèmes de codage afin de justifier le choix d'un turbo-code comme solution efficace.

▷ **Chapitre 2**

• La première partie s'intéresse aux codes convolutifs (récursifs et non récursifs), plus particulièrement aux architectures série *Many To One* (MTO) et *One To Many* (OTM) des codeurs convolutifs ainsi qu'aux difficultés liées à leurs implantations et aux différentes solutions permettant d'augmenter leurs performances (réduction du chemin critique, augmentation du rendement,$\cdots$). Les solutions architecturales parallèles purement combinatoires de type ICC (*Iterative Combinatorial Cells*) sont totalement rejetées car la dégradation de la fréquence de fonctionnement est proportionnelle au degré du parallélisme $\wp$, le débit ne pouvant augmenter de manière effective au delà d'une valeur limite $\wp_{limite}$. En conséquence, cette approche devient sans intérêt en aval de cette valeur.

• La deuxième partie présente les turbo-codes de manière générale puis détaille les turbo-codes parallèles. Cette partie aborde aussi les critères pour le choix des codes constituants et des matrices d'entrelacement qui sont des facteurs déterminants concernant les performances d'un turbo-code. Elle aborde aussi les conditions de bon fonctionnement du processus du codage (problème de la terminaison du treillis).

▷ **Chapitre 3**

• Ce chapitre concerne l'étude des différents algorithmes de décodage itératif des turbo-codes. Il présente le schéma général du décodage SISO (*Soft-Input Soft-Output*) et détaille les algorithmes Viterbi et MAP de décodage sur treillis des codes convolutifs, ainsi que leurs versions adaptées au décodage SISO : SOVA, Log MAP et Max-Log MAP. Cette étude met en évidence les difficultés liées à la parallélisation directe de ce type d'algorithme. Ce chapitre est clos par l'énumération de turbo-codeurs implantés en technologies ASIC ou FPGA et de cœur IP (*Intellectual Property*) proposés sur le marché.

◇ **La deuxième partie** s'intitule « Etude architecturale et implantation ». Elle est divisée en 3 chapitres.

▷ **Chapitre 4**

• Ce chapitre concerne les architectures systoliques dédiées à la convolution. Trois types de convolutions ont été abordés, à savoir, les convolutions non-récursive, récursive et récursive générale. Chaque modèle systolique a été décrit puis synthétisé sur FPGA et simulé. Cette étude a permis de mettre en évidence la simplicité de leur mise en œuvre due à leur structure régulière et leur complexité combinatoire réduite à celle d'une cellule élémentaire. Cependant, cette solution permet d'atteindre dans le meilleur des cas, un débit limité à la fréquence maximale de fonctionnement du composant cible.

▷ **Chapitre 5**

• Ce chapitre détaille l'architecture parallèle proposée appliquée aux codes convolutifs. Il présente le principe général de fonctionnement du codeur rapide et la solution architecturale adoptée. L'architecture proposée consiste à combiner à l'approche combinatoire une technique de pipeline pour maximiser la valeur de $\wp_{limite}$ en minimisant la longueur du chemin critique dû au circuit combinatoire d'une part, et en la rendant indépendante du degré de parallélisme d'autre part. De cette manière, toute augmentation du degré de parallélisme $\wp$ entraînera automatiquement l'augmentation du débit. Le fonctionnement du codeur rapide est basé sur un diviseur parallèle qui a été implanté en deux versions différentes, une version purement combinatoire qui présente l'avantage de consommer moins de surface et une version pipeline qui réduit le chemin critique au dépend d'une augmentation du temps de latence. Quatre modèles de codeurs convolutifs parallèles décrits en VHDL ont été synthétisés sur FPGA. Le codeur RSC parallèle est ensuite utilisé comme code constituant d'un turbo-codeur à deux niveaux.

▷ **Chapitre 6**

• Il porte sur une architecture rapide du décodeur à logique majoritaire qui pourrait être associée au codeur convolutif rapide. Le choix s'est porté sur ce type de décodeur car son principal avantage réside dans la simplicité de sa mise en œuvre et sa rapidité de décodage, ce qui le rend mieux adapté aux applications hauts débits comparé aux décodeurs de type Viterbi ou séquentiel. De plus, comme un décodeur de syndrome est constitué d'une réplique exacte du codeur et d'un registre de syndrome, la parallélisation du décodeur consiste à paralléliser les deux parties. Le bloc codeur étant traité dans le chapitre précédent, cette partie traite la parallélisation du registre de syndrome. Elle présente l'étude de l'architecture série d'un décodeur de syndrome et la structure parallèle générale du décodeur. L'étude est focalisée sur la partie registre de syndrome et ses différentes structures parallèles en fonction de la taille de la mémoire du codeur $m$ et du degré de parallélisme $\wp$.

# I. ETAT DE L'ART ET ÉTUDE THÉORIQUE

# Chapitre 1

# Introduction au codage canal

Avant d'aborder l'aspect pratique de la conception des codeurs, nous nous intéresserons à la théorie de l'information et aux limitations qu'elle impose au codage canal. Ce chapitre n'a pas pour but de re-démontrer la théorie mais juste de donner des notions fondamentales de manière succincte afin de montrer qu'un codage adéquat de l'information permet d'effectuer une transmission fiable (sans erreurs) à travers un support de transmission soumis à des bruits perturbateurs. Pour plus de détails et d'informations on pourra se référer à [8, 9, 10]. La théorie fixe la limite (par rapport à la capacité du canal de transmission) pour effectuer une transmission correcte mais elle n'explicite pas le procédé permettant de l'atteindre. Le deuxième volet de ce chapitre concerne donc les stratégies de codage canal utilisées pour transmettre l'information de la manière la plus fiable, les méthodes les plus performantes étant celles qui se rapprochent le plus de la limite fixée par Shannon.

Ce chapitre se divise en trois parties. La première partie donne des notions de base sur la théorie de l'information et introduit le théorème fondamental du codage canal. La deuxième partie s'intéresse à la chaîne de transmission et à la fonction de chaque élément la constituant. La dernière partie enfin, détaille les techniques utilisées pour réaliser la fonction du codage canal.

**Mots clés :** quantité d'information, chaîne de transmission, codage canal.

## 1.1 Introduction

La « théorie de l'information » est une théorie qui s'intéresse à la transmission des messages d'une source vers un destinataire à travers un canal. Elle a été introduite en 1948 par l'ingénieur américain Claude Elwood Shannon [8, 9] et affirme qu'en codant l'information correctement, celle-ci peut être transmise à travers un canal parasité sans perte de son contenu à la réception. Elle se base sur une mesure quantitative de l'information, c'est à dire qu'elle ne s'intéresse pas au sens du message du point de vue sémantique, mais seulement à la quantité d'information que le message véhicule. Dans [11] on l'assimile à un messager dont le but est de porter une lettre sans avoir connaissance

de son contenu, les seules informations dont il dispose sont le poids et les dimensions extérieures de cette lettre, Le message qu'elle véhicule n'ayant aucune influence sur les moyens de la transporter. La distinction entre message utile et parasite est totalement liée à la finalité du récepteur. Par exemple si deux personnes $A$ et $B$ écoutent chacune leur propre musique de deux sources $S_a$ et $S_b$ proches, $S_b$ s'avérera être une source de perturbation pour la personne $A$ qui essaye d'écouter l'information qui l'intéresse de la source $S_a$ et réciproquement. Afin de dissocier les deux types d'événements, l'information utile est localisée dans le bloc source tandis que les éléments parasites sont dans le bloc canal.

## 1.2 Mesure quantitative de l'information

Un destinataire n'est pas censé connaître au préalable le contenu d'un message qu'il reçoit, sinon l'émission de ce message n'apporte rien au récepteur comme information. De cette affirmation découlent deux points :

- la source d'information fournit de manière aléatoire un ensemble de messages élémentaires qui forment le message émis. Donc, comme on est en présence d'événements aléatoires, chaque message élémentaire est pondéré par sa probabilité d'occurrence (émission) ;
- la quantité d'information est une mesure de son imprévisibilité. Plus le message reçu est prévisible moins est grande la quantité d'information qu'il apporte.

Si $x$ est un message élémentaire avec une probabilité d'émission de $p(x)$ , la quantité d'information qu'il apporte $h(x)$ est une fonction croissante de son improbabilité $\frac{1}{p(x)}$.

$$h(x) = f\left(\frac{1}{p(x)}\right) \tag{1.1}$$

Si $x$ est un message élémentaire avec une probabilité d'émission certaine $p(x) = 1$, alors la quantité d'information qu'il apporte est nulle.

$$h(x) = f(1) = 0 \tag{1.2}$$

De plus si $x$ et $y$ sont deux événements indépendants, la quantité d'information qu'ils apportent est la somme des quantités d'information de chacun.

$$h(x,y) = f(x) + f(y) \tag{1.3}$$

La fonction qui correspond à «$h$» est la fonction logarithmique [8]. Donc, la quantité d'information intrinsèque s'écrit suivant l'équation (1.4).

$$h(x) = -\log_b(p(x)) \tag{1.4}$$

où $b$ représente la base logarithmique. Il définit l'unité d'information. Dans la pratique, les contraintes technologiques des systèmes de télécommunications imposent l'utilisation de symboles binaires à valeurs dans $\{0,1\}$. Dans ce cas, $b = 2$ et l'unité est le Shannon (Sh).

Si $x$ est un message élémentaire avec une probabilité à priori d'émission $p(x)$ et $y$ est un message élémentaire reçu, et si $p(x|y)$ est la probabilité que $x$ ait été émis sachant que $y$ a été reçu, la quantité d'information mutuelle $h(x;y)$ est donnée par :

$$h(x;y) = -\log_2\left(\frac{p(x|y)}{p(x)}\right) \tag{1.5}$$

## 1.3 Définition de l'entropie et de l'information mutuelle

Soit une source aléatoire, discrète et finie, émettant plusieurs messages élémentaires $x_1, x_2, \cdots, x_k$ avec des probabilités d'émission respectives $p_1, p_2, \cdots, p_k$. L'émission des messages par cette source est formalisée par la variable aléatoire discrète $X$ avec $p_i = P\{X = x_i\}$ $(i \in \{1, \cdots, k\})$.

De la même manière, l'ensemble des messages élémentaires reçus $y_1, y_2, \cdots, y_n$ pondérés par les probabilités respectives $q_1, q_2, \cdots, q_n$ est formalisé par la variable aléatoire $Y$ avec $q_i = P\{Y = y_j\}$ $(j \in \{1, \cdots, n\})$.

### 1.3.1 Définition de l'entropie

L'entropie de la source est la quantité d'information moyenne de cette dernière. Elle est définie comme étant l'espérance mathématique de la quantité d'information intrinsèque $h(x)$ de chaque message élémentaire $x$ de la source.

$$\mathbf{H}(X) = E[h(x_i)] = -\sum_{i=1}^{k} p_i \log_2(p_i) \tag{1.6}$$

Dans le cas où tous les messages élémentaires sont équiprobables $p_1 = p_2 = \cdots = p_k$, l'entropie $\mathbf{H}(X)$ est maximale et vaut $\log_2(k)$. En considérant la variable aléatoire $(X, Y)$ de probabilité $p_{ij} =$

$P(X = x_i, Y = y_j)$, l'entropie conjointe de $(X,Y)$ est :

$$\begin{aligned} \mathbf{H}(X,Y) = E\left[h(x_i,y_j)\right] &= -\sum_{i=1}^{k}\sum_{j=1}^{n} p_{ij}\log_2\left(p_{ij}\right) && (1.7)\\ &= \mathbf{H}(X) + \mathbf{H}(Y|X) && (1.8)\\ &= \mathbf{H}(Y) + \mathbf{H}(X|Y) && (1.9) \end{aligned}$$

L'entropie conjointe est toujours inférieure à la somme des entropies intrinsèques de $X$ et $Y$ sauf si celles-ci sont indépendantes.

L'entropie conditionnelle de la variable aléatoire $(Y|X = x_i)$ de probabilité $p_{j|i} = P(Y = y_j|X = x_i)$ est :

$$\mathbf{H}(Y|X = x_i) = -\sum_{j=1}^{n} p_{j|i}\log_2\left(p_{j|i}\right) \tag{1.10}$$

**Entropie d'une variable binaire**

Dans le cas d'une variable binaire $X$ ayant les probabilités d'émissions suivantes :

$$\begin{cases} X = 1 \quad \text{avec} \quad p_1 = p, \\ X = 0 \quad \text{avec} \quad p_0 = 1 - p. \end{cases} \tag{1.11}$$

L'entropie (**H**(X) en Shannon) de cette variable en fonction de $p$ est représentée sur la figure 1.1 et est égale à :

$$\begin{cases} \text{si } 0 < p < 1 \;\Rightarrow\; \mathbf{H}(X) = -p\log_2(p) - (1-p)\log_2(1-p) \\ \text{si } p = 0 \text{ ou } 1 \;\Rightarrow\; \mathbf{H}(X) = 0 \end{cases} \tag{1.12}$$

### 1.3.2 Définition de l'information mutuelle

L'information mutuelle mesure la quantité d'information que la connaissance d'une variable (message élémentaire reçu $y$) permet d'apporter sur l'autre (message élémentaire émis $x$). L'information mutuelle entre l'entrée et la sortie d'un canal est :

$$\begin{aligned} I(X;Y) &= \mathbf{H}(X) - \mathbf{H}(X|Y) && (1.13)\\ &= \mathbf{H}(Y) - \mathbf{H}(Y|X) && (1.14) \end{aligned}$$

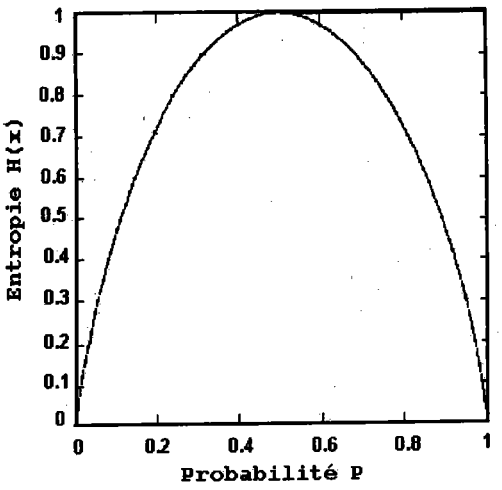



FIG. 1.1 – *Entropie de la variable binaire.*

L'équation (1.14) représente la différence entre l'incertitude à la sortie du canal quand on a aucune connaissance du message élémentaire émis moins celle que l'on a quand on connaît le message élémentaire émis.

$$I(X;Y) = 0 \quad \Rightarrow \quad \text{canal mauvais.} \tag{1.15}$$

$$I(X;Y) = \mathbf{H}(X) \quad \Rightarrow \quad \text{canal sans erreurs.} \tag{1.16}$$

### 1.3.3 Débit d'information

Le débit d'information est défini comme étant le produit de l'entropie de la source par le nombre moyen de symboles par seconde. Si la durée moyenne de chaque symbole est $\tau_m$, le débit de l'information de la source est [10] :

$$D_S = \frac{\mathbf{H}(X)}{\tau_m} \; Sh/s \tag{1.17}$$

## 1.4 Canal de transmission

Un canal de transmission discret sans mémoire est constitué de trois blocs :

- une entrée (émission) modélisée par la variable aléatoire $X$ qui peut prendre n'importe quelle valeur parmi un ensemble de messages élémentaires (symboles) $S_E = \{x_1, x_2, \cdots, x_k\}$ ;
- une sortie (réception) modélisée par la variable aléatoire $Y$ qui peut prendre n'importe quelle valeur parmi un ensemble de symboles $S_R = \{y_1, y_2, \cdots, y_n\}$ ;
- une matrice de transition $T_R$ dont l'élément $t_{ij} = p_{j|i}$ est la probabilité que pour un symbole $x_i$ émis, le symbole reçu soit $y_j$.

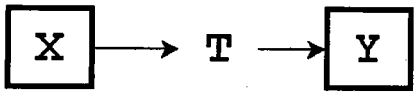



FIG. 1.2 – *Modélisation du canal discret sans mémoire.*

La capacité $C_C$ du canal de transmission est définie comme le maximum de l'information mutuelle moyenne $I(X;Y)$.

$$C_C = \max_{(p_i)} (I(X;Y)) \tag{1.18}$$

### 1.4.1 Canal BSC

Un canal est dit binaire et symétrique (BSC : *Binary Symetric Channel*) si ses symboles d'émissions et de réceptions sont égaux et binaires respectivement ($S_E = S_R = \{0,1\}$) et si sa matrice de transition $T_R$ est symétrique (cf. paragraphe 1.4.3) :

$$T_R = \begin{pmatrix} 1-p & p \\ p & 1-p \end{pmatrix} \tag{1.19}$$

Dans le cas d'un canal BSC, la capacité est représentée sur la figure 1.3 et est définie par :

$$C_C = 1 - \mathbf{H}(X) \tag{1.20}$$

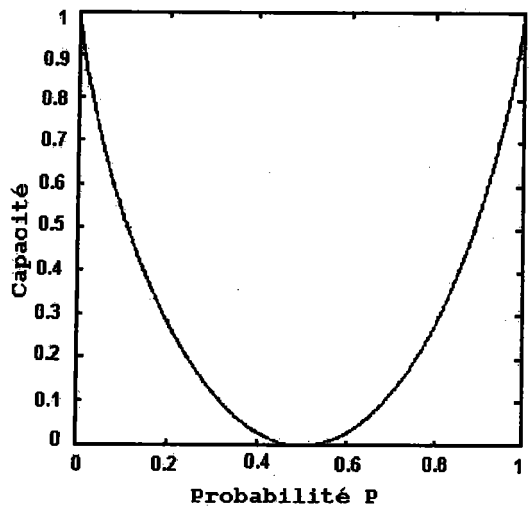



FIG. 1.3 – *Capacité d'un canal BSC.*

### 1.4.2 Canal AWGN

L'entropie d'une source gaussienne de moyenne nulle et de variance $\sigma^2$ est donnée par l'équation (1.21) [8].

$$\mathbf{H}(X) = \frac{1}{2}\log_2(2\pi e\sigma^2) \tag{1.21}$$

Dans beaucoup de cas, on considère qu'un bruit blanc gaussien (AWGN : *Additif White Gaussian Noise*) d'énergie $N_0$, de moyenne nulle et de variance $N_0/2$ (équ. (1.22)) s'additionne au signal utile [8].

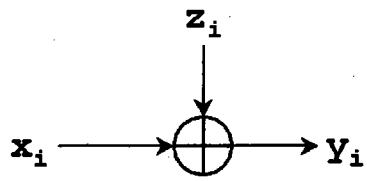



FIG. 1.4 – *Canal AWGN.*

$$\mathbf{H}(Z) = \frac{1}{2}\log_2(2\pi e\frac{N_0}{2}) \tag{1.22}$$

L'information mutuelle du canal est donnée par l'équation (1.14), sachant que $Y = X + Z$ et que $X$ et $Z$ sont indépendantes [8] on obtient :

$$I(X;Y) = \mathbf{H}(Y) - \mathbf{H}\left((X+Z)|X\right) \tag{1.23}$$

$$I(X;Y) = \mathbf{H}(Y) - \mathbf{H}(Z) \tag{1.24}$$

Dans le cas d'un canal binaire, on utilise une modulation de phase à deux états (BPSK : *Binary Phase Shift Keying*) où chaque bit codé est représenté par son signal correspondant ($s_1(t)$ pour 1 et $s_0(t)$ pour 0) pendant une durée de $\tau_m$ secondes [12].

$$s_0(t) = \sqrt{\frac{2E_c}{\tau_m}}\sin\left(2\pi f_0 t + \frac{\pi}{2}\right),\ (0 \le t \le \tau_m) \tag{1.25}$$

$$s_1(t) = \sqrt{\frac{2E_c}{\tau_m}}\sin\left(2\pi f_0 t - \frac{\pi}{2}\right),\ (0 \le t \le \tau_m) \tag{1.26}$$

L'énergie moyenne du bit modulé après codage est $E_c = R \times E_b$ [3], $E_b$ étant l'énergie du signal modulé avant codage, $R$ le rendement du code (cf. paragraphe 2.2) utilisé et $f_0$ est la fréquence de la porteuse multiple de $\frac{1}{\tau_m}$.

Sachant que l'entropie d'une distribution non gaussienne a comme borne supérieure l'entropie de

la distribution gaussienne de même variance [5, 8], on a :

$$I(X;Y) \leq \frac{1}{2}\log_2(2\pi e(E_c + \frac{N_0}{2})) - \frac{1}{2}\log_2(\pi e N_0) \quad (1.27)$$

$$I(X;Y) \leq \frac{1}{2}\log_2(1 + \frac{2E_c}{N_0}) \quad (1.28)$$

En utilisant l'équation (1.18) on aboutit à l'expression de la capacité du canal AWGN :

$$C_C = \frac{1}{2}\log_2(1 + \frac{2E_c}{N_0}) \quad (1.29)$$

La figure 1.5 représente la capacité d'un canal AWGN dans le cas d'entrée continue et binaire avec modulation BPSK en fonction du rapport $\frac{E_b}{N_0}$ (dB).

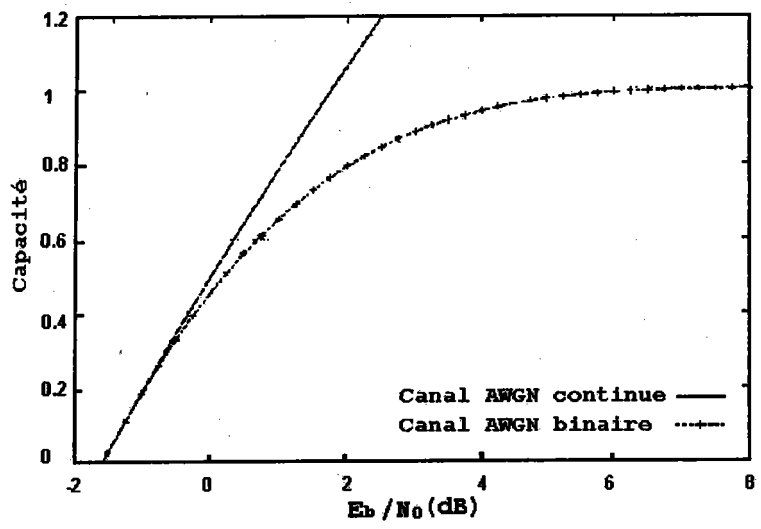



FIG. 1.5 – *Capacité du canal AWGN (continue & binaire).*

Le but dans tout système de communication est de se rapprocher le plus possible de la capacité du canal de transmission. Cependant la présence inévitable du bruit dégrade considérablement les symboles émis. De plus comme le bruit est un phénomène imprévisible on ne peut pas le supprimer de manière directe. D'où la nécessité d'envisager des mécanismes permettant de compenser les effets perturbateurs du bruit et de se rapprocher ainsi de cette limite. En pratique, les canaux de transmission sont limités en puissance et en fréquence. La limitation moyenne de puissance $P_S$ fixe le nombre d'échantillons $n$ en limitant leurs amplitudes à un ensemble fini $x_1, x_2, \cdots, x_n$ [8], elle est donnée par :

$$\frac{1}{n}\sum_{i=1} n(x_i)^2 \leq P_S \quad (1.30)$$

La limitation en fréquence détermine un intervalle de fréquence ou bande passante $B_P$ sur lequel

le signal ne subit pas d'affaiblissement supérieur à une valeur de 3 dB, ce qui correspond à un affaiblissement du signal d'entrée de 50%. L'affaiblissement représente la perte du signal en énergie dissipée dans le canal. Il est caractérisé par un signal en sortie plus faible que le signal en entrée et est calculé de la manière suivante :

$$A_f(dB) = 10 \log \frac{A_s}{A_e} \tag{1.31}$$

où, $A_f$ est l'affaiblissement en $dB$, $A_e$ et $A_s$ sont les amplitudes respectives des signaux d'entrée et de sortie.

### 1.4.3 Modélisation des erreurs

Les modèles des canaux précédents sont des fonctions de la puissance du bruit introduit par le canal de transmission. Ce bruit se traduit par l'introduction d'erreurs dans l'information. La distribution des erreurs peut être représentée par des modèles théoriques tel que :

- le modèle de Poisson [8] qui considère que le nombre d'erreurs par unité de temps est une variable aléatoire qui suit une loi de Poisson ;
- le modèle de Neymann-Pearson [8] utilisé pour modéliser les erreurs qui apparaissent par paquet, qui se base sur deux lois de Poisson, une pour décrire l'apparition d'un paquet et une pour définir le nombre d'erreurs par paquet ;
- le modèle de Pareto [8] qui suppose que les intervalles entre deux erreurs sont indépendants et suivent une loi de Pareto ;
- le modèle de Markov [8], etc.

Dans le cas de canaux sans mémoire, le canal $BSC$ définit par sa matrice de transition $T_R$ (cf. paragraphe 1.4.1) est représenté par son diagramme de transition sur la figure 1.6.

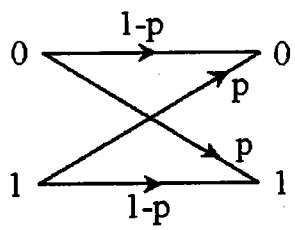



FIG. 1.6 – *Diagramme de transition du canal BSC.*

Chaque bit reçu a une probabilité $p$ d'être faux et $1 - p$ d'être correct, indépendamment des autres bits transmis. Par conséquent, les erreurs dans les canaux sans mémoire sont aléatoires. Ce type de canal est appelé « canal avec erreurs aléatoires ». On le retrouve par exemple dans les communications spatiales (sonde d'exploration, satellite de communication). Ce type de canal peut être décrit par le modèle de Poisson.

Dans le cas d'un canal avec mémoire, la sortie $y$ du canal à l'instant $t$ dépend des perturbations à l'instant $t$ et de celles antérieures à $t$. Un modèle simplifié d'un canal avec mémoire est représenté sur la figure 1.7.

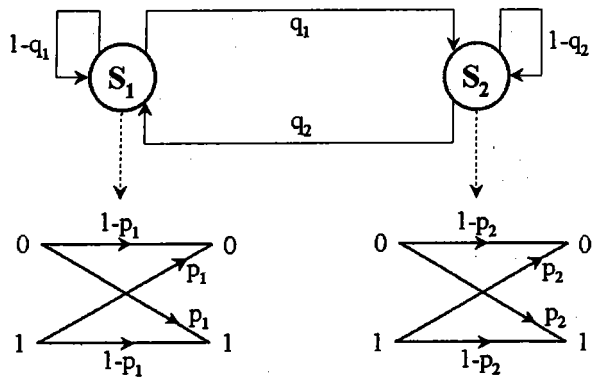



FIG. 1.7 – *Diagramme de transition d'un canal avec mémoire.*

Dans l'exemple le canal a deux états possibles, $S_1$ représentant une transmission sans erreurs et $S_2$ une transmission avec erreurs. La probabilité de transition de $S_1$ vers $S_2$ est $q_1$ et de $S_2$ vers $S_1$ est $q_2$ avec $q_1 << q_2$. Quant le canal est dans l'état $S_1$ la probabilité d'erreur est faible $p_1 \approx 0$ tandis que dans $S_2$ elle est grande $p_2 \approx 0.5$. Ce qui se traduit par l'apparition de paquets d'erreurs dans l'état $S_2$. Les canaux avec mémoire [12] sont appelés « canaux avec paquets d'erreurs », on les retrouvent par exemple dans les transmissions radio, câbles, enregistrements magnétiques, etc. Ce type de canal peut être décrit par le modèle de Neymann-Pearson.

## 1.5 Théorème fondamental du codage canal

Dans le cas de canaux soumis à des perturbations, il est possible en utilisant un codage approprié, de transmettre l'information à un débit proche de la capacité du canal de transmission, avec une petite probabilité d'erreur [13]. En d'autre terme, pour une source d'information de débit $D_S$ bit/s qui transmet sur un canal de capacité $C_C$ (bit/s), il existe, si $D_S < C_C$, un code ayant des mots de codes de longueur $n$ tels que la probabilité d'erreur de décodage $P_e$ [1] soit :

$$P_e \leq 2^{-n \cdot E^g(D_S)} \tag{1.32}$$

où $n$ est la longueur du mot de code et $E^g(x)$ est une fonction dite exposant de l'erreur (de Gallager ou de codage aléatoire). Elle est positive pour tout $D_S \leq C_C$ et complètement définie par la caractéristique du canal de transmission.

1. En considérant un décodage par maximum de vraisemblance.

Les deux affirmations précédentes mettent en évidence la possibilité de transmettre des messages avec une probabilité d'erreur aussi faible que voulue, quel que soit le niveau des perturbations du canal de transmission, la condition nécessaire et suffisante étant que le débit soit inférieur à la capacité du canal [10]. Néanmoins, le théorème fondamental n'explicite pas le procédé de codage permettant d'atteindre cette limite. La seule information est l'association d'une redondance (symbole de contrôle) aux symboles émis de sorte à permettre au destinataire de détecter la présence ou non d'erreur et de les corriger si le type de codage utilisé le permet.

**Remarque :** en considérant qu'un symbole codé est transmis chaque « $\tau_m$ » secondes, la vitesse de transmission des symboles est $\frac{1}{\tau_m}$ bauds. Si le rendement du système de codage est $R = \frac{k}{n}$, alors le débit de l'information est $D_S = \frac{R}{\tau_m}$. De plus le canal est généralement limité en fréquence et est défini par sa bande passante $B_P$. Dans un système non codé où $R = 1$ le débit vaut $D_S = \frac{1}{\tau_m} = 2B_P$ et est limité par la bande passante du canal. Tandis que dans un système codé de rendement $R < 1$ le débit est réduit d'un facteur de $R$ par rapport à un système non codé, $D_S = \frac{R}{\tau_m} = 2RB_P$. Par conséquent, le maintien du débit d'un système après codage nécessite l'utilisation d'une bande passante plus large [12].

## 1.6 Chaîne de transmission

Le schéma classique d'une chaîne de transmission est représenté sur la figure 1.8. En télécommunication, on utilise le modèle simplifié.

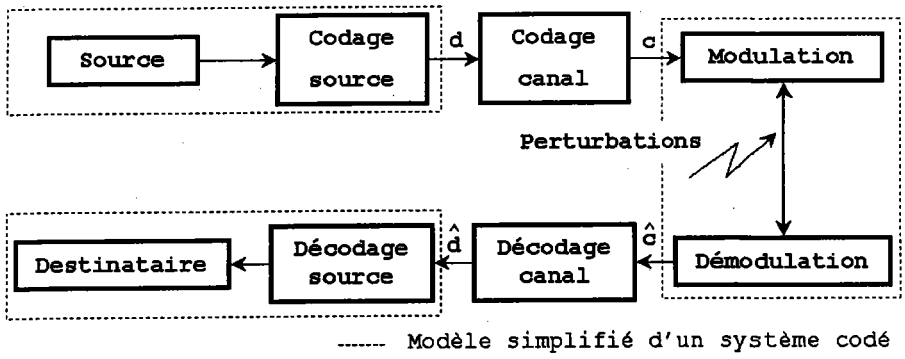



FIG. 1.8 – *Diagramme d'une chaîne de transmission classique.*

### 1.6.1 Source d'information

La source d'information est le premier maillon de la chaîne de transmission. Elle fournit le message porteur de l'information. Le message peut être de nature analogique ou numérique et la source peut être représentée par une interface homme-machine ou machine-machine.

### 1.6.2 Codage source

Le codage source consiste d'une part à convertir le message continu en une séquence numérique (utilisation d'un convertisseur analogique numérique : CAN) et à transformer le message de la source en une séquence d'information « $D(x)$ » de façon à :

- minimiser la taille du message en éliminant les redondances naturelles de l'information source (algorithme de compression) ;
- retrouver le message originel à partir de la séquence de substitution « $D(x)$ » (algorithme réversible).

Le but de cette opération est d'optimiser les ressources nécessaires à la transmission (temps, puissance, bande passante, etc.). Il est à noter que les limites théoriques du codage source sont fixées par le premier théorème de Shannon [8, 13].

### 1.6.3 Codage canal

Le codage canal a pour rôle de protéger l'information émise contre les perturbations du canal de transmission susceptible de modifier son contenu. Il s'agit donc de rajouter de la redondance de manière à détecter et éventuellement corriger les erreurs lors de la réception si la stratégie adoptée le permet. Les conditions d'un codage correct sont déterminées par le second théorème de Shannon (cf. paragraphe 1.5). L'information $D(x)$ issue du codage source est transformée en séquence codée $C(x)$.

### 1.6.4 Modulation

Elle associe à chaque symbole émis un signal de durée $\tau_m$. Le type du signal dépend des propriétés physiques du support de transmission (canal) choisi. Ainsi, l'information peut être véhiculée par une tension ou un courant si le support est filaire (cuivre), une onde lumineuse si le support est optique (fibre optique) ou une onde électromagnétique ou radioélectrique dans le cas d'un support aérien (air ou vide). La modulation permet de remédier à tous les inconvénients liés à une transmission en bande de base, citons quelques limitations :

- les signaux basse fréquence sont les plus atténués sur la ligne ;
- pas de propagation pour les signaux de fréquence en dehors de la bande passante du canal ;
- pertes et affaiblissement proportionnels à la longueur et aux types du support de transmission ;
- impossibilité de différencier plusieurs communications sur un même support ;
- régénération périodique du signal sur une longue distance.

La modulation utilise une onde porteuse qui sert à transposer les données par modification d'une ou de plusieurs caractéristiques de cette onde, amplitude (AM), phase (BPSK, QPSK, etc.)[2], fréquence (FM), amplitude et phase (QAM)[3], etc.

### 1.6.5 Canal de transmission

Il représente la liaison entre l'émetteur et le récepteur et peut être de différentes natures selon le type de grandeur qu'il permet de véhiculer. Le canal de transmission est caractérisé par sa capacité et par sa bande passante (cf. paragraphe 1.4). Comme vu au paragraphe 1.4.3, il existe plusieurs modèles théoriques du canal de transmission en fonction des types d'erreurs les plus fréquents.

Les sources de perturbations [8, 14] sont diverses et dépendent essentiellement du milieu où se trouve le canal de transmission. Les principaux types de bruits sont : les bruits galactiques entre 20 MHz et 200 MHz dus aux rayonnements des différentes sources d'énergie de l'espace ; les bruits atmosphériques jusqu'à 20 MHz induit par les éclairs orageux ; le bruit industriel ; le bruit urbain ; les micro-coupures correspondant à de courtes interruptions du signal ; les sauts de phase et scintillements liés à des variations brusques de phase ou lentes causées par les alimentations électriques ; la diaphonie lors de l'acheminement de plusieurs liaisons par un même câble ; etc.

### 1.6.6 Démodulation

La démodulation [8, 12] permet de récupérer chaque symbole émis à partir de chaque signal modulé reçu de durée $\tau_m$. Le démodulateur fournit une tension continue ou un symbole binaire dans le cas où sa sortie est quantifiée. La valeur fournie par le démodulateur peut être modélisée par une variable aléatoire gaussienne de moyenne $\mu = \sqrt{E_c}$ et d'écart type $\sigma = \sqrt{N_0/2}$, où $E_c$ est l'énergie du symbole transmis et $N_0$ l'énergie du bruit. Donc, La tension fournie par le démodulateur peut être modélisée par deux fonctions gaussiennes $f_0$ dans l'hypothèse où un 0 est émis (équ. (1.33)) et $f_1$ dans celle où un 1 est émis (équ. (1.34)).

$$f_0(x) = \frac{1}{\sqrt{\pi N_0}} \cdot e^{-\frac{(x+\sqrt{E_c})^2}{N_0}} \tag{1.33}$$

$$f_1(x) = \frac{1}{\sqrt{\pi N_0}} \cdot e^{-\frac{(x-\sqrt{E_c})^2}{N_0}} \tag{1.34}$$

Comme le montre la figure 1.9 plus la densité spectrale du bruit augmente plus il sera difficile de prendre une décision à partir d'une tension proche de 0.

Dans le cas où la réponse (tension) du démodulateur est envoyée directement sous forme de tension continue, le décodeur doit être capable de traiter des signaux analogiques (décodeur analogique).

2. (B/Q)PSK : *(Binary/Quadrature) Phase Shift Keying*
3. QAM : *Quadrature Amplitude Modulation*

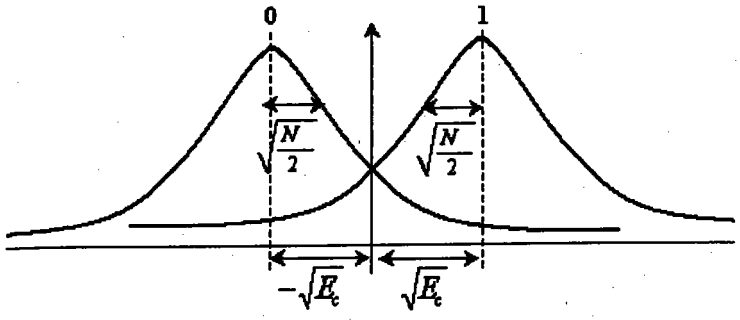



FIG. 1.9 – *Réponses du démodulateur pour la réception d'un* 1 *et d'un* 0.

L'approche la plus commune consiste à fournir au décodeur un ensemble de symboles binaires choisis en fonction de la tension du démodulateur. Dans ce cas, la réponse du démodulateur est quantifiée au prix d'une augmentation de la complexité de l'électronique. La figure 1.10 donne un exemple de quantification sur 8 niveaux.

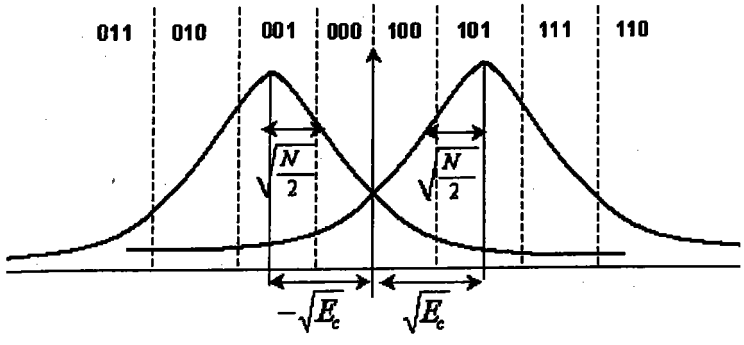



FIG. 1.10 – *Quantification de la réponse du démodulateur.*

On distingue deux types de décisions pouvant être prises par le démodulateur :

- une décision binaire (*hard decision*) où une décision franche sur le symbole reçu est prise, 1 ou 0 sans nuance ;
- une décision pondérée (*soft decision*) où le démodulateur décide du signal reçu et donne la valeur de confiance que l'on peut accorder à cette décision. La valeur de confiance se calcule à partir de l'équation (1.35). $P_1$ donnée par l'équation (1.36) est appelée la probabilité de l'événement complémentaire.

$$P_0 = \frac{f_0(x)}{f_0(x) + f_1(x)} \tag{1.35}$$

$$P_1 = 1 - P_0 \tag{1.36}$$

Enfin le démodulateur fournit au bloc décodeur une séquence binaire $\widetilde{D}(x)$ qui représente l'information émise à laquelle est superposée une séquence d'erreur $E(x)$, $\widetilde{D}(x) = D(x) + E(x)$.

### 1.6.7 Décodage canal

Comme le décrit le théorème fondamental du codage canal, pour se rapprocher de la capacité du canal de transmission , il est nécessaire de coder l'information avant de la transmettre. Au niveau du récepteur, le décodage canal consiste dans un premier temps à détecter la présence d'erreurs dans l'information et puis dans un deuxième temps de les corriger. A partir de ces deux actions découlent trois principales stratégies : les stratégies ARQ (*Automatic Repeat Request*) qui se limitent à détecter la présence d'éventuelles erreurs, la correction s'effectuant par retransmission des blocs erronés ; les stratégies FEC (*Forward Error Correction*) mettant en œuvre les codes permettant la détection et la correction des erreurs sans aucune retransmission ; enfin les systèmes hybrides combinent entre les deux techniques.

### 1.6.8 Décodage source

Le décodage source consiste à reconstituer, par l'application de l'algorithme de décodage source (décompression par exemple), l'information originelle à partir de la séquence de substitution « $D(x)$ ».

## 1.7 Stratégies de codage canal

### 1.7.1 Stratégies ARQ

Les stratégies de détection d'erreurs avec retransmission (ARQ) utilisent des codes permettant uniquement la détection d'erreurs. La retransmission en cas de détection d'erreurs nécessite l'utilisation de canaux bidirectionnels et implique un dialogue entre l'émetteur et le récepteur. Ce type de système est simple à mettre en œuvre. Par contre, le dialogue instauré entre les deux équipements de transmission ne permet pas de transmission à hauts débits surtout lorsque le nombre d'erreurs augmente. Par exemple dans le contexte des communications sans fil en mode paquets, les normes 802.15 (Bluetooth) [15] et 802.11b (Wi-Fi : *Wireless Fidelity*) intègrent dans leur mode de protection un système ARQ avec CRC. Les deux principales variantes des stratégies ARQ sont connues sous le nom de ARQ avec arrêt et attente (*Stop-and-wait ARQ*) et ARQ continue (*Continuous ARQ*) [12].

#### Systèmes « ARQ avec arrêt et attente »

Comme le montre la figure 1.11 l'émetteur attend un accusé de réception après chaque bloc transmis. Si l'accusé est positif (ACK), il transmet le bloc suivant. Dans le cas contraire, lorsqu'il reçoit un accusé négatif (NACK), il retransmet le bloc courant.

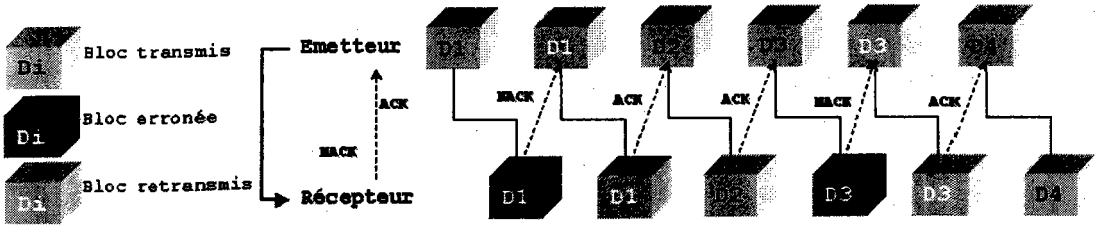



FIG. 1.11 – *Stop-and-wait ARQ.*

La retransmission se répète autant de fois que l'émetteur reçoit l'accusé de réception « NACK ». Ce type de système est adapté aux transmissions semi-duplex.

### Systèmes « ARQ continue »

Cette approche est adaptée aux transmissions de type full-duplex. Dans un système ARQ continu, l'émetteur envoie les blocs d'information de façon continue et reçoit les accusés de réception au fur et à mesure. Dans le cas de la réception d'un « NACK » la retransmission s'effectue de deux façons :

- le bloc détecté comme erroné ainsi que tous les blocs qui le suivent sont retransmis ; dans ce cas, la technique est appelée « *Go-back-N ARQ* », (fig. 1.12) ;

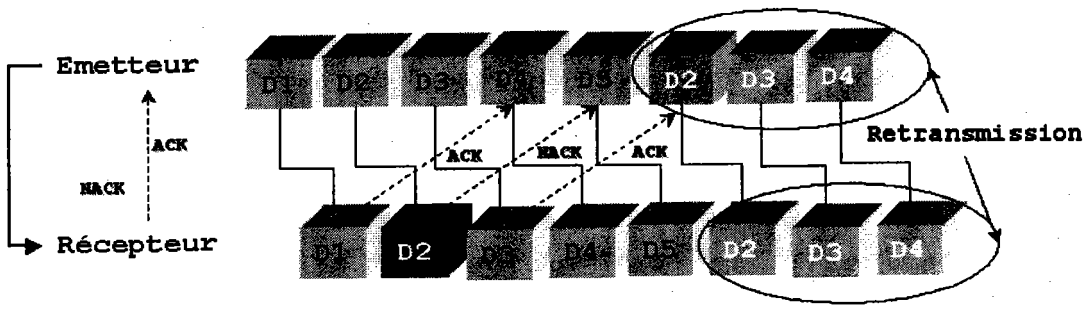



FIG. 1.12 – *Go-back-N ARQ.*

- seul le bloc erroné est retransmis ; dans ce cas la technique est appelée « *Selective-repeat ARQ* » (fig. 1.13).

L'approche « *Selective-repeat ARQ* » est plus efficace que la technique « *Go-back-N ARQ* » mais elle est plus complexe à mettre en œuvre car le récepteur doit être capable de réordonnancer les blocs de manière cohérente.

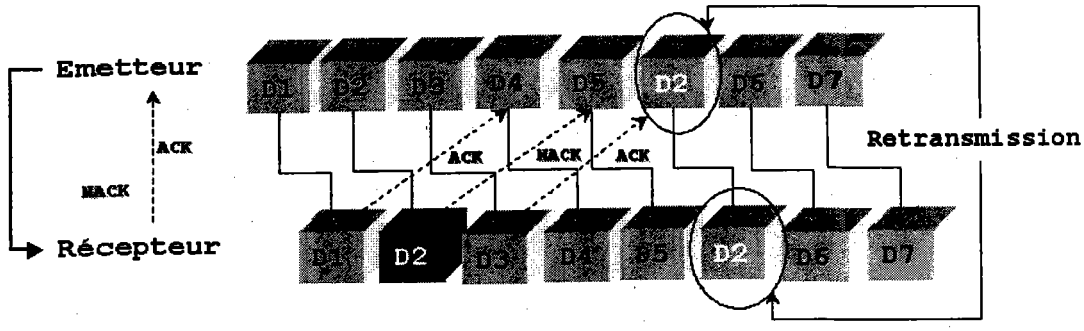



FIG. 1.13 – *Selective-repeat ARQ.*

### 1.7.2 Stratégies FEC

Les stratégies FEC se basent sur l'utilisation de codes permettant la détection, la localisation et la correction des erreurs sans retransmission. Technique largement utilisée dans les canaux unidirectionnels, sa capacité de correction en ligne en fait la plus adaptée aux transmissions orientées hauts débits. Cependant, le point critique des systèmes FEC réside dans la complexité des circuits de décodage qui pour la plupart se basent sur le principe du maximum de vraisemblance et visent à minimiser la probabilité d'erreur de décodage par mot (équ. (1.32)). Plusieurs type de codes, structures de codeur et stratégies de décodage ont étés adoptés selon le type d'application envisagée (cf. paragraphe 1.8).

### 1.7.3 Stratégies hybride ARQ/FEC

Les stratégies hybrides ARQ/FEC sont des systèmes intermédiaires. En effet, leur capacité de correction est supérieure aux stratégies FEC pures et leur débit de traitement supérieur aux stratégies ARQ. Dans le cas de systèmes ARQ en présence d'un canal très bruité, la retransmission des blocs erronés dégrade considérablement le débit. Une solution consiste à intégrer un code correcteur d'erreur dans le système permettant d'améliorer le débit tout en gardant la même capacité de correction. D'un autre point de vue, si le débit n'est pas un paramètre critique l'association de la retransmission à un système FEC permet d'améliorer les performances de décodage et même de réduire la complexité du décodeur. Dans [16] par exemple, il a été rapporté que l'association d'un décodeur de Viterbi et d'une retransmission appliqué aux codes convolutifs permettait de réduire le rapport signal sur bruit de 1 à 1.5 dB par rapport à un décodeur de Viterbi classique ou d'atteindre des taux d'erreurs (BER : *Bit Error Rate*) équivalents en réduisant de 50% la longueur de contrainte du code convolutif.

### 1.7.4 Comparaison des stratégies de codage canal

Le tableau 1.1 présente un bref comparatif des systèmes dédiées au codage canal. A noter que le choix d'une technique dépend des caractéristiques du canal de transmission (capacité, largeur de

bande, type de transmission, type d'erreur, etc.) et du cahier des charges imposé par l'application (débit, coût, complexité, tau de correction, latence, etc.) [8]. On retrouve par exemple les systèmes ARQ dans les réseaux de commutation par paquets de données PSDN (*Packet-Switching Data Networks*) et les réseaux informatiques classiques, tandis que les systèmes FEC s'imposent dans la plupart des systèmes de communication unidirectionnelle et dans les systèmes de stockage de données (bande d'enregistrement, lecteur CD (*Compact Disk*), DVD (*Digital Versatile Disk*), etc.) [12].

| Système | ARQ | FEC | Hybride |
|---|---|---|---|
| **Avantages** | réalisation simple<br>système hautement fiable | débit constant<br>haut débit | débit > ARQ<br>complexité < FEC |
| **Inconvénients** | débit variable<br>dialogue | décodage complexe<br>système onéreux | débit variable<br>dialogue |
| **Transmission** | bidirectionnelle | unidirectionnelle | bidirectionnelle |

TAB. 1.1 – *Stratégies de codage canal.*

## 1.8 Techniques de codage canal FEC

### 1.8.1 Codes blocs

Les codes blocs associent une redondance de taille $k'$ à chaque bloc d'information de taille $k$. Cette redondance est calculée à partir de ce même bloc par une loi linéaire, le mot de code obtenu étant de taille $n = k + k'$. Ils sont définis par les paramètres $(q, k, n, t)$ où $r = k/n$ est le rendement du code et $t$ le nombre d'erreurs corrigées par symbole q-aire.

Historiquement [12, 5] le premier code bloc est le « code de Hamming » qui à été inventé par Richard Hamming en 1946, puis amélioré et généralisé par Marcel Golay qui introduit à son tour les codes connus sous son nom: « codes de Golay ». On retrouve les codes de Hamming dans la norme bluetooth [15] qui utilise un code de Hamming (15,10) de polynôme générateur $G(x) = (1 + x)(1 + x + x^4)$. Le code de Golay binaire a été utilisé par la NASA dans la sonde spatiale Voyager I. Une autre classe importante des codes en bloc sont les codes Reed-Muller introduit par Muller en 1954 et reconnus plus tard par Reed qui proposa l'algorithme de décodage associé. Ils ont été utilisés par la suite dans les années 70 par la sonde spatiale Mariner qui avait pour mission d'aller sur Mars [12]. Les codes CRC (*Cyclic Redundancy Check*) sont apparus en 1957. Ils sont généralement utilisés pour la détection d'erreurs (protocoles X25, Ethernet, FDDI (*Fiber Distributed Data Interface*), ATM-AAL5[4]) [17], la correction s'effectuant par retransmission. Les mots de codes générés par les CRC peuvent être décodés en utilisant des décodeurs de type Meggitt [12]. Cependant, la complexité de ce type de décodeur augmente de façon exponentielle avec le nombre d'erreurs

4. AAL (*ATM Adaptative Layer*) est la troisième couche du protocole ATM (*Asynchronous Transfer Mode*). L'AAL-5, également connu sous le nom de SEAL (*Simple and Efficient Adaptation Layer*), est une des couches d'adaptation utilisées pour la transmission des données et des signalisations à débits variables (VBR : *Variable Bit Rate*) [130].

corrigées [5]. Les codes BCH découverts par Hocquenghem en 1959 ainsi que Bose et Chaudhuri en 1960 sont la généralisation des codes de Hamming pour la correction d'erreurs multiples. Ils ont été étendus au cas non-binaire (code RS) par Reed et Solomon en 1960. Grâce à leur nature non binaire l'utilisation des codes RS est adéquate dans le cas de canaux où les erreurs apparaissent par paquet (cf. paragraphe 1.4.3). L'algorithme de décodage de Berlekamp, introduit en 1967, a permis l'exploitation pratique des codes RS. De nos jours, ils sont largement utilisés dans les lecteurs CD, DVD et préconisés par le standard CDPD (*Cellular Digital Packet Data*) ainsi que par la norme DVB-C (*Digital Video Broadcasting-Cable*) concernant les systèmes de diffusion par câble qui utilisent un code RS ($k = 188, n = 204, t = 8$) [18].

Du point de vue de la conception architecturale, de nombreux codes en bloc on fait l'objet d'études pour améliorer leur débit. Des architectures parallèles, par exemple, ont été proposées pour les codes reed-solomon [19], les codes DSCC utilisés par la télévision japonaise [20] ainsi que pour les codes de redondance cyclique (CRC) en technologie FPGA (*Field Programmable Gate Array*) [21] ou ASIC (*Application Specific Integrated Circuits*) [22]. Dans [23], on propose une architecture ICC (*Iterative Combinatorial Cells*) dédiée à la division polynômiale alors que [24] présente une architecture pipeline de décodeur à logique majoritaire. Dans [25, 17], nous avons proposé une architecture parallèle/pipeline d'un codeur CRC qui a permis d'atteindre sur FPGA des débits de l'ordre de 4 Gbits/s[5].

Cependant, en dépit de leurs vastes champs d'application les codes en blocs présentent de nombreux inconvénients dont les principaux sont :

- de part la nature du codage qui se fait par bloc, la procédure de décodage ne peut se faire avant que le mot de code complet ne soit reçu, d'où la génération d'une latence proportionnelle à la longueur du mot de code ;
- pour que le décodage soit correct, il est nécessaire de synchroniser le système sur le premier symbole du mot de code reçu ;
- En général le décodage se fait uniquement sur les décisions binaires (*hard decision*) fournies par le démodulateur et non sur les sorties pondérées (*soft decision*) ce qui limite les performances de décodage à des rapports signal sur bruit moyens et grands. En effet, le décodage sur les décisions pondérées apporte une amélioration en puissance de l'ordre de 2,5 dB dans le cas d'un code de rendement $1/2$ [5].

## 1.8.2 Codes convolutifs

L'approche du codage convolutif consiste à calculer la redondance à partir du bloc d'information courant de taille $k$ et des $m$ blocs précédents. Les $n$ bits en sortie sont calculés par une combinaison linéaire entre les $k$ bits en entrée et les $m$ blocs précédents. Le rendement du code est $r = k/n$ et sa longueur de contrainte $K_c$ est le nombre maximum de bits associés à une sortie qui peuvent être

5. Résultat obtenu sur un Flex10KE d'Altera ($f_{max} = 250$ MHz) et pour un degré de parallélisation de $\wp$=32.

affectés par un bit quelconque à l'entrée.

Toutes les propriétés des codes en blocs s'appliquent à ce type de codage ainsi qu'a son opération inverse, le décodage. La contrainte principale du décodage convolutif réside dans le fait que le mot de code est très long, ce qui a tendance à compliquer le circuit décodeur. Les algorithmes de décodage les plus répandus sont :

**Décodage séquentiel :** il a été proposé par Wonzencraft et Reiffen en 1961 puis amélioré en 1963 par Fano et en 1968 indépendamment par Jelinek et Zigangorov [12, 26]. Le nombre d'itérations nécessaires pour le décodage d'un bloc est considéré comme une variable aléatoire : bien que la plupart des trames soient décodées rapidement, il se peut que le décodage soit défectueux ou la cause d'effacements, ce qui lui vaut l'appellation de méthode probabilistique de décodage. Quant à la complexité du décodage, elle est indépendante de la longueur de contrainte du code. Ce type de décodage est approprié au cas des canaux sans mémoire [27].

**Algorithme de Viterbi :** proposé en 1967 par A. J. Viterbi [28], il est basé sur le principe du maximum de vraisemblance. Cet algorithme est une méthode optimale de décodage pour les codes convolutifs, ses performances dépendant de la qualité du canal de transmission utilisé. Par contre la complexité des systèmes de décodage augmentant exponentiellement avec la longueur de contrainte du code utilisé restreint leur emploi aux applications où le code a une petite longueur de contrainte.

Cependant, le développement de l'algorithme de Viterbi a permis l'exploitation pratique des codes convolutifs, notamment dans les systèmes de télécommunications. Ils ont par exemple été utilisés dans les applications commerciales des satellites de communication [29] ($K_c = 7$ et R=1/2 et 1/3), ainsi que dans les applications de satellites militaires (DSCS : *Defense Satellite Communication System*) [30] ($K_c = 6$ et R=1/2). On les retrouve aussi dans les sondes spatiales d'exploration comme Voyager [31]. Les codes convolutifs ont été recommandés en télémétrie par le standard CCSDC (*Consultative Committee for Space Data Systems*) [32] et préconisés par HIPERLAN/2 et IEEE 802.16 dans les applications réseaux sans fil LAN/WAN [33]. De plus tous les standards de téléphonie mobile de deuxième génération préconisent l'utilisation de codes convolutifs. La norme GSM (*Global System for Mobile Communications*) incorpore un code de longueur de contrainte $K_c = 6$ et de rendement 1/2. La norme USDC[6] utilise un code avec $K_c = 6$ et un rendement 1/2. La norme IS-95 emploie un code avec $K_c = 9$ et un rendement de 1/2 en canal montant et 1/3 en canal descendant, Globalstar[7] utilise un code de rendement 1/2 avec $K_c = 9$, et pour Iridium[8] le rendement est de

6. USDC est aussi connu sous les Standards IS-54 et IS-136.

7. Le système Globalstar est un réseau de satellites de communications évoluant en orbite basse (LEO : *low earth orbit*) destiné à fournir des services de téléphonie personnelle (voix, données, fax, GPS, etc.). Il comprend 48 satellites de durée de vie de 9 ans. Il a été mis en service en 1999 et est exploité par les opérateurs Loral, Qualcomm, Alcatel, Vodafone et France Telecom [34].

8. De même que Globalstar, Iridium est un système de satellite évoluant en orbite basse conçu pour proposer des services similaires. Le réseau compte aujourd'hui 66 satellites actifs de durée de vie de 5 ans évoluant à une hauteur de

3/4 et la longueur de contrainte $K_c = 7$ [5, 35].

**Décodage par seuil (logique majoritaire) :** le décodage par logique majoritaire appliqué aux codes convolutifs a été proposé par Massey en 1963 [12]. Il diffère des décodages de Viterbi et séquentiel par le fait que la décision finale prise pour un bloc donné est seulement basé sur la longueur de contrainte du bloc en cours de décodage plutôt que sur l'utilisation de toute la séquence reçue, ce qui conduit à des performances de décodage inférieures aux deux autres méthodes. Ses points forts résident dans la simplicité de son implantation et sa rapidité de décodage. Le décodage de syndrome est approprié pour des transmissions sur des canaux a mémoire où les erreurs apparaissent par paquet. Il est plutôt utilisé dans des applications qui ne demandent pas de grands gains de décodage. On le retrouve dans des applications telles que la téléphonie ou la radio HF [36]. Il est aussi utilisé dans les systèmes de télécommunication à haut débit [12].

### Comparaison des algorithmes de décodage convolutif

Le tableau 1.2 donne un bref comparatif des trois principaux algorithmes de décodage des codes convolutifs.

| **Décodeur** | **Séquentiel (SEQ)** |
|---|---|
| **Performance** | LM < SEQ < Viterbi |
| **Vitesse** | en moyenne 1 à 2 cycles/bit décodé<br>risque de décodage défectueux + effacement |
| **Latence** | L=(L'+m) cycles |
| **Complexité** | Indépendant de $m$<br>LM < SEQ |
| **Décodeur** | **Viterbi** |
| **Performance** | LM < SEQ < Viterbi |
| **Vitesse** | $2^m$ cycles/bit décodé |
| **Latence** | L=(L'+m) cycles |
| **Complexité** | $2^m$ registres<br>taille $\approx$ (trame d'information + métrique) |
| **Décodeur** | **Logique Majoritaire (LM)** |
| **Performance** | LM < SEQ < Viterbi |
| **Vitesse** | 1 cycle/bit décodé |
| **Latence** | $K_c$ cycles |
| **Complexité** | réplique codeur +<br>porte LM + quelques portes logiques<br>LM < Viterbi |

TAB. 1.2 – *Comparaison des techniques de décodage convolutifs.*

780 km. Il a été mis en service en 1998 et est exploité par Motorola/Raytheon, Sprint et BCE Mobile [34].

L'algorithme de Viterbi est l'approche la plus optimale du point de vue performance de décodage. L'algorithme séquentiel a des performances proches mais en présence de trames très bruitées le temps de décodage peut augmenter considérablement avec des risques de perte d'information et d'effacement. Quant au décodage à logique majoritaire, il est le moins performant car il ne considère pas la totalité du mot de code lors du décodage mais uniquement la longueur de contrainte.

Cependant, la mise en œuvre d'un décodeur à logique majoritaire reste la plus simple, ce qui le rend très attractif dans les applications de bas coût. En effet, il est constitué d'une réplique exacte du codeur, d'une fonction à logique majoritaire et de quelques portes logiques. La complexité d'un décodeur de Viterbi augmente exponentiellement avec la taille $m$ de sa mémoire[9]. La procédure de décodage impose l'utilisation de $2^m$ registres dont la taille dépend de la longueur de la séquence reçue. En pratique les codes sont limités à des valeurs de $m = 8$. Dans le cas séquentiel, la complexité du décodeur est indépendante de $m$ mais elle reste supérieure à celle du décodeur à logique majoritaire.

Le temps de latence dans les décodeurs de Viterbi et séquentiel dépend de la longueur du mot de code reçu $L = (L' + m)$ où $m$ est le nombre de bits additionnels pour assurer la terminaison du treillis[10]. Cependant, des variantes des deux algorithmes permettent de réduire cette latence à 4 ou 5 fois la longueur de contrainte du code avec une dégradation acceptable de la performance de décodage. Dans le cas de la logique majoritaire, la latence reste la plus petite et est égale à $K_c$. De plus, un bit est décodé à chaque cycle contre 1 à 2 cycles dans le cas séquentiel et $2^m$ cycles dans le cas Viterbi. Cependant la parallélisation du décodeur de Viterbi permet de réduire ce temps d'un facteur $2^m$.

Toutefois, dans les applications où une grande longueur de contrainte est indispensable, la complexité des décodeurs augmente considérablement quel que soit le type utilisé. Dans ce cas, l'avantage en termes de compromis complexité/performance est pour les décodeurs séquentiel et Viterbi. Les décodeurs de Viterbi sont facilement adaptables au traitement de décision pondérée (*soft decision*), tandis que dans le cas de décodeurs à logique majoritaire l'adaptation se fait au dépend d'une augmentation considérable de la complexité [12].

### 1.8.3 Modulation codée en treillis

Dans un système de codage classique de rendement $R < 1$, le débit après codage est réduit d'un facteur $R$ par rapport au même système non codé. Donc si l'on veut maintenir le débit après codage il est nécessaire d'augmenter la bande passante du système (cf. remarque paragraphe 1.5). La modulation codée en treillis (TCM : *Trellis Coded Modulation*) introduite par Ungerboeck [37, 38] permet de remédier à cette contrainte liée au codage. Le principe consiste à fusionner les étapes du codage canal et de la modulation en un seul bloc de façon à maintenir la totalité de la bande après codage. L'approche originelle associe un code convolutionnel et une modulation multidimensionnelle

9. $m$ est la somme des tailles $m_i$ des registres du codeur.
10. La notion de treillis et l'opération de la terminaison du treillis sont définies dans le paragraphe 2.3.1

(N-QAM, QPSK, etc.). L'utilisation de la TCM dans les modems a permis d'obtenir des débits au-delà de 9600 bits/s (Normes V.32, V.34, V.90, etc.). Elle est aussi utilisée dans les applications spatiales telle que les communications satellite. Plus récemment, avec l'apparition des turbo-codes[11] [2], on retrouve des systèmes T-TCM qui substituent au code convolutionnel un turbo-code [39, 40].

### 1.8.4 Codes entrelacés

L'entrelacement ou opération de génération de permutations consiste à changer selon l'application l'ordre des symboles du mot d'information ou du code, de manière déterministe afin de pouvoir les remettre dans l'ordre lors du décodage. La réorganisation des bits permet d'étaler les erreurs qui surviennent par paquets lors d'une transmission (cf. paragraphe 1.4.3). L'utilisation par exemple d'un entrelacement avec un code convolutif diminue sa susceptibilité aux paquets d'erreurs. Les générateurs de permutations sont aussi utilisés dans le cas des turbo-codes afin de minimiser les séquences susceptible de générer des mots de codes à poids faible.

L'entrelacement périodique engendre des permutations selon une fonction périodique du temps. On peut le réaliser, soit par de la mémoire vive (entrelacement par blocs), technique utilisée dans tous les standards de téléphonie de deuxième génération, ou par des registre à décalage (entrelacement convolutionnel). L'entrelacement pseudo-aléatoire consiste à écrire la séquence aux adresses successives d'une mémoire vive, puis d'effectuer la lecture suivant un ordre donné par un générateur pseudo-aléatoire. L'opération inverse consiste à écrire dans la mémoire suivant le même ordre pseudo-aléatoire, et ensuite de lire la séquence suivant l'ordre croissant des adresses de la mémoire.

### 1.8.5 Codes concaténés

La concaténation de codes permet d'augmenter la puissance des systèmes de codage au prix d'une augmentation de la complexité globale. La concaténation peut se faire de trois façons : parallèle, série ou hybride (parallèle et série) et sur deux ou plusieurs niveaux [6]. Dans le cas d'une structure série à deux codes, l'information est codée deux fois. Une première fois par le premier code appelé code externe, puis une seconde fois par le deuxième, dit code interne [8]. Les deux codes utilisés sont en général complémentaires : les codes convolutifs par exemple sont susceptibles aux erreurs qui apparaissent par paquets alors que les codes RS sont adéquats pour ce type d'erreurs. Dans ce cas, le décodeur convolutif s'occupera des erreurs aléatoires pour de faibles rapports signal sur bruit tandis que le décodeur RS s'occupera des erreurs par paquets pour de rapports signal sur bruit élevés. Ce type de concaténation série a été proposée par David Forney en 1966 [41] puis standardisé en 1987 pour les communications spatiales dans les réseaux DSN (*Deep Space Network*) par les agences spatiales NASA (*National Aeronautics and Space Agency*) et ESA (*European Space Agency*), préconisant l'utilisation d'un code RS ($q = 8$, $k = 223$, $n = 255$, $t = 16$). Il a aussi été utilisé dans les systèmes

11. Les turbo-codes sont introduits dans le paragraphe 1.8.6.

de diffusion par satellite (norme DVB-S) [42] et de diffusion hertzienne terrestre (norme DVB-T) [43] qui employe un code RS externe ($k = 188, n = 204, t = 8$) et un code convolutif interne de rendement $R = 1/2, 2/3, 3/4, 7/8$ séparés par un entrelacement. La concaténation d'un code RS externe et d'un code orthogonal interne permettrait de transmettre à faible puissance cependant, ils ne sont utilisables que sur de larges bandes de fréquence car ils ont un faible taux de transmission [8]. Les codes produits, inventés par P. Elias en 1954 [44], sont construits par la concaténation de deux ou plusieurs codes en blocs linéaires à faible pouvoir de correction. En général, les codes utilisés sont les codes BCH et Hamming.

### 1.8.6 Turbo-codes

Les turbo-codes [2] peuvent être considérés comme une classe des codes concaténés. Dans [3], on regroupe dans cette classe tous les codes capables d'être décodés par un décodeur de type SISO (*Soft-Input Soft-Output*), la concaténation série proposée par Forney étant exclue de cette classe car le code RS ne peut pas être décodé par SISO.

Un turbo-code est la concaténation d'un ou de plusieurs codes (convolutifs ou blocs) séparés par des blocs d'entrelacements. Les turbo-codes à plusieurs niveaux présentent un faible rendement par rapport aux turbo-codes à deux niveaux, ce qui explique leur rareté dans les applications. Quant aux turbo-codes hybrides, ils ne présentent pas un grand intérêt car les niveaux de performances atteints pas les turbo-codes parallèles et série sont très suffisants pour les applications actuelles [3].

La première structure pour un turbo-code a été proposée en 1993 par Berrou, Glavieux et Thitimajshima [2]. Le turbo-code présenté était la concaténation parallèle de deux codes convolutifs récursifs et systématiques (RSC) identiques, de rendement $1/2$, reliés par un bloc d'entrelacement. Les codes de type RSC constituent une sous classe des codes convolutifs qui possèdent une réponse impulsionnelle infinie du fait de leur récursivité et dont une des sorties est dédiée au message lui-même (sortie systématique) (cf. paragraphe 2.3.6). Les turbo-codes ont permis d'atteindre des taux d'erreurs de $10^{-5}$ pour un niveau de signal sur bruit égal à 0.7 dB [2, 46]. Au vu de ces résultats, ce sont les codes qui offrent les meilleures performances de décodage en se rapprochant le plus de la limite fixée par le théorème fondamental de Shannon. Les turbo-codes, au sens strict, sont la concaténation parallèle de codes RSC. Ils englobent cependant aussi la concaténation de codes en blocs [4]. Dans [3], les codes en blocs concaténés avec un entrelacement aléatoire sont appelés turbo-codes en bloc et ceux avec entrelacement lignes-colonnes, codes produits. Dans [4], les codes produits sont désignés sous le vocable de code en bloc, la justification étant que le principe du décodage itératif des codes produits est semblable à celui des turbo-codes convolutifs.

**Applications des turbo-codes**

Les performances remarquables des turbo-codes leur ont valu d'être l'objet de nombreuses normalisations et d'être retenus dans de nombreuses applications. En 1999 par exemple, le programme IMT-2000 concernant les spécifications de téléphone mobiles de $3^{eme}$ génération [47] établies par l'UIT (Union Internationale des Télécommunications) les a retenus pour ses normes WCDMA (UMTS)[12] [48], compatible avec les réseaux GSM, et CDMA2000[13], compatible avec les réseaux IS-95. En 2000, les turbo-codes ont été choisis par le standard DVB (*Digital Video Broadcasting*) [49] dans les systèmes de diffusion de vidéo numérique, concernant le codage des canaux de retour par voie terrestre (DVB-RCT) et satellitaire (DVB-RCS). Le turbo-code utilisé [50] fonctionne à des débit compris entre 144 Kbits/s et 2 Mbits/s pour des rendements 1/3, 2/5, 1/2, 2/3, 3/4 et 4/5. Il permet de coder des blocs d'information de taille comprise entre 12 et 216 octets et supporte des trames ATM (53 octets) ou MPEG (188 octets). En 2001, les turbo-codes ont été retenus par les normes Européennes ETSI Hiperacces et Américaine IEEE 802.16.1 comme codage optionnel pour augmenter le débit dans les systèmes de boucle locale radio à haut débit. Ils ont aussi été recommandés par le standard CCSDS (*Consultative Committee on Space Data System*) [131] pour le codage de canaux en télémetrie [32]. Cette nouvelle recommandation apporte une amélioration de 1,5 à 2 dB par rapport aux performances du code utilisé précédemment construit par la concaténation d'un code RS et convolutif [51]. Les turbo-codes recommandés utilisent des codes constituants RSC à 16 états avec des rendements de 1/2, 1/3, 1/4 et 1/6. La taille des entrelaceurs varie de 1784 bits à 16384 bits. Les turbo-codes sont aussi en vue pour être utilisés dans les réseaux de satellites de communication tel que Inmarsat[14], dans le cadre de l'amélioration de la qualité du service dans les offres multimedia [52, 53, 54] ou Intelsat[15] pour des transmissions à 2 Mbits/s [55]. Les agences spatiales américaine NASA et européenne ESA ont utilisé les turbo-codes respectivement dans le satellite d'exploration SOHO et pour la mission SMART-1 en 2001. De plus la NASA envisage de les utiliser dans les sondes spatiales destinées aux prochaines missions en espace profond (DSN) [56]. Les turbo-codes sont aussi étudiés dans le cadre des technologies xDSL, notamment en ADSL (*Asymmetric Digital Subscriber Line*) exploitant la modulation DMT (*Digital MultiTone*). Dans [57, 58] on démontre qu'en utilisant un code concaténé se composant d'un code externe RS et en remplaçant le code interne préconisé par le standard G.922.1 par un turbo-code permettrait une amélioration significative du gain de codage pour

12. Standard né de la collaboration de l'ETSI (*European Telecommunications Standards Institute*) en Europe, l'ARIB (*Association of Radio Industries and Business*) au Japon et le TTA (*Telecommunication Technology Association*) en Corée du Sud.

13. Standard développer par le TIA (*Telecommunication Industry Association*) aux Etats-Unis.

14. Le système Inmarsat est un réseau de communication basé sur onze satellites géostationnaires. Il permet la fourniture de services : voix, fax, transmission de données. En prévision de 2005 un nouveau système Inmarsat I-4 permettra d'intégrer la norme B-GAN (*Broadband Global Area Network*) et qui offrira entre autre des débits de 432 Kbit/s pour l'accès à internet sur téléphone mobile.

15. INTernational TELecommunications SATellite consortium. C'est une organisation internationale basée aux Etats-Unis et ayant longtemps possédé le plus grand réseau de communications par satellite du monde.

un BER de $10^{-7}$ [59]. Les turbo-codes sont aussi étudiés dans le cadre de canaux d'enregistrement magnétique [60, 61, 62] ; ils ont permis d'atteindre, pour un BER de $10^{-5}$, des gains de codage de 7,1 dB pour des rendements de 4/5 et 8/9 et 6,5 dB pour un rendement de 16/17. Enfin les turbo-codes ont fait l'objet de nombreux dépôts de brevet par France Télécom dont certains conjointement avec TDF (Télédiffusion De France).

## 1.9 Comparatifs de quelques systèmes de codage FEC

Dans le cas d'une modulation BPSK et en considérant une transmission sans codage sur un canal AWGN, un rapport signal sur bruit minimum de 9.6 dB est requis pour atteindre un BER de $10^{-5}$. Un système utilisant un turbo-code permet de réduire le rapport signal sur bruit requis à 0.7 dB pour atteindre le même BER de $10^{-5}$ [2], soit un gain d'environ 9 dB. La figure 1.14 donne pour quelques systèmes utilisés dans divers domaines, les gains obtenus après codage pour un taux d'erreurs de $10^{-5}$. A noter que ces systèmes n'utilisent pas forcément une modulation BPSK.

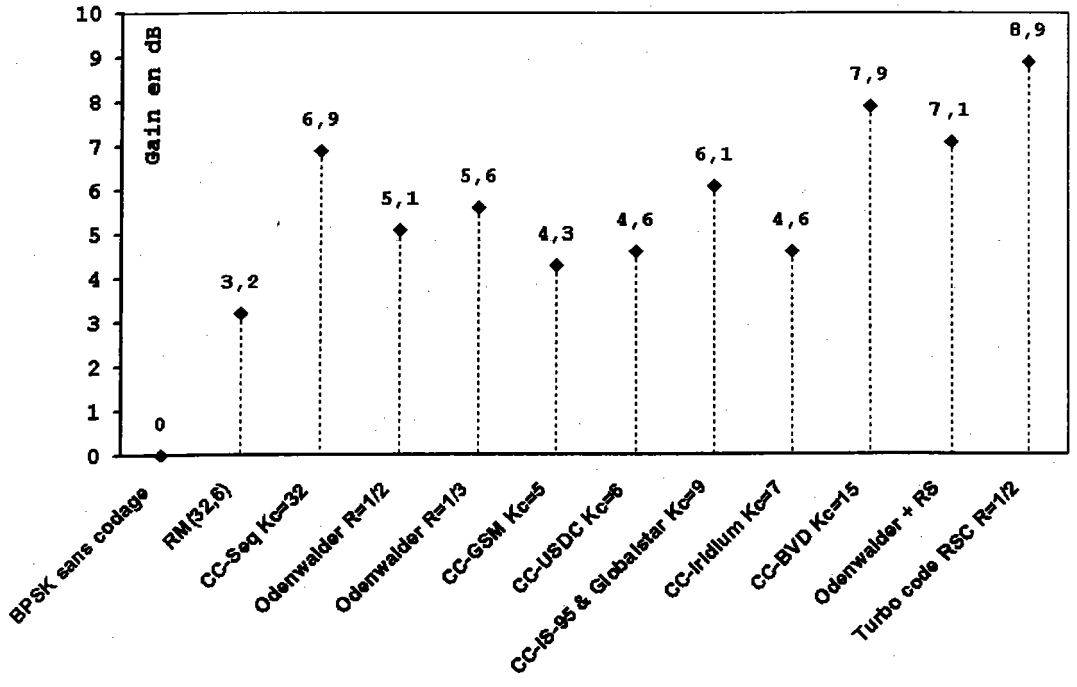



FIG. 1.14 – *Gain en dB de quelques systèmes de codage par rapport à une BPSK sans codage.*

Le code Reed-Muller RM (32, 16) utilisé en 1969 par la sonde spatiale Mariner apporte un gain de 3.2 dB pour un rendement R= 0,1875. Le code convolutif (CC) de longueur de contrainte $K_c = 32$ avec décodage séquentiel utilisé par les sondes spatiales Pionner 10 en 1972, Pionner 11 en 1973 et Pionner 12 en 1977 offre un gain de 6.9 dB. Les codes Odenwalder [63], qui font partie de la famille des codes convolutifs, ont été utilisés dans de nombreux satellites de communications. Le

code Odenwalder de rendement R=1/2 apporte un gain de 5.1 dB, tandis que celui de rendement R=1/3 présente un gain de 5.6 dB. Le code convolutif avec $K_c = 5$ utilisé dans la norme GSM offre un gain de 4.3 dB. Celui utilisé par la norme USDC avec $K_c = 7$ atteint les 4.6 dB et celui des normes IS-95 et Globalstar avec $K_c = 9$ atteint les 6.1 dB. La norme Iridium utilise un code de rendement 3/4 avec $K_c = 7$ et offre un gain de 4.6 dB [35, 5]. Le code convolutif avec $K_c = 15$ et R=1/4 utilisé par la NASA lors de la mission Galileo à destination de Jupiter utilise un décodeur de Viterbi (BVD : *Big Viterbi Decoder*) et offre un gain de 7.9 dB [64]. Le code concaténé convolutionnel de rendement 1/2 et Reed-Solomon ($2^8$, 223, 255, 16) atteint un gain de 7.1 dB pour un rendement de 0.44 [65].

## 1.10 Conclusion

Ce chapitre a introduit la théorie de l'information et démontré la nécessité d'établir une stratégie de protection des données permettant de fiabiliser la transmission. Pour cela, trois approches ont été décrites. Les stratégies ARQ sont simples à mettre en œuvre et permettent une correction totale des informations altérées. Cependant ce sont des systèmes qui ne permettent pas d'atteindre de très hauts débits, la limitation étant due au dialogue nécessaire entre l'émetteur et le récepteur. Cette solution est donc écartée. Les stratégies à base de codes correcteurs d'erreurs (FEC) offrent la possibilité de transmettre à de hauts débits en effectuant une correction en ligne sans interruption. Quant aux solutions hybrides, elles présentent une solution intermédiaire. Elles restent cependant moins performantes en termes de débits de transmission par rapport aux objectifs à atteindre. Dans le contexte FEC, l'énumération des différents systèmes de codage a permis de constater que les codes convolutifs sont très utilisés en télécommunication, soit comme entités indépendantes, soit dans des structures concaténées avec d'autres codes, soit encore comme éléments constituants de turbo-codes. De plus, les turbo-codes permettent d'atteindre des gains de codage très élevés pour des rapports signal sur bruit très faibles. Par conséquent, la suite portera sur une étude approfondie des codes convolutifs et de leur utilisation comme codes constituants dans des turbo-codes convolutifs.

# Chapitre 2

# Codes convolutifs : application aux turbo-codes

Selon le type d'application et le cahier des charges imposé, diverses techniques FEC peuvent être utilisées pour protéger l'information lors d'une transmission à travers un canal bruité. Comme vu au chapitre 1, les codes convolutifs sont très utilisés dans le contexte des télécommunications. De plus, les turbo-codes convolutifs, pour un taux d'erreurs de $10^{-5}$, permettent des transmissions sur des canaux dont le rapport signal sur bruit est de l'ordre de 0.7 dB [2] contre des transmissions à 2.4 dB pour les codes utilisés précédemment [66]. Cependant l'obtention de telles performances dépend fortement des éléments constituants du turbo-codeur, c'est-à-dire de ses codes constituants d'une part, et du type et de la taille des entrelaceurs utilisés d'autre part [67, 68]. Un autre critère déterminant est le choix de l'algorithme de décodage dont la complexité croit proportionnellement avec les performances recherchées (c.f chapitre 3). Dans la première partie de ce chapitre, nous nous intéresserons aux architectures séries « classiques » des codeurs convolutifs et aux critères de choix justifiant leur utilisation comme codes constituants dans un turbo-code. Ensuite, nous aborderons les différentes structures de turbo-codes et les paramètres déterminant leurs performances.

**Mots clés :** codes blocs, codes convolutifs, turbo-codes.

## 2.1 Introduction

Bien qu'ils soient différents des codes blocs, les codes convolutifs présentent de nombreuses similitudes architecturales avec eux, notamment avec les codes cycliques. En effet, les deux peuvent être réalisés avec des registres à décalages et sont définis par leurs polynômes générateurs. Cependant, une de leurs différences fondamentales réside dans le fait qu'un code bloc a une longueur fixe alors qu'un code convolutif n'a pas de longueur fixe. Néanmoins, pour des raisons pratiques on limite la taille des trames d'information traitées à des longueurs qui permettent de maintenir raisonnable la complexité

des circuits. Dès lors, les turbo-codes peuvent être assimilés à des codes blocs (cf. paragraphe 2.4). Aussi, avant d'aborder l'étude des codes convolutifs, ce chapitre traite des codes blocs et introduit les architectures dites MTO et OTM des circuits de codage par multiplication et par division. Cette étude permettra de faire une transition directe vers les codes convolutifs et les différentes variantes d'architectures de codeurs convolutifs, notamment la structure récursive systématique de ces derniers qui est adoptée dans un turbo-codeur.

## 2.2 Codes blocs

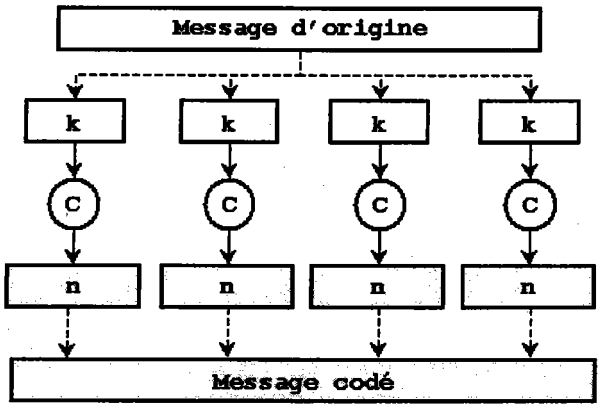



FIG. 2.1 – *Codage en bloc.*

Le codage en bloc consiste à fractionner la trame d'information en plusieurs blocs de taille fixe $k$, ensuite, de transformer chacun des messages $D_i$ en un mot de code $C_i$ de taille $n$ en appliquant une loi linéaire (fig. 2.1). La redondance associée à chaque bloc est de taille $k'$, où $k + k' = n$. Le rendement d'un code bloc est donné par la formule (2.1).

$$R = \frac{k}{n} \tag{2.1}$$

où, $k$ et $n$ représentent respectivement les nombres de bits en entrée et en sortie du codeur.

### 2.2.1 Capacité de détection et correction d'un code

**Distance de Hamming :**

La distance de Hamming représente le nombre de positions identiques dans deux mots de code contenant des bits de valeurs différentes. On calcule cette distance en effectuant l'addition modulo 2 entre les deux mots de code, puis en comptant le nombre de 1 dans le résultat. La distance de Hamming entre le mot de code lui-même et le mot de code 0 est appelée « poids de Hamming ». En d'autres termes, le poids de Hamming est le nombre de 1 dans un mot de code.

**Distance minimale :**

La distance minimale d'un code est la distance de Hamming la plus petite entre deux mots de code $c_i$ et $c_j$ appartenant au même ensemble.

$$d_{min} = \min_{i \neq j}(c_i, c_j) \tag{2.2}$$

Pour détecter $t_e$ erreurs, il est nécessaire d'utiliser un code dont la distance minimale $d_{min}$ est au moins égale à $t_e + 1$.

$$\text{détection de } t_e \text{ erreurs} \Rightarrow d_{min} \geq t_e + 1 \tag{2.3}$$

Pour corriger $t_e$ erreurs il faut utiliser un code dont la distance $d_{min}$ est au moins égale à $2t_e + 1$.

$$\text{correction de } t_e \text{ erreurs} \Rightarrow d_{min} \geq 2t_e + 1 \tag{2.4}$$

### 2.2.2 Codes cycliques

Un code est dit cyclique, si pour chaque décalage cyclique d'un mot de code $C_1 = (c_0, c_1, \cdots, c_{n-1})$ on obtient un autre mot de code $C_2 = (c_{n-1}, c_0, \cdots, c_{n-2})$ valide.
Si $C_1(x) = c_0 + c_1 x + \cdots + c_{n-1}x^{n-1}$ est le polynôme associé au mot de code $C_1$ et $C_2(x) = c_{n-1} + c_0\, x + \cdots + c_{n-2}x^{n-1}$ celui associé à $C_2$ alors si $x^n = 1$ ($x^n$-1= $x^n$+1= 0)[1], $C_2(x) = x \cdot C_1(x)$. Par conséquent la permutation cyclique d'une position s'exprime par l'équation (2.5) et la généralisation à $i$ positions par l'équation (2.6).

$$C_2(x) = x \cdot C_1(x) \bmod [x^{n+1}] \tag{2.5}$$

$$C_{i+1}(x) = x^i \cdot C_1(x) \bmod [x^{n+1}] \tag{2.6}$$

Comme tous les codes, un code cyclique est totalement défini par sa matrice génératrice[2] G :

$$G = \begin{pmatrix} g_0 & g_1 & \cdots & g_m & \cdots & 0 & 0 \\ 0 & g_0 & \cdots & \cdots & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & \cdots & \cdots & g_m & 0 \\ 0 & 0 & \cdots & g_0 & \cdots & g_{m-1} & g_m \end{pmatrix}$$

où $m$ représente la taille de la mémoire du codeur (le nombre de bascules D).

La représentation matricielle s'avère fastidieuse lorsque le nombre de coefficients de $G$ augmente. Une autre manière de spécifier un code cyclique est la représentation polynomiale. En effet, un code

1. Dans le corps de Galois CG(2) l'addition et la soustraction se confondent.
2. Par convention les polynômes et matrice génératrice liés à une boucle récursive seront identifiés par la notation « G », alors que dans le cas d'une branche directe ils seront notés par « H ».

cyclique peut être défini par son polynôme générateur $G(x) = g_0 + g_1x^1 + g_2x^2 + \cdots + g_mx^m$, le mot de code par son polynôme correspondant $C(x)$ et le message par son polynôme $D(x)$. Les codes cycliques sont implantés à l'aide de registres à décalage linéaire.

### Codage par multiplication

Le codage par multiplication consiste à multiplier le bloc d'information par un multiplicateur fixe, le mot de code étant le résultat de la multiplication. L'opération de codage est définie par la multiplication polynomiale $C(x) = D(x) \times H(x)$, ce qui se traduit par une convolution discrète (équ. (2.7)) des coefficients du bloc d'information $D(x)$ et du polynôme générateur $H(x)$.

$$c_i = \sum_{j=0}^{m} d_{i-j} \cdot h_j \tag{2.7}$$

Le circuit correspondant à ce type de codage est un circuit multiplieur de polynôme, dans lequel le polynôme générateur $H(x)$ (multiplicateur) est fixe et le mot d'information $D(x)$ (multiplicande) variable. Dans ce cas le mot d'information attaque le circuit par le bit de poids faible en premier.

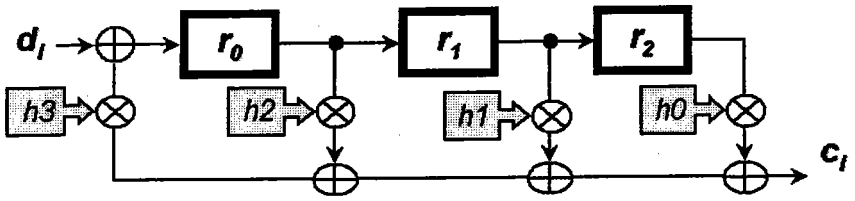



FIG. 2.2 – *Multiplieur série MTO pour* $H(x) = h_0 + h_1x^1 + h_2x^2 + h_3x^3$.

La figure 2.2 représente l'architecture série dite *Many To One* (MTO) d'un circuit multiplieur pour un polynôme générateur $H(x) = h_0 + h_1x^1 + h_2x^2 + h_3x^3$. Le principal inconvénient de ce type d'architecture réside dans le fait que le chemin critique de cette structure dépend fortement du nombre de coefficients $h_i$ non nuls. En effet ce dernier fixe le nombre de portes XOR misent en cascade[3], ce qui a pour effet d'augmenter le chemin critique en fonction du nombre de coefficients $h_i$ qui prennent pour valeur 1. Cependant cet inconvénient peut être évité en adoptant une structure dite *One To Many* (OTM).

Comme le montre la figure 2.3, le chemin critique de cette structure est équivalent à celui d'une seule porte XOR, quel que soit le nombre de coefficients $h'_i$ non nuls du polynôme générateur $H'(x)$. La seule condition à respecter lors du passage d'une structure à l'autre est d'intervertir les coefficients selon l'équation (2.8).

$$h_i = h'_{j-i}, \quad \text{pour } i \in \{0,1,\cdots,j\} \tag{2.8}$$

3. La multiplication se traduit par une connexion simple ou pas de connexion suivant que le coefficient correspondant est un 1 ou 0.

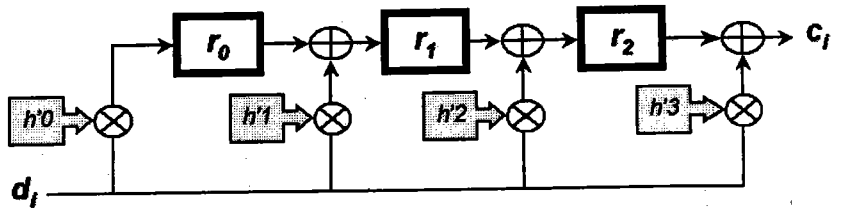



FIG. 2.3 – *Multiplieur série OTM pour* $H'(x) = h'_0 + h'_1x^1 + h'_2x^2 + h'_3x^3$.

**Codage par division**

L'algorithme de division du bloc d'information (dividende) $D(x)$ par le polynôme générateur (diviseur) $G(x)$ dans CG(2) revient à une succession de tests sur le bit de poids fort du dividende. Si ce bit est nul, alors le dividende est décalé d'une position. Si par contre le bit de poids fort n'est pas nul, alors on additionne le diviseur au mot d'information, puis on décale ce dernier.

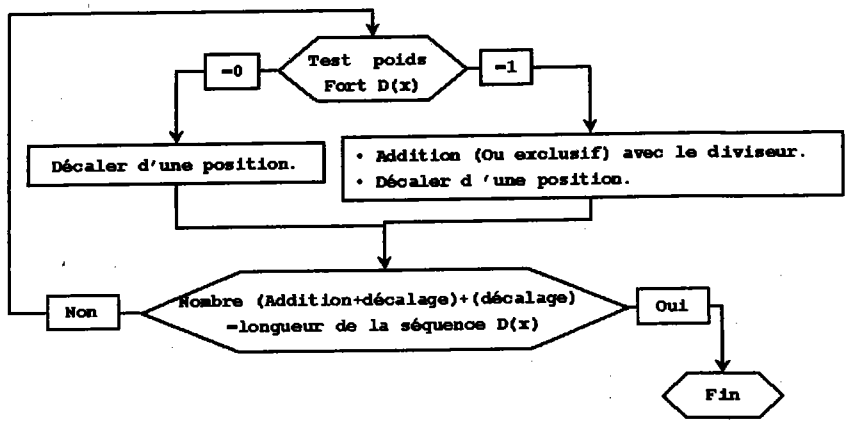



FIG. 2.4 – *Algorithme de la division dans le corps de Galois CG(2).*

Matériellement comme pour le cas d'un multiplieur, les décalages peuvent se faire à l'aide d'un registre à décalage. Le diviseur est matérialisé par une boucle récursive dont les coefficients dépendent du polynôme qui lui est associé. A chaque cycle d'horloge, un bit du quotient de la division est calculé. A la fin du codage, le reste de la division est disponible dans le registre constitué des bascules $(r_0, r_1, \cdots, r_{m-1})$.

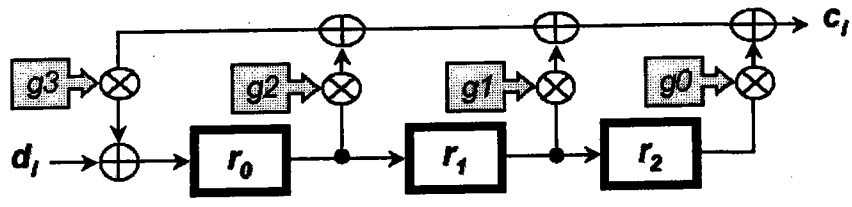



FIG. 2.5 – *Diviseur série MTO pour* $G(x) = g_0 + g_1x^1 + g_2x^2 + g_3x^3$.

La figure 2.5 représente l'architecture série *Many To One* (MTO) d'un diviseur pour un polynôme générateur $G(x) = g_0 + g_1x^1 + g_2x^2 + g_3x^3$. Par analogie avec le multiplieur, le point faible de cette structure réside dans le nombre de coefficients $g_i$ non nuls qui fixe le nombre de portes XOR mises en cascade.

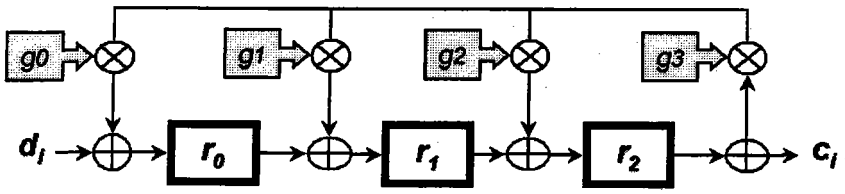



FIG. 2.6 – *Diviseur série OTM pour* $G(x) = g_0 + g_1x^1 + g_2x^2 + g_3x^3$..

La structure *One To Many* (OTM) du diviseur série (fig. 2.6) permet de réduire le chemin critique à une seule porte XOR. Le passage d'une structure à l'autre se fait en respectant les mêmes conditions que pour le codeur multiplieur (équ. (2.8)).

### 2.2.3 Code systématique

Un code est systématique, si le message $D(x)$ est directement contenu dans le mot de code $C(x)$. Les codes blocs peuvent être des codes systématiques.

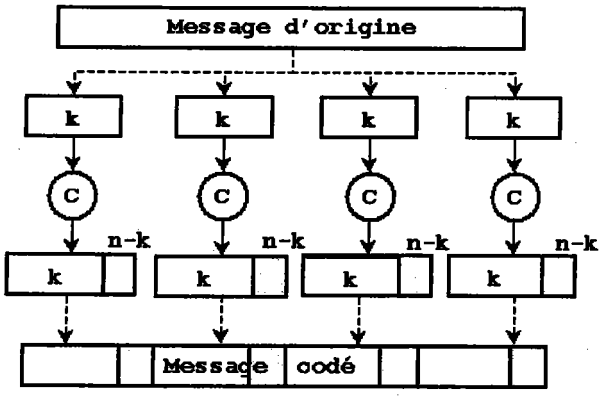



FIG. 2.7 – *Codage systématique.*

La matrice génératrice d'un code systématique s'écrit :

$$G = [P \mid I_k] \tag{2.9}$$

$I_k$ est la matrice identité de taille $k \times k$ et P est une matrice de taille $k \times (n-k)$.

**Codes de redondance cyclique (CRC)**

Le code de redondance cyclique est un code systématique. La procédure de codage consiste à décaler de $m$ positions (équ. (2.11)) le bloc d'information $D(x)$, ensuite de calculer le reste $\Re(x)$ de la division polynomiale de la séquence $x^m \cdot D(x)$ par un polynôme générateur $G(x)$ connu de l'émetteur et du récepteur. Enfin, le reste $\Re(x)$ de la division est concaténé au bloc d'information (équ.(2.13)) pour former la séquence codée $C(x)$ à transmettre.

$$D(x) = d_{k-1}x^{k-1} + d_{k-2}x^{k-2} + \cdots + d_1x^1 + d_0 \tag{2.10}$$

$$x^m \cdot D(x) = d_{k-1}x^{m+k-1} + d_{k-2}x^{m+k-2} + \cdots + d_0x^m + 0x^{m-1} + \cdots + 0 \tag{2.11}$$

$$\Re(x) = x^m \cdot D(x) \bmod G(x) \tag{2.12}$$

$$C(x) = x^m \cdot D(x) + \Re(x) \tag{2.13}$$

Au niveau du récepteur, la séquence reçue $\widetilde{C}(x)$ est divisée par $G(x)$. Si le reste de la division $\Re'(x)$ est nul, alors aucune erreur n'est détectée ; dans le cas contraire la séquence reçue est erronée (fig. 2.8). Pour la correction, les codes CRC sont en général combinés avec des stratégies ARQ.

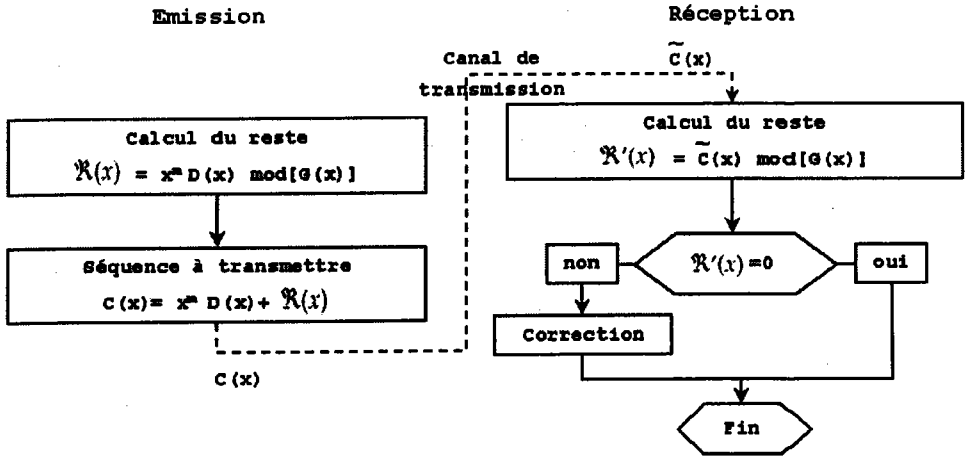



FIG. 2.8 – *Détection d'erreurs par CRC.*

A partir des deux circuits diviseurs vus précédemment (fig. 2.6 et 2.5) il est possible de réaliser un codeur cyclique systématique. Il suffit de relier l'entrée à la sortie pour générer les bits d'information (sortie systématique) et de réaliser les étapes nécessaires au bon fonctionnement du codeur. La figure 2.9 donne l'exemple d'un codeur CRC de polynôme générateur $G(x) = g_0 + g_1x + g_2x^2 + x^3$. L'entrée est décalée de $m$ bascules ce qui revient à faire la pré-multiplication par $x^m$.

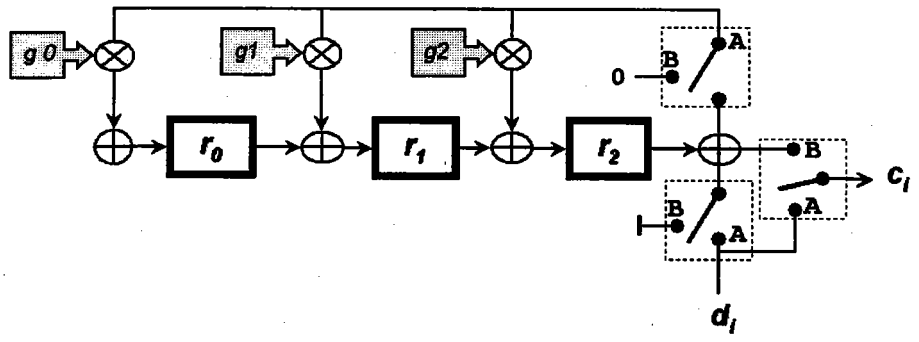



FIG. 2.9 – *Circuit codeur CRC.*

Le fonctionnement du codeur est simple et se déroule en deux phases :

- **Phase 1 :** les deux interrupteurs sont en position A durant $k$ cycles d'horloge. Simultanément, les bits d'information (systématiques) sont transmis à travers le canal de transmission et le codeur calcule le reste de la division. A la fin des $k$ cycles d'horloge la redondance est disponible.
- **Phase2 :** les deux interrupteurs sont en position B pendant $m$ cycles d'horloges. Des zéros sont injectés dans le codeur pour transmettre la redondance.

## 2.3 Codes convolutifs

Un code convolutif diffère d'un code bloc par le fait que chaque bloc de $n$ éléments en sortie ne dépend pas seulement des $k$ entrées à un instant donné mais aussi des $m$ blocs précédents (fig 2.10).

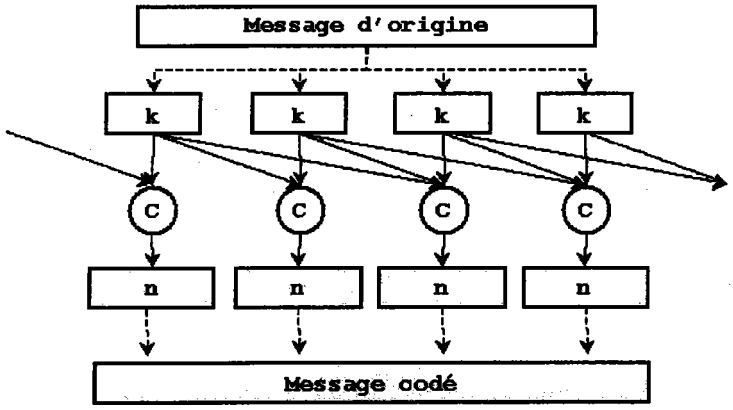



FIG. 2.10 – *Codage convolutif.*

Les codes convolutifs présentent des similitudes avec les codes cycliques. En effet, ils peuvent être systématiques et sont réalisables en utilisant les registres à décalage. Néanmoins, ils diffèrent de ces derniers dans le fait que les codes cycliques ont une longueur fixe à l'inverse des codes convolutifs qui eux ne sont pas de taille fixe. De plus, les codeurs cycliques sont des systèmes linéaires à une seule

entrée et une seule sortie totalement spécifiés par un seul polynôme générateur, alors que les codeurs convolutifs sont des systèmes à $k$ entrées et $n$ sorties spécifiés par $n \times k$ polynômes générateurs. La figure 2.11 donne l'exemple d'un codeur convolutif à 2 entrées et 4 sorties.

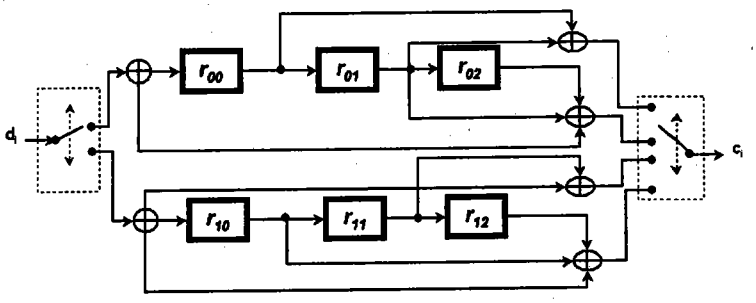



FIG. 2.11 – *Codeur convolutif à 2 entrées et 4 sorties.*

Le codeur contient $k$ registres à décalage qui correspondent chacun à une entrée. Si $m_i$ est la taille du registre à décalage associé à l'entrée $i$, la taille totale $m$ de la mémoire du codeur est :

$$m = \sum_{i=0}^{k-1} m_i \tag{2.14}$$

Le nombre maximum de bits associés à une sortie pouvant être affectés par un bit quelconque à l'entrée est appelé longueur de contrainte $K_c$ du codeur.

$$K_c = 1 + \max_i m_i \tag{2.15}$$

Chaque bit du mot de code est calculé suivant l'équation (2.16).

$$c_i^{(e,s)} = \sum_{j=0}^{m_e} d_{i-j}^{(e)} \cdot h_{sj}^{(e)} \tag{2.16}$$

où $c_i^{(e,s)}$ et $h_{sj}^{(e)}$ sont respectivement le mot de code et les coefficients du polynôme générateur associés à l'entrée $e$ et à la sortie $s$. $d_{i-j}^{(e)}$ est le bit de donnée à l'entrée $e$. Le mot de code $C$ est formé par multiplexage des sorties $C^{(s)}$.

## 2.3.1 Représentation des codes convolutifs

Un codeur convolutif[4] peut être vu comme un filtre à réponse impulsionnelle finie (FIR : *Finite Impulse Response*) possédant $k$ entrées et $n$ sorties. Son état est déterminé par le contenu de ses

4. Le terme « codeur convolutif » indique par défaut le cas non récursif sinon il sera mentionné explicitement que le codeur est récursif, auquel cas il est considéré comme un filtre à réponse impulsionnelle infinie IIR (*Infinite Impulse Response*).

registres à décalage. En pratique, un codeur convolutif peut être considéré comme une machine d'état et être représenté par un diagramme d'état, une structure en arbre ou une représentation en treillis.

**Diagramme d'état**

Le diagramme d'état (fig.2.12) est une représentation du fonctionnement du codeur ne faisant pas apparaître explicitement le temps.

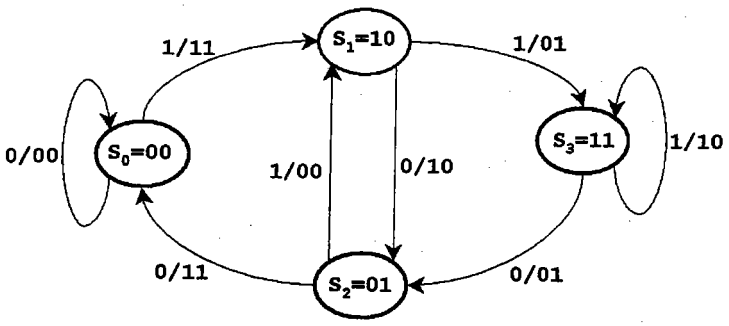



FIG. 2.12 – *Exemple d'un diagramme d'état* ($R = 1/2, m = 2$).

Il représente les transitions possibles entre les états. Les valeurs des sorties du codeur sont indiquées sur chacune des transitions. Tous les états internes possibles du codeur sont représentés par des nœuds $S_j$. Pour un codeur de rendement $1/n$ possédant une mémoire de taille $m$, il existe $2^m$ états internes possibles. Chaque nœud est connecté à un autre via une branche et le passage se fait par une transition $y/x_0x_1$, où $y$ correspond au bit d'entrée et $x_0x_1$ représente la séquence correspondante en sortie.

**Structure en arbre**

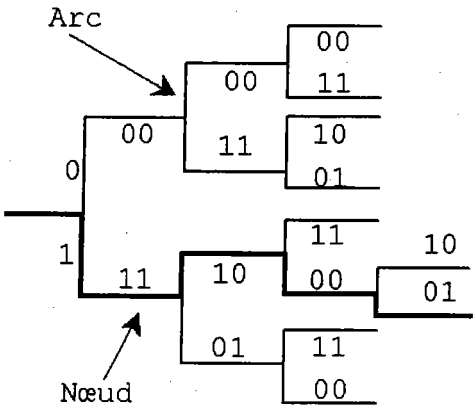



FIG. 2.13 – *Exemple d'une structure en arbre* ($R = 1/2, m = 2$).

Un arbre (fig. 2.13) est une structure partant d'un point appelé racine, il se compose d'arcs et de nœuds. Les arcs sont des traits verticaux dont le sens est déterminé par le bit d'information. Par convention, le 0 est représenté par un arc montant et le 1 par un arc descendant. Les nœuds sont des traits horizontaux indexés par les $n$ sorties correspondant au bit d'entrée. Le chemin en gras sur la figure 2.13 correspond au mot de code 11100001 généré par la séquence d'entrée 1011.

**Représentation en treillis.**

Contrairement aux deux précédentes, la représentation en treillis met en évidence le paramètre temporel. Chaque nœud $S_{(j,i)}$ correspond à un état particulier $S_j$ du codeur à un instant $i$, où $i$ représente l'indice du temps. Chaque branche est indexée par les bits qui se présentent en entrée et en sortie du codeur $e/s$. Chaque mot de code est associé à un chemin unique du treillis qu'on appelle « séquence d'état », dénotée par $\mathbf{s} = (s_0, \cdots, s_L)$. Un chemin complet commence à l'état $s_0 = S_{(0,0)}$ et se termine à l'état $s_L = S_{(0,L)}$.

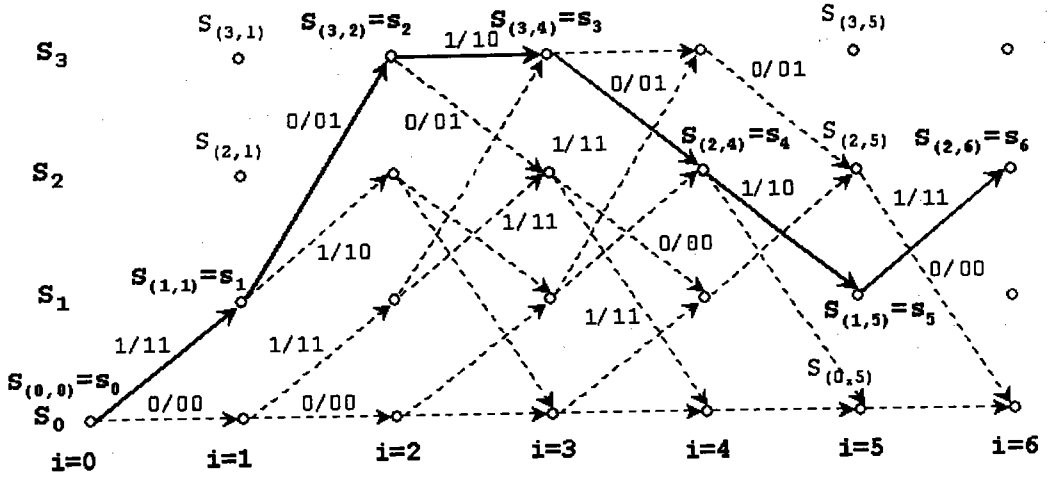



FIG. 2.14 – *Exemple d'une représentation en treillis* ($R = 1/2, m = 2$).

**Terminaison du treillis :** Un mot de code convolutif n'a pas de longueur fixe. Cependant pour des raisons pratiques dans la plupart des applications il est nécessaire d'imposer une longueur maximale fixe pour le mot de code. A cet effet, on ajoute à la séquence d'information $m$ bits appelés bits de queue et dont les valeurs sont nulles, ce qui à pour effet de forcer le treillis à revenir à l'état $S_{(0,L)}$. Dans ce cas, les codes convolutifs avec terminaison du treillis sont vus comme des codes blocs avec un rendement $R'$ :

$$R' = \frac{k \cdot (L - m)}{n \cdot L} < \frac{k}{n} \tag{2.17}$$

La réduction du taux de codage est définie par le paramètre $\xi$ :

$$\begin{aligned} \xi &= \frac{R - R'}{R} = \frac{n}{k} \cdot \left( \frac{k}{n} - \frac{k \cdot (L-m)}{n \cdot L} \right) \\ &= \frac{m}{L} \end{aligned} \tag{2.18}$$

$R$ est le rendement du code sans terminaison du treillis. L'équation (2.18) permet de voir que l'influence de la procédure de terminaison du treillis devient négligeable lorsque $L \gg m$.

### 2.3.2 Distance minimale d'un code convolutif

La distance minimale de liberté $d_{free}$ d'un code convolutif représente le plus petit poids de Hamming des mots de code associés aux séquences d'états débutant à l'état initial $S_{(0,0)}$. Elle est le premier critère de mesure des performances d'un code convolutif.

### 2.3.3 Période maximale du codeur

La période $T$ du codeur est la période observée à la sortie quand on applique à l'entrée une impulsion $(\cdots 0001)$. La période maximale $T_{max}$ d'un codeur convolutif de longueur de contrainte $K_c$ est :

$$T \leq T_{max} = 2^{(K_c - 1)} - 1 \tag{2.19}$$

### 2.3.4 Codeur convolutif de rendement 1/n

Dans le cas d'un codeur convolutif de rendement $1/n$, la taille $m$ de la mémoire devient celle du registre unique du codeur, la longueur de contrainte devient $K_c = 1 + m$ et chaque bit de parité se calcule suivant l'équation (2.20)[5].

$$c_i^{(s)} = \sum_{j=0}^{m} d_{i-j} \cdot h_{sj} \tag{2.20}$$

Le mot de code $C$ est formé en muliplexant les sorties $C^{(s)}$

$$C = \text{Fmux}[C^0, C^1, \cdots, C^{n-1}] = (c_0^{(0)} c_0^{(1)} \cdots c_0^{(n-1)}, c_1^{(0)} c_1^{(1)} \cdots c_1^{(n-1)}, \cdots) \tag{2.21}$$

où $C^i = (c_0, c_1, c_2, \cdots)^{(i)}$ avec $i \in \{0, 1, \cdots, n-1\}$.

La figure 2.15 représente un codeur convolutif de rendement $R = 1/2$ et de mémoire $m = 3$ pour les polynômes générateurs $H_0 = h_{00} + h_{01}x + h_{02}x^2 + h_{03}x^3$ et $H_1 = h_{10} + h_{11}x + h_{12}x^2 + h_{13}x^3$.

Le codeur doit être initialisé à zéro avant chaque opération de codage. Les bits d'informations sont ensuite propagés à travers les registres. En sortie, les bits de codage sont recueillis toujours dans

5. Le paramètre (e) dans l'équation (2.16) devient implicite puisque dans ce cas il n'y a qu'une seule entrée.

le même ordre (équ. (2.21)). A la fin on complète le message d'information par les $m$ bits de queue pour la terminaison du treillis.

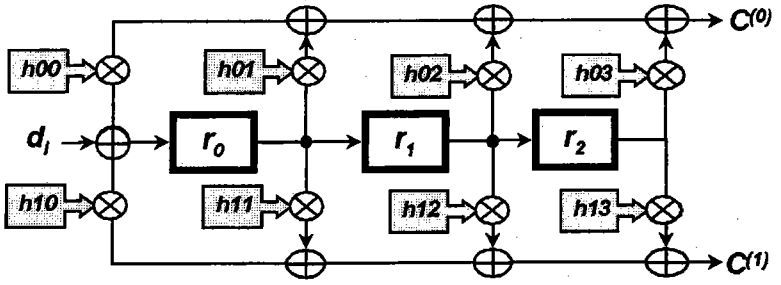



FIG. 2.15 – *Exemple d'un codeur convolutif* ($R = 1/2$, $m = 3$).

La structure d'un codeur convolutif de rendement $1/n$ est similaire à celle d'un multiplieur. En effet, le codeur convolutif se comporte comme $n$ multiplieurs dont chacun multiplie la même séquence par son propre polynôme générateur $H_i(x)$. Le chemin critique du système dépend du polynôme générateur qui possède le plus de coefficients non nuls.

### 2.3.5 Poinçonnage (ou perforation)

Les codes convolutifs à une seule entrée permettent d'atteindre au mieux des rendements de 1/2, ce qui s'avère insuffisant dans des applications où des taux plus élevés tels que 3/4 ou 2/3 sont requis. La méthode la plus directe pour augmenter le rendement consiste à utiliser des codeurs avec plusieurs entrées $k$. Cependant la complexité des circuits augmente avec le nombre d'entrées. Dans le cas notamment d'un décodeur de Viterbi, la complexité croit exponentiellement avec $k$. Une deuxième méthode consiste à utiliser un codeur à une seule entrée et de supprimer d'une manière systématique un certain nombre de bits à la sortie. Ainsi le nombre de bits supprimés détermine le nouveau rendement. Lors d'un décodage itératif la perforation uniquement des bits de parité permet de garantir une meilleure convergence de l'algorithme [3].

Dans le cas par exemple d'un codeur de rendement 1/2, la suppression en sortie de 1 bit sur 4 permet d'améliorer le rendement à 2/3. En supprimant 2 bits sur 6 on augmente le rendement à 3/4. A cet effet, on utilise une matrice de poinçonnage qui permet de masquer les bits qui ne seront pas transmis. En ce qui concerne le décodage, la matrice de perforation est essentielle pour déterminer avec précision la position de chaque bit supprimé. Cette technique permet d'avoir des systèmes fonctionnant à plusieurs rendements selon la matrice de poinçonnage appliquée. Dans [5] on donne un exemple d'une matrice $M_p$ qui est utilisée pour augmenter le rendement d'un code de 1/2 à 3/4.

$$M_p = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

– avant poinçonnage :

$$C = [c_0^{(0)}, c_0^{(1)}, c_1^{(0)}, c_1^{(1)}, c_2^{(0)}, c_2^{(1)}, c_3(0), c_3^{(1)}, c_4^{(0)}, \cdots] ;$$

– après le poinçonnage :

$$C = [c_0^{(0)}, c_0^{(1)}, c_1^{(0)}, -, c_2^{(0)}, c_2^{(1)}, c_3^{(0)}, -, c_4^{(0)}, \cdots].$$

### 2.3.6 Codes convolutifs récursifs et systématiques

Les codes convolutifs peuvent, comme les codes blocs, devenir systématiques en dédiant une sortie aux bits d'information. La récursivité est obtenue en appliquant une boucle de retour au codeur.

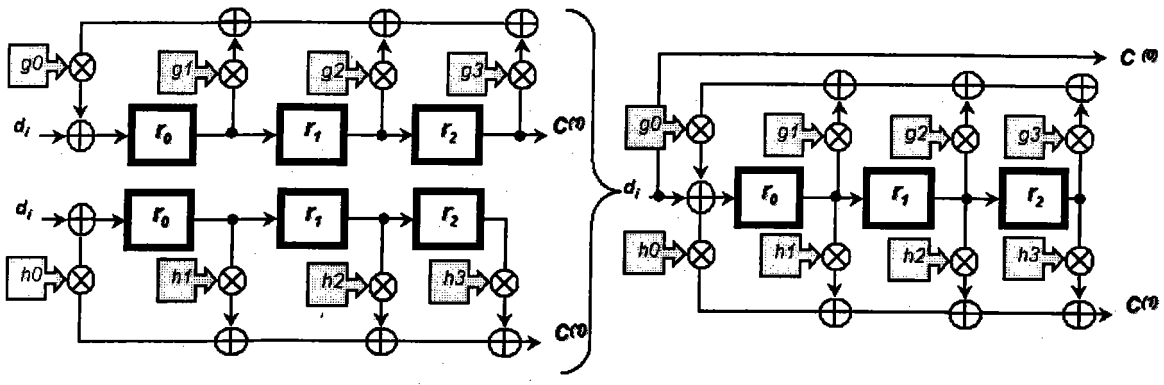



FIG. 2.16 – *Construction d'un codeur RSC* ($R = 1/2$, $m = 3$).

La figure 2.16 représente un codeur RSC de type MTO. De la même manière, en appliquant les conditions énumérées dans le paragraphe 2.2.2, on aboutit au codeur RSC de type OTM.

#### Terminaison du treillis des codes RSC

Comme pour le cas non récursif, le fonctionnement d'un codeur RSC peut être décrit par un diagramme d'état, un arbre ou une représentation en treillis. Pour un code de rendement $1/n$, la terminaison du treillis nécessite $T_t = K_c - 1$ cycles d'horloge (branches du treillis).

Alors que les codes convolutifs non récursifs peuvent être vus comme des filtres à réponse impulsionnelle finie (FIR), les codes RSC sont équivalents à des filtres à réponse impulsionnelle infinie (IIR). Dans ce cas, pour forcer le treillis à l'état initial zéro, il ne suffit plus de rajouter une séquence de $m$ bits nuls mais de les calculer de façon à mettre le registre d'état à zéro.

En considérant le codeur MTO de la figure 2.17 (interrupteur en position A) et résolvant l'équation

d'état de la variable $a_k$ [45] on trouve :

$$a_k = d_k + g_0 \cdot (g_1 \cdot a_{k-1} + g_2 \cdot a_{k-2} + g_3 \cdot a_{k-3}) \quad (2.22)$$

$$a_k = d_i + g'_1 \cdot a_{k-1} + g'_2 \cdot a_{k-2} + g'_3 \cdot a_{k-3} \quad (2.23)$$

Si on veut initialiser les registres à zéro il faut que $a_k$ soit mis à zéro :

$$0 = d_k + g'_1 \cdot a_{k-1} + g'_2 \cdot a_{k-2} + g'_3 \cdot a_{k-3} \quad (2.24)$$

On obtient ainsi l'entrée requise pour l'initialisation du codeur :

$$d_k = g'_1 \cdot a_{k-1} + g'_2 \cdot a_{k-2} + g'_3 \cdot a_{k-3} \quad (2.25)$$

Cette condition de fermeture du treillis est réalisable en commuttant l'interrupteur du codeur en position B [46] (fig. 2.17).

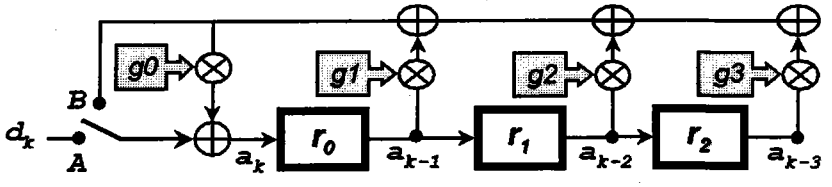



FIG. 2.17 – *Terminaison du treillis cas MTO.*

Dans le cas d'un codeur RSC MTO, il est nécessaire de rajouter une boucle de retour de coefficient $a_i$ pour la terminaison du treillis (interrupteur sur B). De même que dans le cas MTO, les coefficients sont calculés en résolvant l'équation d'état du codeur à un instant donné [40]. Dans le cas général d'un codeur OTM de rendement $k/n$, le nombre de cycles nécessaires à la terminaison du treillis est $T_t = \frac{K_c - 1}{k}$ [3, 40].

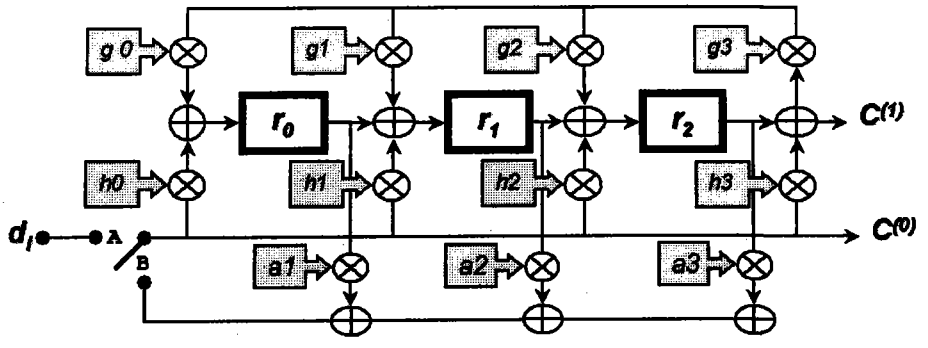



FIG. 2.18 – *Terminaison du treillis cas OTM.*

## 2.4 Turbo-codes

Un turbo-code convolutif[6] est la concaténation d'un ou de plusieurs codes convolutifs séparés par des blocs d'entrelacement. En pratique, il est considéré comme un code bloc (K, N) linéaire binaire[7] [67]. Les concaténations les plus connues sont la parallèle et la série. Les turbo-codes série sont plus complexes à mettre en œuvre cependant ils apportent de plus grand gain en distance minimale et en entrelacement. Les turbo-codes parallèle permettent d'approcher la limite de Shannon de manière efficace pour des rapports signal sur bruit très petits tandis que les turbo-codes série permettent de diminuer considérablement le taux d'erreurs pour des rapports signal sur bruit faibles à moyens [3].

### 2.4.1 Turbo-codes de type série

Un turbo-code convolutif série est obtenu par la concaténation série de codes convolutifs séparés par des entrelacements $\alpha_i$. Dans le cas d'un turbo-code à deux niveaux (fig. 2.19) le code en amont de l'entrelaceur est appelé code externe et celui en aval est le code interne. Pour un codage correct les deux codes constituants démarrent de l'état zéro $S_{(0,0)}$ et se terminent à l'état zéro $S_{(0,L)}$. En pratique, le code externe $C_1$ peut être un code RSC ou NRNSC. Quant au code interne $C_2$ il doit toujours être RSC.

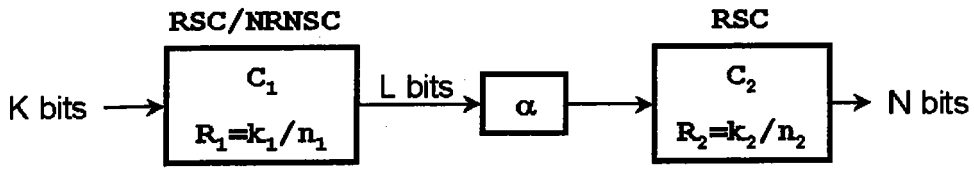



FIG. 2.19 – *Turbo-code série.*

En considérant que $R_1 = k_1/n_1$ et $R_2 = k_2/n_2$ sont les rendements respectifs du code externe $C_1$ et interne $C_2$, que $K$ bits d'informations entrent dans le codeur et que $N$ bits codés en sortent, le code $C_1$ génère $L = K/R_1$ bits et le code $C_2$ en génère $N = L/R_2$ .

En négligeant la fermeture du treillis le rendement total du codeur série est :

$$R = K/N = R_1 \cdot R_2 \tag{2.26}$$

$T_{t1}$ cycles sont nécessaires pour terminer le treillis de $C_1$ et $T_{t2}$ cycles pour $C_2$. Le nombre de bits additionnels en sortie est donc $n_1 T_{t1}$ pour $C_1$ et $n_2 T_{t2}$ pour $C_2$, dans ce cas le rendement est :

$$R = \frac{K}{J + n_1 T_{t1} + n_2 T_{t2}} = \frac{R_1 R_2}{1 + (n_1 T_{t1} + n_2 T_{t2}) \cdot (R_1 R_2 / K)} \tag{2.27}$$

6. Dans le cadre de cette étude nous nous intéressons uniquement à la concaténation de codes convolutifs.

7. Bien que, par définition, un mot de code convolutif n'ait pas de longueur fixe, pour des raisons pratiques sa longueur est limitée par une valeur maximale fixe. Dans ce cas il peut être assimilé à un code bloc linéaire binaire.

### 2.4.2 Turbo codes de type parallèle

Un turbo-code parallèle convolutif est construit par la concaténation parallèle de codes convolutifs séparés par des entrelacements. Chaque code constituant traite une version entrelacée différente de la même séquence d'information.

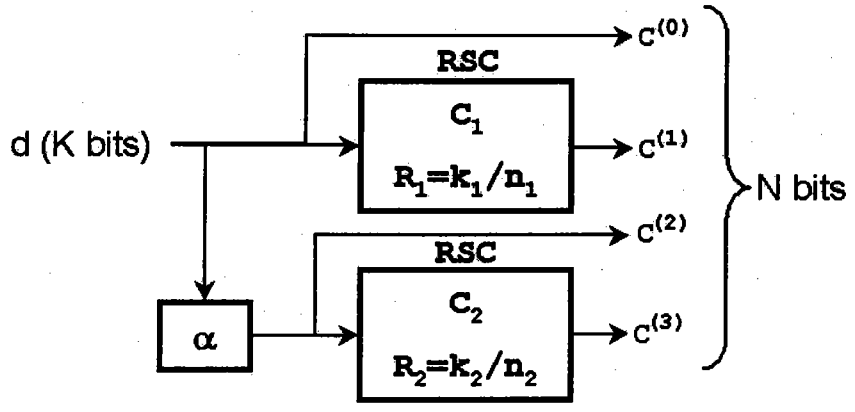



FIG. 2.20 – *Turbo-code parallèle.*

La figure 2.20 représente un turbo-code construit à partir de deux codes constituants $C_1$ et $C_2$. En pratique, on choisit les deux codes $C_1$ et $C_2$ similaires de type RSC. De plus, afin d'augmenter le rendement, il est possible de ne pas utiliser la sortie systématique $C^{(2)}$ [3]. En négligeant la fermeture du treillis, le rendement total du turbo-code parallèle est donné par les équations (2.28) et (2.29).

$$R1 \neq R_2 \Rightarrow R = \frac{K}{N} = \frac{R_1 \cdot R_2}{R_1 + R_2 - R_1 \cdot R_2} \tag{2.28}$$

$$R1 = R_2 \Rightarrow R = \frac{K}{N} = \frac{R_1}{2 - R_1} \tag{2.29}$$

$K$ est le nombre de bits en entrée et $N$ le nombre de bits codés en sortie. $R_1 = k_1/n_1$ et $R_2 = k_2/n_2$ sont les rendements respectifs de chaque code constituant $C_1$ et $C_2$.

En considérant la terminaison du treillis de chaque code constituant, $n_1 T_{t1}$ bits pour $C_1$ et $n_2 T_{t2}$ bits pour $C_2$ doivent être rajoutés, dans ce cas le rendement est :

$$R1 \neq R_2 \Rightarrow R = \frac{K}{N + n_1 T_{t1} + n_2 T_{t2}} = \frac{R_1 \cdot R_2}{R_1 + R_2 - R_1 \cdot R_2 + \frac{(n_1 T_{t1} + n_2 T_{t2})}{K}} \tag{2.30}$$

$$R1 = R_2 \Rightarrow R = \frac{K}{N + n_1 T_{t1} + n_2 T_{t2}} = \frac{R_1}{2 - R_1 + \frac{2 n_1 T_{t1}}{R_1 K}} \tag{2.31}$$

Les turbo-codes série peuvent atteindre des rendements plus élevés que les parallèles. Aussi, la distance minimale d'un turbo-codeur série est plus élevée que celle du parallèle car la concaténation série permet d'exploiter les séquences d'information, qui présentent au moins la distance minimale

$d_{min}(\mathcal{C}_1)$, issues du premier codeur. Par contre, la mise en œuvre est plus complexe du fait de la complexité de son treillis.

## 2.5 Performances d'un turbo-code

Dans cette partie nous nous intéressons uniquement aux performances des turbo-codes parallèles. Alors que les codes traditionnels ont pour but d'augmenter la distance de Hamming minimale $d_{min}$ d'un code, l'approche du turbo-codage est de limiter le nombre de mots de code de poids faible. De plus, comme les séquences d'information de poids 2 sont les plus susceptibles de générer les mots de code de poids faible [68], elles sont les plus importantes dans la conception des codes constituants. Plus leur poids augmente, plus leur importance dans la conception diminue. En effet, la combinaison d'un entrelacement avec un code de type RSC permet de maximiser le poids de Hamming des mots de code en sortie. Les codes NRNSC ne sont pas utilisables à cause de leur propriété faible, en terme de distance ($d_{min}$ faible), alors que l'aspect filtre à réponse impulsionnelle infinie d'un codeur RSC lui assure en général un grand poids de Hamming. La présence de l'entrelaceur permet de minimiser la probabilité que les deux codes constituants gérèrent en même temps un mot de code de poids faible. En effet, grâce à l'entrelacement, les deux codes constituants ne reçoivent pas la séquence d'information dans le même ordre. Ainsi, si le premier codeur reçoit une séquence susceptible de produire un mot de code de poids faible, il est fort peu probable que cette même séquence après l'entrelacement induit aussi un mot de code de poids faible à la sortie. Néanmoins, les performances d'un turbo-code dépendent fortement du choix des codes constituants ainsi que du type et de la taille des entrelaceurs utilisés.

### 2.5.1 Evaluation des performances par la borne de l'union

L'étude des performances d'un turbo-code passe par l'étude de ses codes constituants et de son entrelaceur, notamment de l'influence de ces derniers sur les probabilités d'erreur par mot $P_w$ et par bit $P_b$ du turbo-codeur. Les probabilités $P_w$ et $P_b$ peuvent être calculées par la méthode de « La borne de Gallager ». Cependant, malgré la précision dans l'estimation des probabilités $P_w$ et $P_b$, notamment dans le cas d'un canal AWGN, elle reste fastidieuse du fait qu'il n'est pas possible de calculer cette borne sans le balayage complet de tous les mots du code [3]. Par conséquent, cette approche est délaissée en faveur de la méthode de la borne de l'union tronquée, extrapolée vers des valeurs faibles de $E_b/N_0$, elle reste cependant valable si le rapport signal sur bruit est inférieur à la limite fixée par le débit de coupure (*cutoff rate*) $R_0$ [66, 7].

$$\left(\frac{E_b}{N_0}\right)_{dB} > 10 \cdot \log_{10}\left(-\frac{\ln(2^{(1-R_0)}-1)}{R_0}\right) \tag{2.32}$$

Comme vu précédemment (cf. paragraphe 2.4) un turbo-code de distance libre $d_{free}$ peut être assimilé à un code bloc $\mathtt{C}$ $(K,N)$[8] de rendement $K/N$ et de distance minimale $d_{min} = d_{free}$. En considérant par exemple un turbo-code parallèle à deux niveaux de rendement $1/3$, en prenant en compte les $2 \cdot m$ bits de la terminaison du treillis (m : mémoire du codeur), un turbo code peut être considéré comme un code bloc $(K,N = 3K + 2m)$. Dans ce cas, en considérant une transmission sur un canal BSC et un décodage par maximum de vraisemblance, les bornes de l'union sur la probabilité d'erreur par mots $P_w$ et par bits $P_b$ sont données respectivement par les équations (2.33) et (2.34) [67, 69] :

$$P_w \leq \sum_{w=1}^{K} W^w \cdot A^{\mathtt{C}}(w,Z) \tag{2.33}$$

$$P_b \leq \sum_{w=1}^{K} \frac{w}{K} \cdot W^w \cdot A^{\mathtt{C}}(w,Z) \tag{2.34}$$

où $A^{\mathtt{C}}(w,Z)$ est la fonction conditionnelle de distribution des poids des mots du code $\mathtt{C}$, pour un poids fixe d'une séquence d'information. Elle est calculée suivant l'équation (2.35) [70].

$$A^{\mathtt{C}}(w,Z) = \sum_{d=d_{min}}^{K} A^{\mathtt{C}}_{w,d} \cdot Z^d \tag{2.35}$$

où $A^{\mathtt{C}}_{w,d}$ est le nombre de mots de poids $d$ du code $\mathtt{C}$ générés par des séquences d'information en entrée de poids $w$. $Z$ et $W$ sont des variables dont la valeur dépend du type du canal de transmission. Dans le cas d'un canal AWGN, $Z = W = e^{-RE_b/N_0}$, alors que pour un canal de Rayleigh avec évanouissement, en considérant une estimation correcte des échantillons, $Z = W = 1/(1 + RE_b/N_0)$ [69].

## 2.5.2 Calcul du paramètre $A^{\mathtt{C}}_{w,d}$

La figure 2.21 représente un code concaténé à trois niveaux composé de quatre codes $\mathtt{C}_1, \mathtt{C}_2, \mathtt{C}_3, \mathtt{C}_4$ dont trois $\mathtt{C}_{i=2,3,4}$ sont précédés par un générateur de permutations $\alpha_{i=2,3,4}$ de taille $k_i$ [69].

Chaque code $\mathtt{C}_i$ est considéré comme un code bloc linéaire $(k_i, n_i)$ avec $A^{\mathtt{C}_i}_{w,d}$ les coefficients de poids correspondants. A noter que tous les entrelaceurs utilisés sont aléatoires et uniformes (cf. paragraphe 2.8.2).

Si $A^{\mathtt{C}_i}_{w,d}$ est le coefficient de poids d'un code $\mathtt{C}_i$ la probabilité qu'une séquence d'information de

8. $N$ inclut les bits dus à la terminaison du treillis de chaque code constituant. Néanmoins comme dans [3] il est possible de négliger les bits de terminaison du treillis en considérant des longueurs de blocs trés supérieurs à la taille $m$ de la mémoire du codeur.

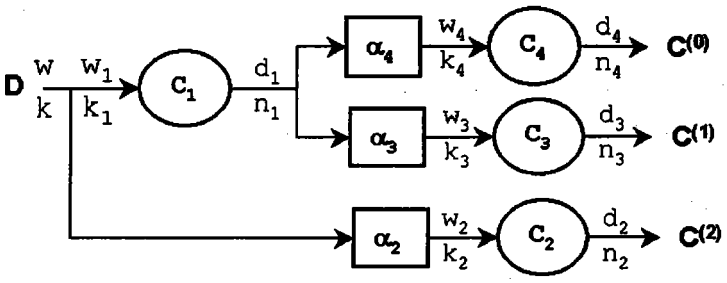



FIG. 2.21 – *Exemple d'un code concaténé à trois niveaux.*

poids $w$ génère après un entrelacement uniforme un mot de code de poids $d$ est [71] :

$$P^{\mathtt{C}_i}(w \to d) = \frac{A^{\mathtt{C}_i}_{w,d}}{C^k_w}, \quad \text{avec } C^k_w = \frac{k!}{w!(k-w)!} \tag{2.36}$$

Connaissant la taille des entrelaceurs et les coefficients $A^{\mathtt{C}_i}_{w,d}$ de chaque code $\mathtt{C}_i$, il est possible de calculer le facteur $A^{\mathtt{C}}_{w,d}$ moyen du codeur complet.

Soit l'ensemble $s_q = \{1,2,\cdots,q\}$ et les deux sous-ensembles $s_E \subset s_q$ et $s_S \subset s_q$ , tels que $s_E = \{i \in s_q : \mathtt{C}_i \text{ est connecté à l'entrée}\}$ et $s_S = \{i \in s_q : \mathtt{C}_i \text{ est connecté à la sortie}\}$. Le système de la figure 2.21 est considéré comme un code bloc $(k, n)$ avec $n = \sum_{i \in s_S} n_i$. Le coefficient moyen de poids $A^{\mathtt{C}}_{w,d}$ de l'ensemble est donné par l'équation (2.37) [69, 72].

$$A^{\mathtt{C}}_{w,d} = \sum_{(d_i \,:\, i \in s_S / \sum d_i = d)} \; \sum_{(d_i \,:\, i \in \overline{s_S})} A^{\mathtt{C}_1}_{w_1,d_1} \prod_{i=2}^{q} \frac{A^{\mathtt{C}_i}_{w_i,d_i}}{C^{k_i}_{w_i}} \tag{2.37}$$

$$\text{si } i \in s_E \;\Rightarrow\; d_i = d \tag{2.38}$$

$$\text{si } \mathtt{C}_i \text{ est précédé par } \mathtt{C}_j \;\Rightarrow\; w_i = d_j \tag{2.39}$$

où $\overline{s_S}$ est l'ensemble complémentaire de $s_S$. Le coefficient de poids équivalent $A^{\mathtt{C}}_{w,d}$ du code bloc $(k,n = n_2 + n_3 + n_4)$ du système concaténé est :

$$A^{\mathtt{C}}_{w,d} = \sum_{(d_1,d_2,d_3,d_4 / d_2+d_3+d_4=d)} A^{\mathtt{C}_1}_{w_1,d_1} \cdot \frac{A^{\mathtt{C}_2}_{w_2,d_2}}{C^{k_2}_{w_2}} \cdot \frac{A^{\mathtt{C}_3}_{w_3,d_3}}{C^{k_3}_{w_3}} \cdot \frac{A^{\mathtt{C}_4}_{w_4,d_4}}{C^{k_4}_{w_4}} \tag{2.40}$$

En appliquant les règles (2.38) et (2.39) dans l'équation (2.40) on obtient :

$$A^{\mathtt{C}}_{w,d} = \sum_{(d_1,d_2,d_3,d_4 / d_2+d_3+d_4=d)} A^{\mathtt{C}_1}_{w,d_1} \cdot \frac{A^{\mathtt{C}_2}_{w,d_2}}{C^{k}_{w}} \cdot \frac{A^{\mathtt{C}_3}_{d_1,d_3}}{C^{n_1}_{d_1}} \cdot \frac{A^{\mathtt{C}_4}_{d_1,d_4}}{C^{n_1}_{d_1}} \tag{2.41}$$

**Cas d'un turbo-code parallèle :**

Soit un turbo-code $\mathtt{C}_p$ de rendement $R = 1/3$ composé de deux codes constituants $\mathtt{C}_1$ et $\mathtt{C}_2$ de rendement $R_1 = R_2 = 1/2$. Les deux séquences de bits de parités sont émises sans perforation. Par déduction de l'équation (2.37) et (2.41) le coefficient $A_{w,d}^{\mathtt{C}_p}$ est :

$$A_{w,d}^{\mathtt{C}_p} = \sum_{(d_1,d_2/d_1+d_2=d)} \frac{A_{w,d_1}^{\mathtt{C}_1} \cdot A_{w,d_2}^{\mathtt{C}_2}}{C_w^k} \tag{2.42}$$

$$A_{w,d}^{\mathtt{C}_p} = \sum_{(d_1,d_1/2\cdot d_1=d)} \frac{(A_{w,d_1}^{\mathtt{C}})^2}{C_w^k} \quad (\text{ si } \mathtt{C}_1 = \mathtt{C}_2 = \mathtt{C}) \tag{2.43}$$

Dans ce cas, la distribution conditionnelle des poids du turbo-code $\mathtt{C}_p$ s'obtient en appliquant l'équation (2.35) [67, 3].

$$A^{\mathtt{C}_p}(w,Z) = \frac{\left[A^{\mathtt{C}}(w,Z)\right]^2}{C_w^k} \tag{2.44}$$

La fonction conditionnelle d'énumération des poids $A^{\mathtt{C}}(w,Z)$ liée au code constituant est calculée à partir de la matrice de transition des codes constituants [67, 69]. Cette matrice de transition est élevée à la puissance $k$ (taille de l'entrelaceur) [69]. La matrice de transition d'un code convolutif représente les transitions possibles entre deux états du codeur. Chaque transition est représentée par un 0 si elle n'est pas possible. Dans le cas contraire, elle est représentée par un monôme $L^l I^i D^d$ dont $l$ est toujours égal à 1, $i$ et $d$ correspondant aux poids associés à cette transition de l'entrée et de la sortie, respectivement.

La matrice de transition est directement déduite du diagramme d'état du codeur.

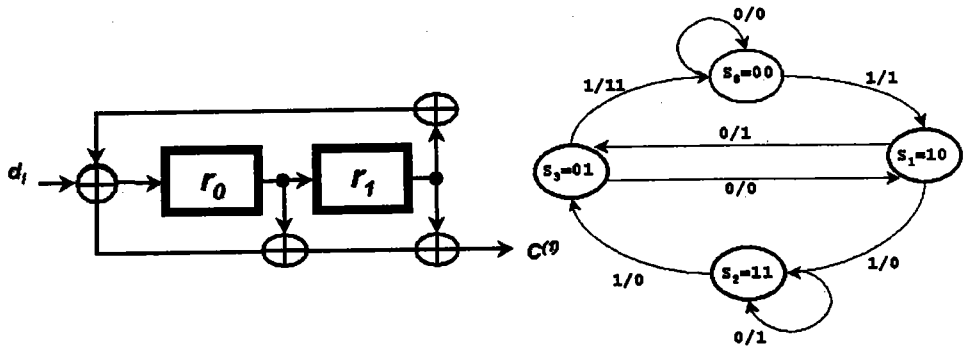



FIG. 2.22 – *Codeur RSC* $(7,5)_8$ *et son diagramme d'état.*

En prenant par exemple le code RSC de polynôme générateur $(H(x),\ G(x)) = (7,\ 5)_8$[9] ainsi que son diagramme d'état (fig. 2.22), la distribution des poids lors d'une transition est donnée par la

9. $()_8$ dénote la représentation des polynômes en octal, $(7)_8$ correspond au polynôme $H(x) = 1 + x + x^2$ et $(5)_8$ au polynôme $G(x) = 1 + x^2$.

matrice de transition $T_W$ [7] :

$$T_W = \begin{pmatrix} L & LID & 0 & 0 \\ 0 & 0 & LD & LI \\ LID & L & 0 & 0 \\ 0 & 0 & LI & LD \end{pmatrix}$$

Afin d'obtenir la distribution des poids pour $k$ transitions et en considérant que le codeur commence à l'état $S_{(0,0)}$ du treillis et se termine à l'état $S_{(0,L)}$, on calcule $T_W(S_{(0,0)} \rightarrow S_{(0,L)}) = T_W^k$ [66], ce qui détermine le spectre.

## 2.6 Rôle des codes constituants et de l'entrelaceur

Pour permettre une meilleure analyse de l'importance du choix des codes constituants et de l'entrelacement, une approximation de la borne de l'union sur la probabilité d'erreur par bit (équ. (2.34)), pour de grandes valeurs d'entrelacement $k = K$, est calculée par l'équation (2.45) [67, 3].

$$P_b \lessapprox \sum_{d=d_{min}}^{K} d \cdot \frac{d!}{n_{max}!^2} K^{2n_{max}-d-1} \cdot w^d \cdot [A^{\mathtt{C}}(w,Z,n_{max})]^2 \Big|_{\text{avec } (W=Z=e^{-R_c\frac{E_b}{N_0}})} \tag{2.45}$$

où $A^{\mathtt{C}}(w,Z,n_{max})$ est la fonction énumérateur d'événements [3]. Elle énumère tous les événements d'erreurs de $\mathtt{C}$ formés par la concaténation directe de $n_{max}$ événements simples. Elle est donnée par l'équation (2.46).

$$A^{\mathtt{C}}(d,Z,n_{max}) = \sum_j A^{\mathtt{C}}_{d,j,n} Z^j \tag{2.46}$$

où $A^{\mathtt{C}}_{d,j,n}$ est le nombre de mots de code formés de $n$ événements simples concaténés et correspondant à un poids total $d$ des bits d'information en entrée et $j$ des bits de parité en sortie [3].

Dans le cas d'un turbo-code dont les codes constituants sont NRNSC, la probabilité d'erreur par bit (équ. (2.34)) est donnée par l'équation (2.47) [67, 3].

$$P_b \lesssim \sum_{d=1}^{K} \frac{K^{d-1}}{(d-1)!} \cdot W^d \cdot [A^{\mathtt{C}}(1,Z,1)]^{2d} \Big|_{\text{avec } (W=Z=e^{-R_c\frac{E_b}{N_0}})} \tag{2.47}$$

Le cas le plus favorable dans l'équation (2.47) correspond à $d = 1$ [67, 3], auquel cas la probabilité d'erreur par bit $P_b$ est indépendante de la taille de la matrice d'entrelacement. Lorsque les codes constituants sont RSC, la probabilité d'erreur par bit $P_b$ est exprimée par l'équation (2.48) [67].

$$P_b \lesssim \sum_{i=1}^{K/2} 2i \cdot C_i^{2i} \cdot K^{-1} \cdot \frac{H^{(2+2d_2^p)^i}}{(1-H^{d_2^p-2})^{2i}} \Big|_{\text{avec } (H=e^{-R_c\frac{E_b}{N_0}})} \tag{2.48}$$

où $d_2^p$ est le poids minimal de la parité (sans prendre en considération le poids des bits systématiques) associé aux séquences d'information en entrée de poids 2. Dans ce cas, la distance de liberté $d_{free}$ [10] du turbo-code composé de deux codes constituants RSC identiques est définie par :

$$d_{free} = 2 + 2 \cdot d_2^p \tag{2.49}$$

L'équation (2.48) montre que le gain d'entrelacement varie en $1/K$. L'augmentation de la taille de $K$ permet de diminuer la probabilité d'erreur par bit.

## 2.7 Choix des codes constituants

Le choix des codes constituants de type RSC permet d'introduire un gain d'entrelacement qui varie en $1/K$ et de maximiser la distance de liberté $d_{free}^{\mathbf{C}}$ du turbo-code car les codes NRNSC présentent de faibles propriétés de distance par rapport aux codes RSC. En effet, dans le premier cas, la distance minimale $d_{min}$ vaut 1, alors que dans le cas RSC elle est toujours supérieure à 1. En particulier, pour des codes RSC de rendement $1/n$, $d_{min}$ est égal à 2 [73]. A noter que le choix des polynômes générateurs des codes RSC est aussi un critère non négligeable. L'optimisation des codes RSC est un facteur important pour des rapports signal sur bruit entre 1.5 et 2 dB [67]. Il a été démontré dans [68, 74] que les séquences d'information à faible poids ont plus de chance de générer des mots de code à faible poids. Par conséquent, la probabilité d'erreur par bit $P_b$ peut être approchée avec uniquement les premiers termes de la somme. Dans le cas de la concaténation de $q$ codes RSC la probabilité $P_b$ est donnée par l'équation (2.50) [68, 74] :

$$P_b \approx \frac{\beta}{K^{q-1}} \cdot Q\left(\sqrt{2r\frac{E_b}{N_0} \cdot d_{free}}\right) \tag{2.50}$$

où $\beta = K_{free} \cdot \overline{W}_{free}$ est une constante indépendante de la taille de l'entrelaceur, $K_{free}$ est le nombre de mots de code donnant à la distance de liberté et $\overline{W}_{free}$ représente le poids moyen des mots induisant la distance de liberté du code [66]. $1/K^{q-1}$ est le gain d'entrelacement du code. $Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^{+\infty} e^{-y/2} dy$.

Dans le cas général d'un code RSC de rendement $k/n$, la distance minimale $d_2^p$ à une borne supérieure $(d_2^p)_{max}$ :

$$(d_2^p)_{max} = \frac{(n-k) \cdot 2^{m-1}}{k} + 2(n-k) \tag{2.51}$$

Pour un code RSC de rendement $1/n$, la distance minimale $d_2^p$ est bornée par $(d_2^p)_{max} = (n-1) \cdot (2^{m-1} + 2)$. Le seul cas où la limite $(d_2^p)_{max}$ peut être atteinte correspond au choix d'un polynôme

10. Dans le cas de $q$ codes constituants différents $d_{free} = \sum_{i=1}^{q} (d_2^p)_i + 2$ , $(d_2^p)_i$ est le poids minimal des bits de parité à la sortie du code constituant $\mathbf{C}_i$ qui est généré par des séquences de poids 2 [40].

primitif pour la boucle de retour du codeur, auquel cas $d_2^p \leq (d_2^p)_{max}$. Quand le polynôme n'est pas primitif, la limite $(d_2^p)_{max}$ ne peut jamais être atteinte [40]. Le premier critère de choix d'un code RSC est qu'un polynôme primitif soit utilisé dans sa boucle récursive. Ensuite, de tous les polynômes directs qui permettent d'atteindre la limite $(d_2^p)_{max}$, le choix se portera sur celui qui a la meilleure distribution des poids. La première approche consiste à faire une étude spectrale sur les vecteurs de faible poids (entre 2 et 8) [66] : le choix se portera sur celui qui aura la plus grande distance de liberté. Dans le cas où deux codeurs ont la même distance de liberté, le codeur choisi est celui qui a le moins de vecteurs génèrant la distance libre. Une deuxième approche, plus précise, consiste à calculer la probabilité d'erreur par bit $P_b$ pour chaque code et de choisir celui qui offre une $P_b$ minimale pour la taille d'entrelacement et le rapport signal sur bruit voulus [67].

## 2.8 Matrices d'entrelacements

Le rôle de l'entrelaceur est d'éviter la génération simultanée par les codes constituants de séquences de parité de poids faible. Les vecteurs auto-terminants[11] de poids 2 sont les plus susceptibles de générer la distance libre du code [66, 68]. D'autre part, les codeurs RSC ont une réponse impulsionnelle infinie, ce qui veut dire que pour un vecteur de taille infinie et de poids 1 en entrée, le vecteur en sortie aura un poids qui tend vers l'infini. Le poids du mot de code augmente donc linéairement avec le nombre de zéros qui séparent les 1 du vecteur de poids 2. En effet, si les 1 sont éloignés les uns des autres, on peut assimiler une séquence de poids 2 à deux vecteurs « impulsion » qui génèrent à la sortie du codeur RSC des réponses impulsionnelles disjointes. Dans ce cas le poids augmente indéfiniment jusqu'à la terminaison du treillis [68]. Le rôle de la matrice d'entrelacement consiste à éloigner le plus possible les 1 dans les séquences d'information de poids 2. Cependant, il existe aussi des vecteurs auto-terminants de poids $n$, $n = 3,4,5\cdots$, qui génèrent des mots de code de distance faible. La distance entre les 1 de chaque vecteur auto-terminant doit aussi être prise en compte. Auquel cas, le choix de permutation optimale s'élargit à tous les vecteurs auto-terminants de poids $n$. Il est néanmoins plus probable qu'un vecteur auto-terminant de poids supérieur à 2 soit transformé en vecteur non auto-terminant en entrée des codes constituants [75]. En conclusion, si l'entrelaceur peut transformer tous les vecteurs auto-terminants en vecteurs non auto-terminants, le turbo-codeur aura un poids minimal qui augmentera linéairement avec la taille K de ce dernier [75].

La taille et le type d'entrelacement choisis sont deux facteurs liés aux performances d'un turbo-code. Le type d'entrelacement joue un rôle important seulement dans le cas de grands rapports signal sur bruit. Le paramètre le plus important dans le cas de faibles rapports s'avère être la taille de l'entrelacement [67]. D'un autre coté, plus le nombre des codes constituants et des entrelaceurs augmente plus les mauvaises séquences ont de chances d'être brisées.

11. Séquence d'information en entrée qui permet au codeur qui se situe avant l'entrelacement de revenir à l'état zéro.

### 2.8.1 Entrelacement périodique

L'entrelacement périodique engendre des permutations selon une fonction périodique du temps :

**Entrelacement par blocs :** Ce type d'entrelaceur est défini par une matrice rectangulaire de $l$ lignes et de $c$ colonnes de (taille $K = l \times c$). La matrice est remplie ligne par ligne. La lecture des données s'effectue ensuite colonne par colonne. Afin de réordonner le bloc entrelacé, on effectue l'opération inverse. Ce type d'entrelacement est efficace quand le vecteur pénalisant[12] se situe dans une seule colonne. Dans le cas où plusieurs vecteurs pénalisants sont confinés dans plusieurs colonnes consécutives, l'émission de chaque colonne doit se faire de manière à étaler le plus possible les vecteurs pénalisants [68]. Dans [76] par exemple, une méthode est proposée de ré-ordonnancement des colonnes en fonction des vecteurs pénalisants. Dans [77], une méthode est indiquée pour optimiser la génération des permutations basée sur l'utilisation d'une table de correspondance de poids pré-calculés.

**Entrelacement convolutif :** Chaque symbole de la séquence est aiguillé à travers un réseau de registres à décalage de différentes longueurs. La séquence entrelacée est ensuite formée par le multiplexage des sorties de chaque registre à décalage [26]. Si le nombre d'étages (registres à décalage) de l'entrelaceur est $K$ et si $K \cdot B$ représente le nombre de bascules additionnelles en passant de l'étage $i$ à l'étage $i+1$, alors la longueur $L_i$ du registre $i$ est $L_i = i \cdot K \cdot B$ avec $i \in \{0, \cdots, K-1\}$. La taille de l'entrelaceur ou son délai de remplissage est alors $D = (K-1) \cdot K \cdot B$ et sa fonction de permutation est donnée par la formule (2.52) [78]

$$\pi(j) = j + (j \bmod [K]) \cdot K \cdot B \tag{2.52}$$

L'utilisation d'un entrelaceur convolutif dans un turbo-code doit satisfaire la condition $gcd(K \cdot B, T) = 1$, en maintenant un bon compromis entre les valeurs de $K$ et $B$ avec que $K > B$ [78][13]. En comparaison avec un entrelacement par bloc, la surface utilisée par un entrelaceur convolutif peut être réduite d'environ 50% [79]. Les performances atteintes par un turbo-code utilisant un entrelaceur convolutif peuvent surpasser celles obtenues en employant un entrelacement par bloc. Pour un BER = $10^{-5}$ par exemple, un turbo-code utilisant un entrelaceur convolutif ($K = 14$, $B = 2$ et $D_c = 182$) apporte un amélioration de 0.2 dB du rapport signal sur bruit par rapport au même turbo-code utilisant un entrelacement par bloc, $l \times c = 12 \times 16$ et $D_b = 2 \cdot (12-1) \cdot (16-1) = 330$. En prenant le délai de l'entrelaceur convolutif égale à $D_c = 364 \approx D_b$ l'amélioration obtenue est de 1 dB [78]. Cependant, comme pour l'approche par bloc, l'entrelacement convolutif ne permet pas de casser tous les vecteurs pénalisants.

12. Vecteur induisant la distance de liberté du codeur.

13. Cette règle de conception a été déduite de l'étude d'un turbo-code de rendement $R = 1/2$ avec deux codes constituants RSC $(H, G)_8 = (33, 31)_8$.

**Entrelacement circulaire :** Ce type de permutation est une technique dérivée des décalages circulaires. La formule de base pour générer les permutations circulaires est donnée par :

$$\pi(j) = (a \cdot j + r) \bmod [K] \tag{2.53}$$

où $r$ est une constante de compensation et $a$ représente le pas d'itération : $a$ et $r$ sont toujours inférieurs à $K$. Le choix de la constante de compensation n'est pas important si les effets de bord sont négligeables [68]. Dans [68] on préconise de prendre la valeur $a = \sqrt{2K} - 1$, si $d_1$ représente la distance entre deux 1 d'une séquence de poids 2 avant permutation et $d_2$ celle de la séquence après permutations alors $d_1 + d_2 \geq \sqrt{2K}$. Toutefois cette valeur n'est pas unique, tout coefficient de la forme $a = l\sqrt{2K} \pm 1$ avec $l$ entier positif et $l < \sqrt{K/2}$, permet d'atteindre cette limite.

Dans le cas où $2K$ n'est pas un entier, on choisissant correctement le pas on obtient $d_1 + d_2 \geq \tau$ avec $\tau$ légèrement inférieur à $\sqrt{2K}$. La permutation circulaire est très efficace dans le traitement des vecteurs de poids 2. Par contre, elle amplifie les effets des séquences pénalisantes de poids supérieurs.

## 2.8.2 Entrelacement aléatoire

L'approche idéale pour générer des permutations est d'utiliser un entrelaceur aléatoire uniforme. Si la taille de l'entrelaceur est $k$, alors celui-ci associe à chaque séquence de poids $w$ en entrée une des $\alpha$ permutations possibles en sortie de l'entrelaceur avec une probabilité $P_w^k$ égale [72] :

$$\alpha = C_w^k \text{ et } P_w^k = \frac{1}{C_w^k} = \frac{1}{k!/\left(w! \cdot (k-w)!\right)}$$

Dans le cas d'un turbo-code à $q$ codes constituants et $q-1$ entrelaceurs aléatoires, la probabilité pour qu'une séquence de poids $n$ en entrée soit reproduite dans un bloc de $K$ bits est :

$$1 - \left[1 - \left(\frac{\beta}{K^{n-1}}\right)^{q-1}\right]^K \tag{2.54}$$

où $\beta$ est un paramètre qui dépend du poids $n$ et n'augmente pas en fonction de $K$. Pour des grandes valeurs de $K$ cette probabilité devient proportionnelle à $1/(K)^{nq-n-q}$ et décroît rapidement pour des valeurs de $n$ et $q$ supérieures à 2 [75]. Dans la pratique on essaye de concevoir des systèmes qui approchent le plus possible le comportement des entrelaceurs aléatoires uniformes. Citons deux exemples :

### Entrelaceur de Berrou-Glavieux

La taille $K$ de l'entrelaceur est choisie comme étant une puissance de 2 : $K = Q \cdot J = 2^i \cdot 2^j$. De plus, un tableau de nombres, premiers avec 8, est utilisé (voir tableau 2.1).

| $\vartheta$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| $p_\vartheta$ | 17 | 37 | 19 | 29 | 41 | 23 | 13 | 7 |

TAB. 2.1 – *Tableau de nombres premiers avec* 8.

Les permutations sont obtenues en appliquant la formule (2.55).

$$\pi(k) = c(k) + J \cdot r(k) \tag{2.55}$$

avec
$r(k) = (p_{\vartheta+1} \cdot (c_0 + 1) - 1) \bmod [Q]$,
$c(k) = ((\frac{Q}{2} + 1) \cdot (r_0 + c_0)) \bmod [J]$,
et $r_0 = k \bmod [J]$, $c_0 = \frac{(k-r_0)}{J}$, $\vartheta = (r_0 + c_0) \bmod [8]$.

**Entrelaceurs JPL**

Quels que soient J pair et Q, les permutations sont obtenues en appliquant la formule (2.56).

| $\vartheta$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| $p_\vartheta$ | 31 | 37 | 43 | 47 | 53 | 59 | 61 | 67 |

TAB. 2.2 – *Tableau de nombres premiers avec* 8.

$$\pi(k) = 2 \cdot r(k) \cdot Q \cdot c(k) - k \bmod [2] + 1 \tag{2.56}$$

avec
$r(k) = (19 \cdot r_0) \bmod [Q/2]$,
$c(k) = (p_{\vartheta+1} \cdot c_0 + 21 \cdot (k \bmod [2])) \bmod [J]$,
et $\left(r_0 = \frac{i-m}{2} - c_0\right) \bmod [J]$, $c_0 = (\frac{i-m}{2}) \bmod [J]$, $m = r_0 \bmod [8]$.

### 2.8.3 Entrelacement pseudo-aléatoire

L'entrelacement pseudo-aléatoire consiste à écrire la séquence aux adresses successives d'une mémoire vive, puis d'effectuer la lecture suivant un ordre donné par un générateur pseudo-aléatoire. L'opération inverse consiste à écrire dans la mémoire suivant le même ordre pseudo-aléatoire puis à lire la séquence selon l'ordre croissant des adresses de cette mémoire [26]. Dans [68] on propose une méthode pour générer des nombres $i$ compris entre 1 et $K$. Chaque nombre généré est comparé aux $S$ nombres précédents. Le nombre courant n'est pris que s'il est distant de $\pm S$ de n'importe lequel des $S$ précédents nombres. Le temps de calcul de cet algorithme augmente en fonction de $S$ et la convergence vers une solution n'est pas garantie. Cependant il a été observé dans [68] qu'en choisissant le paramètre $S$ inférieur à $\sqrt{2K}$, l'algorithme produit une solution en un temps raisonnable tout en garantissant une distance $d_1 + d_2 \geq S + 1$.

## 2.9 Terminaison du treillis d'un turbo-code

L'opération indispensable de fermeture du treillis d'un code convolutif en limite les performances. C'est d'autant plus vrai pour un turbo-code où il faut assurer la fermeture simultanée de deux codes. Deux « nouveaux » problèmes apparaissent dans ce cas :

- alors que les bits de fin utilisés pour terminer le treillis du premier codeur se situent en fin du message, ils sont vus comme des bits d'information par le deuxième codeur après être passés par l'entrelaceur ;
- l'entrelacement et la nature récursive du code RSC rendent difficile le calcul des bits de fin. En effet, les bits de fin ne peuvent être connus uniquement qu'à la fin du codage, ce qui complique le calcul de la séquence qui permettra de terminer les deux treillis en même temps.

Pour résoudre ces problèmes de fermeture du treillis, plusieurs solutions existent. L'une d'elles consiste à forcer la terminaison du treillis du premier codeur, tout en gardant le deuxième codeur ouvert [80]. Une deuxième solution consiste à calculer l'entrelaceur de manière à fermer le treillis des deux codes RSC avec la même séquence [81, 82]. Pour cela la conception du treillis doit satisfaire la condition

$$j \bmod [T] = \pi(j) \bmod [T] \tag{2.57}$$

où $T$ représente la période du codeur RSC (cf. paragraphe 2.3.3) et $\pi(j)$ représente la position du bit $j$ à la sortie de l'entrelaceur. Tous les types d'entrelaceurs pseudo-aléatoires satisfaisant cette condition peuvent être utilisés dans un turbo-codeur, sachant que certains sont plus performants que d'autres. Une troisième solution consiste à isoler les deux codes constituants pour qu'ils génèrent leurs séquences de terminaison du treillis indépendamment l'un de l'autre. Cette approche exploite la méthode décrite dans la section 2.3.6 qui permet de terminer le treillis d'un code RSC sans connaître au préalable l'état final du codeur [46, 40]. Une autre approche encore consiste à utiliser des « bits précurseurs » et de coder les données deux fois [83]. On concatène $2 \cdot m$ bits précurseurs au début de la séquence pour effectuer un premier codage. Ensuite grâce à une table de correspondance, les valeurs des bits précurseurs sont établies puis positionnées en fonction de l'état final du codeur. Ensuite une deuxième étape de codage est effectuée. Cette technique nécessite l'utilisation d'une table de correspondance entre les bits précurseurs et l'état final du codeur, et restreint le choix des entrelaceurs.

## 2.10 Perforation de la sortie d'un turbo-code

La perforation permet d'augmenter le rendement du turbo-codeur. Dans la pratique, il est préférable de ne perforer que les bits de parité afin de garantir une meilleure convergence lors du décodage. La figure 2.23 représente un turbo-code perforé de rendement $1/2$. La perforation s'effectue en connectant les deux sorties de parité à un multiplexeur qui sélectionne les bits à envoyer. Une technique consiste à envoyer les bits d'index pair du premier code constituant et ceux d'index impair du

second. Les bits systématiques sont transmis sans perforation. Si $n_1$ bits par bloc sont pris du premier codeur et $n_2$ bits par bloc du second, alors les rendements $R_1$ et $R_2$ respectivement des premier et deuxième codes constituants sont [2] :

$$R_1 = \frac{k}{2n_1 + n_2} \tag{2.58}$$

$$R_1 = \frac{k}{n_1 + 2n_2} \tag{2.59}$$

Dans le cas où $n_1 \neq n_2$, le turbo-code complet est vu comme étant la concaténation de deux codes constituants de rendements différents.

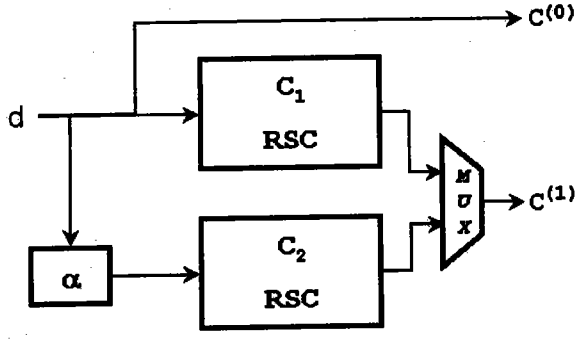



FIG. 2.23 – *Exemple d'un turbo-code perforé.*

Un exemple de turbo-codeur avec perforation est donné dans [84]. Le multiplexeur choisit trois bits du premier code constituant contre un seul du deuxième. Cette technique peut améliorer, sous certaines conditions, les performances de décodage [2].

Dans un turbo-code avec perforations les effets combinés de l'entrelacement et de la perforation peuvent dégrader les propriétés de distance. Dans [83], on suggère de concevoir l'entrelaceur de telle manière que les bits dont les positions ont un index pair (impair) avant l'entrelacement soient permutés sur des positions d'index pair (impair).

L'avantage de la perforation de la sortie est la possibilité de concevoir des systèmes de rendement variable. Une application directe est la conception de systèmes s'adaptant à l'importance des données et au niveau du bruit présent sur le canal. Dans le cas par exemple, d'un canal propre ou des données qui ne nécessitent pas de protection particulière, on transmet par perforation à un débit de $1/2$. Dans le cas contraire, on ralenti le système à $1/3$ [85].

## 2.11 Conclusion

Dans ce deuxième chapitre de la première partie, les codes convolutifs ont été introduits de manière générale. Les structures *Many To One* et *One To Many* de codeurs convolutifs ont été présentées.

Il a été montré qu'un codeur non récursif a la même architecture qu'un circuit multiplieur et que la version récursive utilisée dans les turbo-codes est la combinaison d'un diviseur et d'un multiplieur. Ensuite les différentes structures de turbo-codes ont été abordées. Comme nous nous intéressons plus particulièrement aux codes convolutifs, une étude approfondie sur les turbo-codes convolutifs a été effectuée incluant les stratégies de terminaison du treillis et méthodes de perforation. Cette étude a porté aussi sur l'influence de ces différents éléments (entrelaceurs et codes constituants) sur les performances de codage. Il en ressort que les différentes études [3, 40, 67, 68, 66, 73] effectuées sur le sujet convergent vers le choix d'un code convolutif de type RSC. Quant à l'entrelaceur, les deux paramètres à prendre en compte sont : sa taille pour de très faibles rapports signal sur bruit et son type pour des rapports signal sur bruit faibles à moyens. Dans le chapitre qui suit, on s'intéressera aux différents algorithmes de décodage itératif des turbo-codes.

# Chapitre 3

# Algorithmes de décodage itératif des turbo-codes

A l'image du circuit turbo-codeur, le circuit de décodage d'un turbo-code convolutif est constitué de décodeurs élémentaires de type SISO (*Soft-Input Soft-Output*) fonctionnant de manière itérative. Le décodage SISO s'effectue en adaptant les principaux algorithmes de décodages sur treillis des codes convolutifs, les algorithmes de Viterbi et MAP (Maximum A Posteriori), aux traitements des entrées souples fournies lors de la démodulation. La première partie de ce chapitre présente le schéma général d'un turbo-décodeur ainsi que le principe du décodage SISO. La deuxième partie est consacrée aux algorithmes Viterbi et MAP de décodage sur treillis des codes convolutifs ainsi qu'à leurs variantes SOVA (*Soft Output Viterbi Algorithm*), Log MAP et Max-Log MAP qui permettent la réalisation des décodeurs SISO utilisés dans le turbo-décodage. A noter que cette étude s'inspire fortement de [5, 66, 86]. La troisiéme partie est consacrée à l'étude architecturale d'un décodeur de Viterbi ainsi qu'à la présentation de quelques réalisations de circuits turbo-codeur existant sur le marché.

**Mots clés :** turbo-codes, algorithmes de décodage itératif, décodage SISO.

## 3.1 Introduction

L'opération de décodage consiste à retrouver une séquence d'états **s** d'un processus de Markov en présence de bruit. Les deux principales solutions sur treillis qui permettent la résolution de ce type de problème sont connues sous les noms d'algorithme de Viterbi et d'algorithme MAP. Le premier algorithme détermine la séquence d'états la plus probable à partir de la séquence reçue $y$ selon l'équation (3.1), tandis que le deuxième estime chaque état individuellement à partir de la séquence reçue $y$

d'après (3.2).

$$\widehat{\mathbf{s}} = \arg\max\{P(\mathbf{s}|y)\} \tag{3.1}$$

$$\widehat{s_i} = \arg\max\{P(s_i|y)\} \tag{3.2}$$

Une fois que la séquence d'états est évaluée, il devient aisé de déterminer le message où le mot de code à lui associer . En communication numérique, l'algorithme de Viterbi minimise la probabilité d'erreur par mot de code (FER : *Frame Error Rate*) tandis que MAP minimise la probabilité d'erreur par bit (ou symbole)(BER : *Bit Error Rate*).

## 3.2 Décodage SISO




FIG. 3.1 – *Décodeur SISO de rendement* $1/2$.

Un algorithme de type SISO traite des informations *a priori* et fournit à sa sortie des informations *a posteriori*. Les bits souples de décision sont généralement représentés par le logarithme du rapport de vraisemblance (LRV) (LLR : *Log-Likelihood Ratio*) donné par l'équation (3.3).

$$\Lambda_i = \ln\left(\frac{Pr\{d_i = 1 \mid y\}}{Pr\{d_i = 0 \mid y\}}\right) \tag{3.3}$$

En considérant une modulation BPSK la sortie modulée est exprimée suivant l'équation (3.5).

$$y' = a\sqrt{E_c \cdot (2c-1)} + n' \tag{3.4}$$

où $a$ représente une amplitude. Dans le cas d'un canal AWGN, $a$ est une constante, sinon on est en présence d'un canal à évanouissement de type Rayleigh ou Rice selon la distribution de $a$. Le terme $\sqrt{E_c \cdot (2c-1)}$ représente le symbole modulé, avec $E_c$ l'énergie par symbole modulé et $c \in \{0,1\}$. $n'$ est un bruit blanc gaussien de moyenne nulle et de variance $\sigma^2 = N_0/2$. L'équation (3.3) peut être exprimée de manière simplifiée par (3.5).

$$y = a(2c-1) + n \tag{3.5}$$

Dans ce cas, la variance s'exprime par $\sigma^2 = N_0/2E_c$ et le LRV par (3.6).

$$\Lambda_i = \frac{4a_i^{(s)}E_c}{N_0}y_i^{(s)} + z_i + l_i \tag{3.6}$$

où $y_i^{(s)}$ représente les bits systématiques reçus et $z_i$ l'information *a priori* qui provient du deuxième décodeur. $l_i$ est appelé information intrinsèque et représente à chaque itération une information dérivée des LRV des deux décodeurs et de l'entrée systématique.

$$l_i = \Lambda_i - \frac{4a_i^{(s)}E_c}{N_0}y_i^{(s)} - z_i \tag{3.7}$$

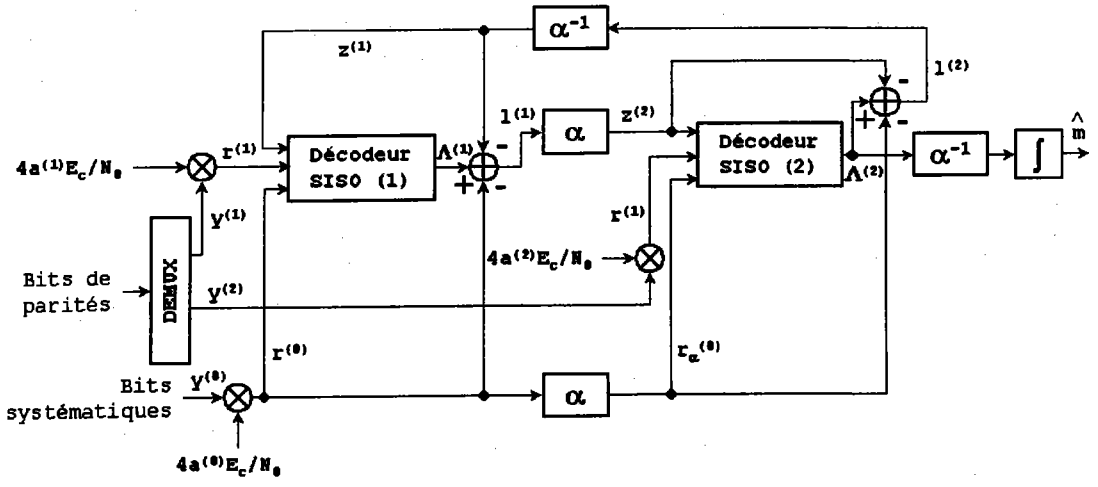



FIG. 3.2 – *Exemple d'un turbo-décodeur.*

Le turbo-décodeur fonctionne de la manière suivante :

1. Le premier décodeur SISO reçoit :
   - $r^{(0)}$ : les bits systématiques reçus $y^{(0)}$ multipliés par $\frac{4a_i^{(0)}E_c}{N_0}$ ;
   - $r^{(1)}$ : les bits de parités $y^{(1)}$ multipliés par $\frac{4a_i^{(1)}E_c}{N_0}$ ;
   - $z^{(1)}$ : l'information *a priori* dérivée du deuxième décodeur. Elle représente l'information intrinsèque $l^{(2)}$ après désentrelacement $l^{(2)}_{\alpha^{-1}}$. $l^{(2)}$ est calculé par la soustraction de l'information intrinsèque entrelacée $l^{(1)}_{\alpha}$ et $r^{(0)}_{\alpha}$ du premier décodeur à la sortie LRV $\Lambda^{(2)}$ du deuxième décodeur.
2. Le deuxième décodeur SISO reçoit :
   - $r^{(0)}_{\alpha}$ : les bits systématiques reçus $y^{(0)}$ multipliés par $\frac{4a_i^{(0)}E_c}{N_0}$ puis entrelacés ;
   - $r^{(2)}$ : les bits de parités $y^{(2)}$ multipliés par $\frac{4a_i^{(2)}E_c}{N_0}$ ;
   - $z^{(2)}$ : l'information a priori dérivée du premier décodeur. Elle représente l'information intrinsèque $l^{(1)}$ après entrelacement $l^{(1)}_{\alpha}$. $l^{(1)}$ est calculé par soustraction de l'information

intrinsèque désentrelacée $l_{\alpha^{-1}}^{(2)}$ et $r^{(0)}$ du deuxième décodeur à la sortie LRV $\Lambda^{(1)}$ du premier décodeur.

Après un certain nombre d'itérations qui déterminent la précision du décodage, le message estimé $\widehat{d}_i$ est obtenu par seuillage de la valeur LRV $\Lambda^{(2)}$ après désentrelacement $\Lambda_{\alpha^{-1}}^{(2)}$ :

$$\widehat{d}_i = \begin{cases} 1 & si \quad \Lambda_{\alpha^{-1}}^{(2)} \geq 0, \\ 0 & si \quad \Lambda_{\alpha^{-1}}^{(2)} < 0. \end{cases} \tag{3.8}$$

## 3.3 Algorithme de décodage SISO

Les algorithmes de décodage SISO sont dérivés des deux principaux algorithmes de décodage sur treillis, Viterbi et MAP. L'algorithme de Viterbi à sortie souple SOVA fournit en sortie une valeur de confiance des bits estimés. L'algorithme SOVA amélioré quant à lui, applique un facteur de correction à la valeur de confiance fournie, ce qui permet d'augmenter les performances de l'approche SOVA. L'algorithme MAP nécessite des calculs intensifs d'où la complexité de sa mise en œuvre. Les algorithmes qui en dérivent, Log Map et Max-Log Map [87], s'exécutent dans le domaine logarithmique d'où une complexité réduite par rapport au MAP originel. Le Log Map correspond à deux algorithmes de Viterbi « généraux » dont l'un parcourt le treillis en sens direct et l'autre en sens contraire. Le LRV est calculé en combinant les valeurs des métriques correspondantes dans les deux sens. Le Max-Log Map s'exécute de la même manière que le Log Map mais en simplifiant la formule de calcul des métriques.

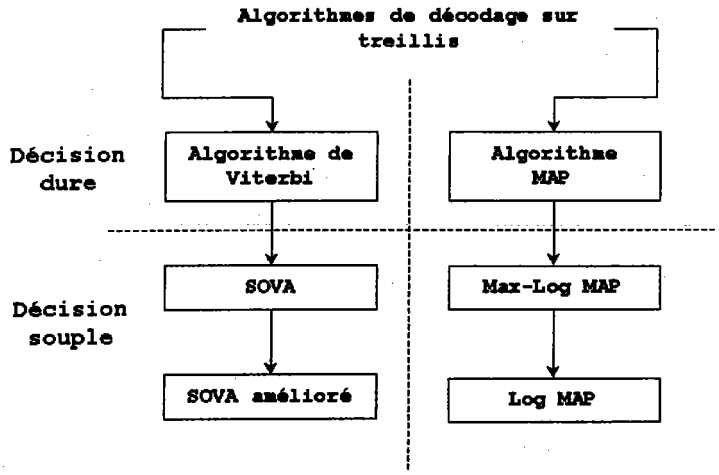



FIG. 3.3 – *Classification des algorithmes de décodage sur treillis.*

Du point de vue complexité, l'algorithme SOVA est le moins complexe à mettre en œuvre ; en outre il requiert moins de mémoire que les algorithmes SISO dérivées du MAP. La complexité du

Log map est double de celle du SOVA. Log MAP requiert l'utilisation d'une table de correspondance pour les facteurs de corrections. L'algorithme Max-Log Map présente une complexité intermédiaire. En effet, il est la version simplifiée du Log Map. Il ne prend pas en compte de facteur de correction et ne nécessite pas de table de correspondance. Du point de vue de la performance de décodage dans le cas d'un canal AWGN, l'algorithme Log Map apporte une amélioration du gain de 0.5 dB par rapport au Max-Log Map, qui surclasse à son tour l'algorithme SOVA de 0.5 dB. Quant au SOVA amélioré, il présente un gain de 0.3 à 0.5 dB par rapport à l'algorithme SOVA.

### 3.3.1 Algorithme de Viterbi

L'algorithme de Viterbi essaye de déterminer, à partir de symboles reçus, le chemin le plus probable à travers le treillis. Pour une estimation correcte, l'état initial ainsi que l'état final du codeur doivent être connus. En considérant la séquence d'états $\mathbf{s} = \{s_0, s_1, \cdots, s_K\}$ [1] l'algorithme de Viterbi détermine $\widehat{\mathbf{s}}$ suivant l'équation (3.1) :

$$\widehat{\mathbf{s}} = \arg\max_{\mathbf{s}}\{P(\mathbf{s}|\mathbf{y})\}.$$

où y représente l'ensemble des observations bruitées.

Comme les états du décodeur $\{s_0, s_1, \cdots, s_K\}$ sont considérés comme les états discrets d'un processus de Markov, le décodeur de Viterbi exploite deux propriétés des processus de Markov :

- $P(s_{i+1}|s_0, \cdots, s_i) = P(s_{i+1}|s_i)$ : la probabilité conditionnelle d'un état particulier connaissant tous les états précédents est égale à la probabilité conditionnelle du même état connaissant uniquement l'état qui le précède ;
- $P(y_i|\mathbf{s}) = P(y_i|s_i \rightarrow s_{i+1})$ : la probabilité conditionnelle d'une observation particulière $y_i$ connaissant la séquence d'états entière s est égale la probabilité conditionnelle de cette observation connaissant uniquement la transition de l'état $s_i$ vers l'état $s_{i+1}$.

L'équation (3.1) peut être réécrite en suivant le théorème de Bayes :

$$\widehat{\mathbf{s}} = \arg\max_{\mathbf{s}}\left\{\frac{P(\mathbf{y}|\mathbf{s})P(\mathbf{s})}{P(\mathbf{y})}\right\} \tag{3.9}$$

Comme le dénominateur $P(\mathbf{y})$ est commun à toutes les séquences évaluées, sa suppression n'affectera pas l'évaluation globale de $\widehat{\mathbf{s}}$. Une nouvelle valeur de $\widehat{\mathbf{s}}$ pourra être calculée suivant l'équation (3.10).

$$\widehat{\mathbf{s}} = \arg\max_{\mathbf{s}}\{P(\mathbf{y}|\mathbf{s})P(\mathbf{s})\} \tag{3.10}$$

L'équation (3.10) peut être résolue par la méthode directe en calculant la probabilité jointe de

1. K : représente la taille de l'entrelaceur du turbo-code.

chaque séquence **s** possible et de prendre ensuite le maximum. Cependant, la méthode directe nécessite un grand nombre d'opérations de calcul. Une deuxième approche exploite les deux propriétés liées aux processus de Markov citées précédemment. L'équation (3.10) devient :

$$\widehat{\mathbf{s}} = \arg\max_{s} \left\{ \prod_{i=0}^{K-1} P(y_i|s_i \rightarrow s_{i+1}) \prod_{i=0}^{K-1} P(s_{i+1}|s_i) \right\} \tag{3.11}$$

en appliquant le logarithme naturel à l'équation (3.11), on obtient :

$$\widehat{\mathbf{s}} = \arg\max_{s} \left\{ \sum_{i=0}^{K-1} (\ln P(y_i|s_i) \rightarrow s_{i+1}) + \ln P(s_{i+1}|s_i)) \right\} \tag{3.12}$$

$$\widehat{\mathbf{s}} = \arg\max_{s} \left\{ \sum_{i=0}^{K-1} \lambda(s_i \rightarrow s_{i+1}) \right\} \tag{3.13}$$

Le terme $\lambda(s_i \rightarrow s_{i+1}) = \ln P(y_i|s_i \rightarrow s_{i+1}) + \ln P(s_{i+1}|s_i)$ représente la métrique de la branche associée à la transition d'état $s_i \rightarrow s_{i+1}$. Il peut être exprimé en fonction des mots de code transmis $c_i = \left\{ c_i^{(0)}, c_i^{(1)}, \cdots, c_i^{(n-1)} \right\}$ et des messages $d_i$ correspondants qui sont associés à la transition d'état $s_i \rightarrow s_{i+1}$.

$$\lambda(s_i \rightarrow s_{i+1}) = \ln P(y_i|c_i) + \ln P(d_i) \tag{3.14}$$

Dans le cas de turbo-décodage, le terme $\ln P(d_i)$ est calculé à partir de l'information *a priori* $z_i$ fournie par le second décodeur élémentaire selon l'équation (3.15).

$$P(d_i) = \begin{cases} \frac{e^{z_i}}{1+e^{z_i}} & \leftarrow \quad d_i = 1 \\ \frac{1}{1+e^{z_i}} & \leftarrow \quad d_i = 0 \end{cases} \tag{3.15}$$

$$\ln P(d_i) = z_i d_i - \ln(1 + e^{z_i}) \tag{3.16}$$

En considérant une modulation BPSK (équ. (3.5)) dans le cas d'un canal à évanouissement uniforme (*flat-fading channel*), la métrique de la branche est donnée par :

$$\begin{aligned} \lambda(s_i \rightarrow s_{i+1}) &= \ln P(d_i) - \frac{1}{2} \ln \left( \frac{\pi N_0}{E_c} \right) - \frac{E_c}{N_0} \sum_{q=0}^{n-1} \left( y_i^{(q)} - a_i^{(q)} (2c_i^{(q)} - 1) \right)^2 \\ &= z_i d_i - \left[ \ln(1 + e^{z_i}) + \frac{1}{2} \ln \left( \frac{\pi N_0}{E_c} \right) \right] - \frac{E_c}{N_0} \sum_{q=0}^{n-1} \left( y_i^{(q)} - a_i^{(q)} (2c_i^{(q)} - 1) \right)^2 \\ &= z_i d_i + \eta - \frac{E_c}{N_0} \sum_{q=0}^{n-1} \left( y_i^{(q)} - a_i^{(q)} (2c_i^{(q)} - 1) \right)^2 \end{aligned} \tag{3.17}$$

où $\eta$ est le même pour toutes les hypothèses. Le calcul de la métrique est une combinaison de l'information intrinsèque générée par le deuxième décodeur. Elle représente le carré de la distance euclidienne entre l'observation à la sortie du canal et le symbole reçu. Le niveau du signal sur bruit détermine le degré de confiance qu'on porte au signal reçu ou à l'information intrinsèque fournie par le deuxième décodeur. L'équation (3.17) peut être simplifiée en développant le terme au carré et en regroupant en un seul terme, les termes qui sont les mêmes pour toutes les hypothèses :

$$\begin{aligned}\lambda(s_i \rightarrow s_{i+1}) &= \eta + z_i d_i - \frac{E_c}{N_0}\sum_{q=0}^{n-1}\left[(y_i^{(q)})^2 - 2a_i^{(q)}y_i^{(q)}(2c_i^{(q)}-1) + (a_i^{(q)})^2(2c_i^{(q)}-1)^2\right] \\ \lambda(s_i \rightarrow s_{i+1}) &= \eta' + z_i d_i + \frac{E_c}{N_0}\sum_{q=0}^{n-1} 4a_i^{(q)}y_i^{(q)}c_i^{(q)} \qquad (3.18)\end{aligned}$$

avec

$$\begin{aligned}\eta' &= \eta - \frac{E_c}{N_0}\sum_{q=0}^{n-1}\left[(y_i^{(q)})^2 - 2a_i^{(q)}y_i^{(q)} + (a_i^{(q)})^2(2c_i^{(q)}-1)^2\right] \qquad (3.19) \\ &= \eta - \frac{E_c}{N_0}\sum_{q=0}^{n-1}\left[(y_i^{(q)})^2 - 2a_i^{(q)}y_i^{(q)} + (a_i^{(q)})^2\right] \qquad (3.20)\end{aligned}$$

L'équation (3.20) est obtenue en remplaçant le terme $(2c_i^{(q)}-1)^2$ par la valeur 1, ce qui est toujours vrai dans le cas où $c_i^{(q)} \in \{0,1\}$. Comme $\eta'$ est un terme commun à toutes les expressions des métriques, sa suppression n'aura pas d'influence lors de l'évaluation de la métrique la plus grande. La nouvelle forme « compacte » de l'expression qui calcule la métrique d'une branche est donnée par l'équation suivante :

$$\lambda(s_i \rightarrow s_{i+1}) = z_i d_i + \sum_{q=0}^{n-1} r_i^{(q)} c_i^{(q)} \qquad (3.21)$$

avec $r_i^{(q)} = (4a_i^{(q)}E_c/N_0)y_i^{(q)}$. Dans le cas d'un code systématique, $d_i = c_i^{(0)}$ et le calcul de la métrique d'une branche s'effectue suivant l'équation (3.22).

$$\lambda(s_i \rightarrow s_{i+1}) = \left(z_i + r_i^{(0)}\right)c_i^{(0)} + \sum_{q=0}^{n-1} r_i^{(q)} c_i^{(q)} \qquad (3.22)$$

L'algorithme de Viterbi se déroule de la manière suivante :

1. Créer un tableau $\Gamma(j,i)$ avec $j \in \{0,\cdots,2^m-1\}$ et $i \in \{0,\cdots,K\}$, où $m$ représente la mémoire du codeur et $K$ la longueur de la séquence. Ce tableau contient les valeurs des métriques

cumulées correspondant au nœud $S_{(j,i)}$ du treillis. Il est initialisé comme suit :

$$\Gamma(j,0) = \begin{cases} 0 & \leftarrow j = 0, \\ -\infty & \leftarrow j \neq 0. \end{cases} \tag{3.23}$$

2. Créer un tableau $\mathbf{S}(j,i)$ avec $j \in \{0,\cdots,2^m-1\}$ et $i \in \{0,\cdots,K\}$. Ce tableau contient la séquence d'états $(s_0,\cdots,s_K)$ associée au chemin survivant jusqu'à l'état $S_{(j,i)}$. Il est initialisé comme suit :

$$\mathbf{S}(0,0) = S_0 \tag{3.24}$$

où $S_0$ est l'état initial du codeur.

3. Effectuer $i = 1$.
4. Effectuer $j = 0$.
5. Effectuer $s_i = S_j$.
6. Mettre à jour le tableau des métriques cumulées selon :

$$\Gamma(j,i) = \max_{s_{i-1}=S'_j \in \Omega} \{\Gamma(j',i-1) + \lambda(s_i \rightarrow s_{i+1})\} \tag{3.25}$$

où $\Omega$ représente l'ensemble des états $s_{i-1}$ connectés à l'état $s_i$.

7. Sauvegarder le chemin survivant en appliquant :

$$\mathbf{S}(j,i) = (\mathbf{S}(j',i-1), s_i) \tag{3.26}$$

où $j'$ correspond à l'index de l'état $S_{j'}$ qui est l'argument de la maximisation de l'équation (3.25).

8. Effectuer j=j+1.
9. Si $j = 2^m$ aller à l'étape 10, sinon revenir à l'étape 5.
10. Effectuer i=i+1.
11. Si i=K+1 aller à l'étape 12, sinon revenir à l'étape 4.
12. Si le treillis est terminé, le chemin le plus probable est stocké dans $\mathbf{S}(j',K)$ avec $j' = 0$, sinon $j' = \arg\max_j \{\Gamma(j,K)\}$.
13. Le message $\widetilde{d}_i$ et les mots de code $\widetilde{c}_i$ peuvent être directement déterminés à partir de la séquence d'états $\widehat{s}$.

### 3.3.2 Algorithme de Viterbi à sortie souple SOVA

L'algorithme de Viterbi à sortie souple s'exécute en deux phases. La première consiste à exécuter l'algorithme de Viterbi standard avec quelques modifications suivi d'une étape de calcul des valeurs

de confiance associées aux bits estimés. Les modifications apportées à la première étape sont[2] :

**1.bis** Créer un tableau $\Delta(j,i)$, avec $j \in \{0,\cdots,2^m-1\}$ et $i \in \{0,\cdots,K\}$. Ce tableau contient la différence entre les métriques du chemin survivant et de son compétiteur.

**2.bis** Créer un tableau $\mathbf{C}(j,i)$, avec $j \in \{0,\cdots,2^m-1\}$ et $i \in \{0,\cdots,K\}$. Ce tableau contient la séquence d'états $(s_0^*,\cdots,s_i^*)$ du chemin compétiteur.

**6.bis** Mettre à jour le tableau $\Delta(j,i)$ selon :

$$\begin{aligned}\Delta(j,i) &= \max_{s_{i-1}=S_j'\in\Omega}\{\Gamma(j',i-1)+\lambda(s_i\rightarrow s_{i+1})\} \\ &- \min_{s_{i-1}=S_j''\in\Omega}\{\Gamma(j'',i-1)+\lambda(s_i\rightarrow s_{i+1})\}\end{aligned} \tag{3.27}$$

**7.bis** Sauvegarder la séquence d'états du chemin compétiteur suivant :

$$\mathbf{C}(j,i) = (\mathbf{S}(j'',i-1),s_i) \tag{3.28}$$

$j''$ correspond à l'index de l'état $S_{j''}$ qui est le paramètre de la minimisation de l'équation (3.27).

La deuxième étape consiste, pour chaque bit estimé, à calculer les valeurs de confiance associées.

1. Créer un vecteur de confiance $\rho = (\rho_0,\cdots,\rho_{K-1})$, avec $\rho_i = \infty\ \forall$ i.
2. Effectuer $i = K$.
3. Effectuer $S_j = \hat{s}_i$, $\hat{s} = (\hat{s}_0,\cdots,\hat{s}_K)$ étant la séquence d'états calculée par l'algorithme de Viterbi lors de la première phase.
4. Soit $\hat{s^*} = \mathbf{C}(j,i)$=$(\hat{s^*}_0,\cdots,\hat{s^*}_i)$ la séquence d'états associée au chemin compétiteur sélectionnée au nœud $S_{j,i}$.
5. Soit $\hat{d} = (\hat{d}_0,\cdots,\hat{d}_i)$ le message estimé par l'algorithme de Viterbi lors de la première phase et $\tilde{d} = (\tilde{d}_0,\cdots,\tilde{d}_i)$ le message associé au chemin compétiteur sélectionné au nœud $S_{j,i}$.
6. Effectuer $k = 1$.
7. Si $\tilde{d}_{i-k} \neq \hat{d}_{i-k}$, alors $\rho_{i-k} = \min\{\rho_{i-k},\Delta(j,i)\}$.
8. Effectuer $k = k+1$.
9. Si $\hat{s^*}_{i-k} = \hat{s}_{i-k}$ aller à l'étape 10, sinon revenir à l'étape 7.
10. Effectuer $i = i-1$.
11. Si $i > 0$ revenir à l'étape 3, sinon aller à l'étape 12.
12. Calculer les LRV suivant l'équation (3.29).

$$\Lambda_i = (2\hat{d}_i - 1)\rho_i,\ i \in \{0,\cdots,K\} \tag{3.29}$$

2. Dans le cas de codes de rendement $1/n$, deux branches sont connectées à chaque nœud du treillis.

### 3.3.3 Algorithme SOVA amélioré

Il a été démontré dans [88] que les sorties estimées $\Lambda_i$ fournies par l'algorithme SOVA sont biaisées (*over-optimistic*). En effet l'algorithme SOVA doit fournir un LRV de la forme :

$$L_i = \frac{P(d_i = 1|\Lambda_i)}{P(d_i = 0|\Lambda_i)} = \frac{2d_\Lambda}{\sigma_\Lambda^2}\Lambda_i \tag{3.30}$$

Le SOVA amélioré corrige les estimations biaisées du SOVA en les multipliant par un facteur de correction $F_c$.

$$F_c = \frac{2d_\Lambda}{\sigma_\Lambda^2} \tag{3.31}$$

Dans ce cas la moyenne et la variance de la sortie SOVA doivent être estimées au préalable ce qui a pour effet d'augmenter la complexité du circuit. Néanmoins, la correction des sorties SOVA apporte une amélioration de 0.3 à 0.5 dB par rapport au SOVA sans correction.

### 3.3.4 Algorithme MAP

L'algorithme MAP (Maximum A Posteriori) a été proposé pour la première fois en 1974 par Bahl et al. [89]. Deux versions de cet algorithme existent, la version appelée type-I parcourt le treillis dans les sens direct et inverse, elle est orientée vers le traitement en bloc des données. Quant à la version type-II, elle parcourt le treillis uniquement en sens direct et est orientée vers le traitement continu des données. Elle est plus complexe à mettre en œuvre et nécessite plus de mémoires que la type-I. Comme les turbo-codes sont assimilés à des codes blocs, l'algorithme MAP de type-I est utilisé de préférence [5]. Dans le processus de décodage, l'algorithme MAP calcule les probabilités de justesse *a posteriori* (APP : *A Posteriori Probability*) des bits du message reçu. $P(d_i = 1|\mathbf{y})$ et $P(d_i = 0|\mathbf{y})$ sont ensuite mises sous la forme de rapports de vraisemblance logarithmique (LRV). En utilisant la définition de la probabilité conditionnelle (équ. (3.32)) et en appliquant les propriétés du processus de Markov pour partitionner le numérateur on obtient l'équation (3.33).

$$P(s_i \rightarrow s_{i+1}|\mathbf{y}) = \frac{P(s_i \rightarrow s_{i+1},\mathbf{y})}{P(\mathbf{y})} \tag{3.32}$$

$$P(s_i \rightarrow s_{i+1},\mathbf{y}) = \alpha(s_i)\gamma(s_i \rightarrow s_{i+1})\beta(s_{i+1}) \tag{3.33}$$

avec

$$\alpha(s_i) = P(s_i,(y_0,\cdots,y_{i-1})) \tag{3.34}$$

$$\gamma(s_i \rightarrow s_{i+1}) = P(s_{i+1},y_i|s_i) \tag{3.35}$$

$$\beta(s_{i+1}) = P((y_{i+1},\cdots,y_{K-1})|s_{i+1}) \tag{3.36}$$

où $\gamma(s_i \rightarrow s_{i+1})$ représente la métrique de la branche qui est associée avec la transition d'état $(s_i \rightarrow s_{i+1})$.

$$\begin{aligned}\gamma(s_i \rightarrow s_{i+1}) &= P(s_{i+1}|s_i)P(y_i|s_i \rightarrow s_{i+1}) \\ &= P(d_i)P(y_i|c_i)\end{aligned} \tag{3.37}$$

$d_i$ et $c_i$ sont le message et son mot de code associés à la transition d'état $s_i \rightarrow s_{i+1}$. Le premier terme de l'équation (3.37) est calculé en utilisant l'information à priori $z_i$ du deuxième codeur et l'équation (3.15). Le second terme dépend du type de modulation et du modèle du canal de transmission. Dans le cas d'un canal à évanouissement uniforme et d'une modulation BPSK, $P(y_i|c_i)$ est donné par l'équation (3.38).

$$P(y_i|c_i) = \frac{1}{\sqrt{\pi \cdot N_0/E_c}} \exp\left\{-\frac{E_c}{N_0}\sum_{q=0}^{n-1}\left(y_i^{(q)} - a_i^{(q)}(2c_i^{(q)} - 1)\right)^2\right\} \tag{3.38}$$

La probabilité $\alpha(s_i)$ est calculée en parcourant le treillis en sens direct selon l'équation :

$$\alpha(s_i) = \sum_{s_{i-1}\in\Omega_f} \alpha(s_{i-1})\gamma(s_{i-1} \rightarrow s_i) \tag{3.39}$$

où $\Omega_f$ représente l'ensemble des états $s_{i-1}$ connectés à l'état $s_i$.

La probabilité $\beta(s_i)$ est calculée en parcourant le treillis en sens inverse d'après l'équation :

$$\beta(s_i) = \sum_{s_{i-1}\in\Omega_b} \beta(s_{i+1})\gamma(s_i \rightarrow s_{i+1}) \tag{3.40}$$

où $\Omega_b$ représente l'ensemble des états $s_{i+1}$ connectés à l'état $s_i$.

Une fois les probabilités *a posteriori* de chaque transition d'état $P(s_i \rightarrow s_{i+1})$ déterminées, les probabilités des bits du message sont calculées d'après les équations (3.41) et (3.42).

$$P(d_i = 1|\mathbf{y}) = \sum_{S_1} P(s_i \rightarrow s_{i+1}|\mathbf{y}) \tag{3.41}$$

$$P(d_i = 0|\mathbf{y}) = \sum_{S_0} P(s_i \rightarrow s_{i+1}|\mathbf{y}) \tag{3.42}$$

où $S_1 = \{s_i \rightarrow s_{i+1} : d_i = 1\}$ est l'ensemble de toutes les transitions d'états associées au bit de donnée 1 et $S_0 = \{s_i \rightarrow s_{i+1} : d_i = 0\}$ l'ensemble de toutes les transitions d'états associées au bit de donnée 0. Enfin le LRV est donnée par l'équation (3.43).

$$\Lambda_i = \ln\frac{P(d_i = 1|\mathbf{y})}{P(d_i = 0|\mathbf{y})} = \ln\frac{\sum_{S_1}\alpha(s_i)\gamma(s_i \rightarrow s_{i+1})\beta(s_{i+1})}{\sum_{S_0}\alpha(s_i)\gamma(s_i \rightarrow s_{i+1})\beta(s_{i+1})} \tag{3.43}$$

L'algorithme du maximum *a posteriori* se déroule de la manière suivante :

1. Parcours du treillis en sens direct :
   (a) Créer un tableau $\alpha(j,i)$, avec $j \in \{0,\cdots,2^m-1\}$ et $i \in \{0,\cdots,K\}$. Ce tableau permet de mémoriser les probabilités $\alpha(s_i)$ calculées. Il est initialisé comme suit :

$$\alpha(j,0) = \begin{cases} 1 & \text{si } j=0, \\ 0 & \text{si } j \neq 1. \end{cases} \tag{3.44}$$

   (b) Effectuer $i=1$.
   (c) Effectuer $j=0$.
   (d) Effectuer $s_i = S_j$.
   (e) Mettre à jour $\alpha(s_i)$ d'après :

$$\alpha(j,i) = \sum_{s_{i-1}=S_{j'}\in\Omega_f} \alpha(j',i-1)\gamma(s_{i-1}\rightarrow s_i) \tag{3.45}$$

   où $\Omega_f$ représente l'ensemble des états $s_{i-1}$ connectés à l'état $s_i$.
   (f) Effectuer $j=j+1$.
   (g) Si $j=2^m$, tous les états ont été parcourus, aller à l'étape 1(h). Sinon retourner à l'étape 1(d).
   (h) Effectuer $i=i+1$.
   (i) Si $i=K$ alors la fin du treillis est atteinte, continuer à l'étape 2. Sinon retourner à l'étape 1(c).

2. Parcours du treillis en sens inverse :
   (a) Créer un tableau $\beta(j,i)$, avec $j \in \{0,\cdots,2^m-1\}$ et $i \in \{0,\cdots,K\}$. Ce tableau permet de mémoriser les probabilités $\beta(s_i)$ calculées. Il est initialisé de la manière suivante :
      - Si le treillis est terminé

$$\beta(j,K) = \begin{cases} 1 & \text{si } j=0, \\ 0 & \text{si } j \neq 1. \end{cases} \tag{3.46}$$

      - Sinon

$$\beta(j,K) = \frac{1}{2^m} \ \forall j \tag{3.47}$$

   (b) Effectuer $i=K-1$.
   (c) Effectuer $j=0$.
   (d) Effectuer $s_i = S_j$.

(e) Mettre à jour $\beta(s_i)$ suivant :

$$\beta(j,i) = \sum_{s_{i+1}=S_{j'} \in \Omega_b} \beta(j',i+1)\gamma(s_i \rightarrow s_{i+1}) \tag{3.48}$$

où $\Omega_b$ représente l'ensemble des états $s_{i+1}$ qui sont connectés à l'état $s_i$.

(f) Effectuer $j = j + 1$.

(g) Si $j = 2^m$, tous les états ont été parcourus, aller à l'étape 2(h). Sinon retourner à l'étape 2(d).

(h) Effectuer $i = i - 1$.

(i) Si $i = 0$ alors le début du treillis est atteint, continuer à l'étape 3. Sinon retourner à l'étape 2(c).

3. Calculer les LRV suivant :

$$\Lambda_i = \ln \frac{\sum_{S_1} \alpha(j,i)\gamma(s_i \rightarrow s_{i+1})\beta(j',i+1)}{\sum_{S_0} \alpha(j,i)\gamma(s_i \rightarrow s_{i+1})\beta(j',i+1)}, i \in \{0, \cdots, K-1\} \tag{3.49}$$

où $S_1 = \{(s_i = S_j) \rightarrow (s_{i+1} = S_{j'}) : d_i = 1\}$ est l'ensemble de toutes les transitions d'états associées au bit de donnée 1 et $S_0 = \{(s_i = S_j) \rightarrow (s_{i+1} = S_{j'}) : d_i = 0\}$, l'ensemble de toutes les transitions d'états associées au bit de donnée 0.

### 3.3.5 Algorithme Log MAP

Dans la pratique l'algorithme MAP est difficile à mettre en œuvre car il nécessite un grand nombre de calculs pour chaque bit estimé ($6 \times 2^m$ multiplications + $6 \times 2^m$ additions). De plus, il est sensible aux erreurs d'arrondis lorsqu'on utilise des valeurs numériques de précision finie. Ces deux contraintes pratiques sont résolues en exécutant l'algorithme MAP directement dans le domaine logarithmique, au lieu d'appliquer le logarithme au rapport de vraisemblance à la dernière étape. Dans le domaine logarithmique, toutes les multiplications se transforment en addition selon l'équation (3.50).

$$\begin{aligned} \ln(e^x + e^y) &= \max(x,y) + \ln(1 + \exp(-|y-x|)) \\ &= \max(x,y) + f_c(|y-x|) \end{aligned} \tag{3.50}$$

En considérant que $\Theta(x,y) = \max(x,y) + f_c(|y-x|)$ $\alpha_\ell(s_i)$ et $\beta_\ell(s_i)$ représentent respectivement

les logarithmes de $\alpha(s_i)$ et $\beta(s_i)$ alors :

$$\begin{aligned}
\alpha_\ell(s_i) &= \ln\left(\alpha(s_i)\right) && (3.51)\\
&= \ln\left(\sum_{s_{i-1}\in\Omega_f} \exp\left\{\alpha_\ell(s_{i-1}) + \gamma_\ell(s_{i-1}\rightarrow s_i)\right\}\right) && (3.52)\\
&= \Theta_{s_{i-1}\in\Omega_f}\left[\alpha_\ell(s_{i-1}) + \gamma_\ell(s_{i-1}\rightarrow s_i)\right] && (3.53)
\end{aligned}$$

où $\Omega_f$ est l'ensemble des états $s_{i-1}$ connectés à l'état $s_i$.

$$\begin{aligned}
\beta_\ell(s_i) &= \ln\left(\beta(s_i)\right) && (3.54)\\
&= \ln\left(\sum_{s_{i+1}\in\Omega_b} \exp\left\{\beta_\ell(s_{i+1}) + \gamma_\ell(s_i\rightarrow s_{i+1})\right\}\right) && (3.55)\\
&= \Theta_{s_{i+1}\in\Omega_b}\left[\beta_\ell(s_i) + \gamma_\ell(s_i\rightarrow s_{i+1})\right] && (3.56)
\end{aligned}$$

où $\Omega_b$ est l'ensemble des états $s_{i+1}$ connectés à l'état $s_i$. $\gamma_\ell(s_i \rightarrow s_{i+1})$ représente le logarithme de $\gamma(s_i \rightarrow s_{i+1})$ :

$$\begin{aligned}
\gamma_\ell(s_i\rightarrow s_{i+1}) &= \ln\left(\gamma(s_i\rightarrow s_{i+1})\right) && (3.57)\\
&= \ln P(d_i) + \ln P(y_i|c_i) && (3.58)
\end{aligned}$$

En considérant une modulation BPSK dans le cas d'un canal à évanouissement uniforme :

$$\begin{aligned}
\gamma_\ell(s_i\rightarrow s_{i+1}) &= \ln P(d_i) - \frac{1}{2}\ln\left(\frac{\pi K}{E_c}\right) - \frac{E_c}{K}\sum_{q=0}^{n-1}\left(y_i^{(q)} - a_i^{(q)}(2c_i^{(q)}-1)\right)^2\\
&= \lambda(s_i\rightarrow s_{i+1}) && (3.59)
\end{aligned}$$

Dans le domaine logarithmique, le calcul de la métrique d'une branche d'après l'algorithme Log MAP s'effectue de la même manière que pour l'algorithme de Viterbi. Par conséquent, Log MAP effectue le calcul des $\alpha_\ell(s_i)$ (parcours du treillis en sens direct) et des $\beta_\ell(s_i)$ (parcours du treillis en sens inverse) en utilisant une version modifiée de l'algorithme de Viterbi appelée algorithme de Viterbi général. La modification se situe au niveau de l'étape de calcul des métriques où est rajoutée la fonction $f_c(|y-c|)$. L'algorithme Log Map s'exécute de la même manière que l'algorithme MAP en procédant à quelques modifications reliées à son exécution dans le domaine logarithmique. Les étapes de l'algorithme MAP qui sont remplacées sont énumérées ci dessous :

1. Parcours du treillis en sens direct :

   **1(a)** Créer un tableau $\alpha_\ell(j,i)$, avec $j \in \{0,\cdots,2^m-1\}$ et $i \in \{0,\cdots,K\}$. Ce tableau permet

de mémoriser les probabilités $\alpha_\ell(s_i)$. Il est initialisé de la manière suivante :

$$\alpha_\ell(j,0) = \begin{cases} 0 & \text{si } j=0, \\ -\infty & \text{si } j \neq 1. \end{cases} \tag{3.60}$$

**1(e)** Mettre à jour $\alpha_\ell(s_i)$ selon :

$$\alpha_\ell(j,i) = \Theta_{s_{i-1}=S_{j'} \in \Omega_f}[\alpha_\ell(j',i-1) + \gamma_\ell(s_{i-1} \rightarrow s_i)] \tag{3.61}$$

où $\Omega_f$ représente l'ensemble des états $s_{i-1}$ connectés à l'état $s_i$.

2. Parcours du treillis en sens inverse :

**2(a)** Créer un tableau $\beta_\ell(j,i)$, $j \in \{0,\cdots,2^m-1\}$ et $i \in \{0,\cdots,K\}$. Ce tableau permet de mémoriser les probabilités $\beta_\ell(s_i)$. Il est initialisé de la manière suivante :

- Si le treillis est terminé

$$\beta_\ell(j,K) = \begin{cases} 0 & \text{si } j=0, \\ -\infty & \text{si } j \neq 1. \end{cases} \tag{3.62}$$

- Sinon

$$\beta_\ell(j,K) = 0 \ \forall j \tag{3.63}$$

**2(e)** Mettre à jour $\beta_\ell(s_i)$ selon :

$$\beta_\ell(j,i) = \Theta_{s_{i+1}=S_{j'} \in \Omega_b}[\beta_\ell(j',i-1) + \gamma_\ell(s_i \rightarrow s_{i+1})] \tag{3.64}$$

où $\Omega_b$ représente l'ensemble des états $s_{i+1}$ qui sont connectés à l'état $s_i$.

3. Calculer les LRV d'après :

$$\begin{aligned} \Lambda_i = \ & \Theta_{S_1}[\alpha_\ell(j,i) + \gamma_\ell(s_i \rightarrow s_{i+1}) + \beta_\ell(j',i+1)] \\ & -\Theta_{S_0}[\alpha_\ell(j,i) + \gamma_\ell(s_i \rightarrow s_{i+1}) + \beta_\ell(j',i+1)], \ i \in \{0,\cdots,K-1\} \end{aligned} \tag{3.65}$$

où $S_1 = \{(s_i = S_j) \rightarrow (s_{i+1} = S_{j'}) : d_i = 1\}$ est l'ensemble de toutes les transitions d'états associées au bit de donnée 1 et $S_0 = \{(s_i = S_j) \rightarrow (s_{i+1} = S_{j'}) : d_i = 0\}$, l'ensemble de toutes les transitions d'états associées au bit de donnée 0.

### 3.3.6 Algorithme Max-Log MAP

Comme pour le Log Map, l'algorithme Max-Log MAP s'exécute dans le domaine logarithmique. Seulement, dans ce cas, la fonction $f_c(|y-x|)$ de l'équation (3.50) est négligée, ce qui réduit $\Theta(x,y)$

à une simple fonction de maximisation (équ. (3.66)).

$$\ln(e^x + e^y) \approx \max(x,y) = \Theta(x,y) \tag{3.66}$$

Par conséquent, l'algorithme Max-Log MAP effectue le calcul des $\alpha_\ell(s_i)$ et $\beta_\ell(s_i)$ en utilisant l'algorithme de Viterbi. L'algorithme Max-Log Map (paragraphe 3.3.5) s'exécute de la même façon que le Log Map mais en prenant la fonction $\Theta(x,y) = \max(x,y)$.

## 3.4 Comparaison des algorithmes de décodage SISO

Du point de vue complexité, l'algorithme SOVA est le moins coûteux et nécessite moins de mémoire. Il peut être implanté en associant à un décodeur de Viterbi standard un co-processeur pour calculer les valeurs de confiance [90]. Dans le cas du SOVA amélioré, chaque sortie est multipliée par un facteur de correction dont la valeur dépend de la moyenne et de la variance des sorties. Chaque facteur doit être calculé au préalable ce qui a pour effet d'augmenter la complexité du circuit. L'algorithme Max-Log MAP réduit considérablement la complexité du MAP qui nécessite un grand nombre de multiplications et l'utilisation de fonctions non-linéaires [91]. L'exécution de l'algorithme dans le domaine logarithmique permet de réduire toutes les opérations à des additions et des calculs de « max » [91]. Le Max-Log MAP requiert cependant beaucoup de mémoire pour stocker les valeurs des métriques cumulées. Le Log MAP applique un terme de correction à l'algorithme du Max-Log MAP, ce qui lui permet de compenser l'erreur due à l'approximation $\ln \sum_j e^{a_j} \approx \max_j a_j$ (équ. (3.66)). A cet effet, une table de correspondance doit être utilisée pour stocker les facteurs de correction [91].

En pratique, pour réduire la complexité du circuit, l'algorithme SOVA peut s'exécuter uniquement sur des blocs partiels d'information. La taille du bloc partiel traité (*fenêtre*) doit être choisie suffisamment grande pour pouvoir supposer que les chemins survivants sont assez fiables. A chaque fin de traitement, le chemin qui a la plus faible métrique cumulée est choisi. En cas d'égalité dans la métrique de deux survivants, on peut utiliser un algorithme permettant de les départager ou en prendre un au hasard. L'algorithme SMAP (*sliding MAP*) permet de réduire les besoins en mémoire de l'algorithme MAP en s'exécutant sur des blocs partiels de données [92]. Cependant comme pour le SOVA, l'algorithme doit fournir un point de départ fiable pour le traitement des blocs partiels suivants. Cette technique existe aussi pour le Log MAP et le Max-Log MAP (SLog-MAP et SMax Log MAP). L'algorithme Log MAP apporte un gain de 0.5 dB par rapport au Max-Log MAP et le Max-Log MAP un gain de 0.5 dB par rapport au SOVA [5, 90].

Le tableau 3.1 donne un bref comparatif des principaux algorithmes de décodage SISO vue précédemment :

| | Log MAP | Max-Log MAP | SOVA | SOVA amélioré |
|---|---|---|---|---|
| Performance | -0.3 dB[a]<br>+0.5 dB[c] | +0.5 dB[b] | le moins performant | +0.3 à +0.5 dB[b] |
| Complexité | 2 × SOVA | Log-Map < ML Map<br>ML Map <SOVA | le moins complexe | SOVA < |

TAB. 3.1 – *Comparaison des algorithmes de décodage SISO*

[a] Dégradation par rapport au MAP.
[b] Amélioration par rapport au SOVA .
[c] Amélioration par rapport au Max-Log Map.

L'ensemble des algorithmes SISO passés en revue sont basés sur l'algorithme de Viterbi. Le SOVA et le SOVA amélioré utilisent des versions modifiées de Viterbi. L'algorithme Max-Log MAP utilise deux algorithmes de Viterbi, l'un parcourant le treillis en sens direct pour le calcul des $\alpha_\ell(s_i)$ et le deuxième parcourant le treillis en sens inverse pour le calcul les $\beta_\ell(s_i)$. Le Log Map enfin, utilise deux algorithmes de Viterbi généraux et s'exécute de la même façon que le Max-Log Map.

## 3.5 Etude architecturale d'un décodeur de Viterbi

Du fait que les algorithmes de décodage itératifs SISO, SOVA, Log MAP et Max-Log MAP, sont basés sur l'algorithme de Viterbi, cette partie est consacrée à l'étude architecturale d'un décodeur de Viterbi et plus particulièrement au bloc ACS (*Add Compare Select*) qui est traditionnellement le bloc critique limitant le débit du décodeur [93]. L'architecture classique de l'algorithme de Viterbi est composée de trois blocs :

**BMC (*Branch Metric Calculation*)** Ce bloc calcule les métriques de chaque branche du treillis à partir des données reçues. Le bloc BMC peut être implanté à l'aide d'un additionneur ou d'une ROM.

**ACS (*Add Compare Select*)** Ce bloc fait le calcul et la mise à jour des métriques des différents chemins du treillis, sélectionne le chemin survivant qui a la plus grande[3] ou la plus faible[4] métrique, selon la phase d'initialisation, et fournit le nœud correspondant dans le treillis. Comme de plus, les valeurs des métriques sont croissantes et pour éviter tout risque de dépassement de capacité (*overflow*) on procède à une normalisation des valeurs accumulées. Cette normalisation consiste à soustraire chaque fois qu'un seuil est atteint, une valeur commune à toutes les

3. Si la phase d'initialisation est effectuée d'après l'équation (3.23), l'algorithme de Viterbi choisit la métrique la plus grande (cf. paragraphe 3.3.1).

4. Si la phase d'initalisation s'effectue d'après l'équation (3.67), l'algorithme de Viterbi choisit la métrique la plus faible [66].

$$\Gamma(j,0) = \begin{cases} 0 & \leftarrow \ j = 0, \\ +\infty & \leftarrow \ j \neq 0. \end{cases} \tag{3.67}$$

métriques. Une première méthode, consiste à définir un seuil de normalisation, en général fixé à $2^n - 1$ et où $n$ est le nombre de bits des métriques cumulées, puis à le soustraire de chaque métrique une fois qu'il est dépassé. Une autre méthode consiste à fixer un pas de normalisation qui est soustrait à chaque itération de toutes les métriques calculées. L'approche directe consiste à soustraire systématiquement la métrique la plus faible aux métriques cumulées à chaque pas de calcul.

SMU (***Survivor Memory Unit***) Ce bloc permet le stockage, à chaque cycle, des $2^m - 1$ résultats de la sélection du bloc ACS, et de reconstituer la séquence émise par la relecture des chemins survivants stockés. Deux méthodes principales peuvent être utilisées. Soit l'échange de registres [94], soit le chaînage arrière [95]. La combinaison des deux est possible.

Afin d'augmenter le débit deux approches sont envisageables. L'approche algorithmique consiste à reformuler l'algorithme de décodage afin d'exploiter les éventuels traitements concurrents [96] et l'approche architecturale. L'approche architecturale (directe) consiste à traiter les informations reçues par plusieurs décodeurs mis en parallèle. Dans ce cas, le débit est proportionnel à la complexité du circuit [97]. D'un autre coté, le débit des décodeurs SOVA étant principalement limité par la conception du bloc ACS, des solutions visant à optimiser le chemin critique de ce dernier sont envisageables. En effet, la principale contrainte du bloc ACS réside dans sa nature récursive qui empêche l'utilisation de structures pipelines pour en réduire le chemin critique [93]. Une première solution consiste à dérouler la boucle récursive de la structure ACS [98, 99] : cette approche augmente le chemin critique du bloc mais améliore le débit global. Une deuxième solution consiste à transformer le bloc ACS en bloc CSA en réorganisant et remplaçant les opérations successives {addition, comparaison, sélection} par {comparaison, sélection, addition} [100, 101]. L'approche la plus simple se base sur les structures Radix-$k$ du bloc ACS. Ce dernier est dupliqué puis associé à chaque état du codeur pour le calcul des métriques [102].

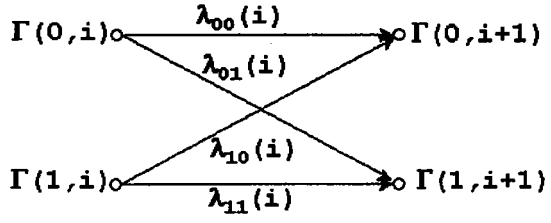



FIG. 3.4 – *Treillis Radix-2.*

La forme la plus simple du treillis est appelée Radix-2 et est représentée sur la figure 3.4. A chaque itération, l'algorithme de Viterbi met à jour la métrique $\Gamma(j,i)$ correspondant au nœud $S_{(j,i)}$ du treillis. Pour cela, il additionne les métriques $\lambda_{j'j}(i)$ liées aux branches entrant dans chaque nœud $S_{(j,i+1)}$ avec celles cumulées lors de l'itération précédente $\Gamma(j',i)$ (addition), compare les valeurs

calculées des métriques (comparaison) et choisit le chemin (sélection) dont la métrique est la plus grande ou la plus faible selon la phase d'initialisation[5].

Si l'on considère que la phase d'initialisation s'effectue selon (3.67), alors les métriques $\Gamma(0,i+1)$ et $\Gamma(1,i+1)$ correspondant aux nœuds $S_{(0,i+1)}$ et $S_{(1,i+1)}$ du treillis de la figure 3.4 se calculent avec :

$$\Gamma(0,i+1) = \min\{\Gamma(0,i)+\lambda_{00}(i),\Gamma(1,i)+\lambda_{10}(i)\} \quad (3.68)$$

$$\Gamma(1,i+1) = \min\{\Gamma(0,i)+\lambda_{01}(i),\Gamma(1,i)+\lambda_{11}(i)\} \quad (3.69)$$

Le processeur élémentaire qui permet le calcul de la métrique d'un nœud du treillis ($S_{(0,i+1)}$ dans l'exemple) est représenté sur la figure 3.5. En combinant deux processeurs élémentaires, on obtient un bloc ACS Radix-2 (fig. 3.6).

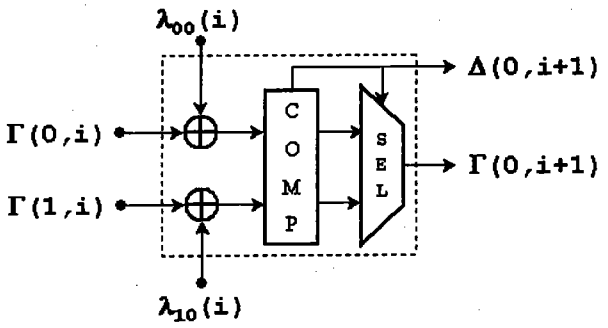



FIG. 3.5 – *Processeur élémentaire Radix-2.*

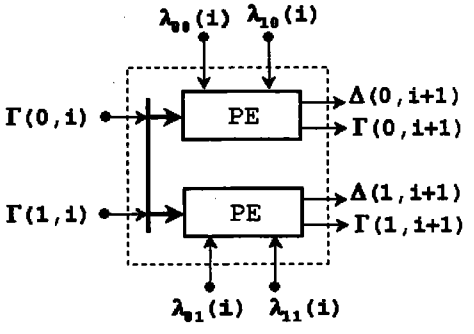



FIG. 3.6 – *ACS Radix-2.*

La complexité du circuit décodeur augmente proportionnellement avec la complexité du treillis. Une approche pour la réduire consiste à réorganiser le treillis de manière à aboutir à deux ou plusieurs sous-treillis simples indépendants les uns des autres. Bien sûr, cette décomposition ne peut se faire que si la structure du treillis le permet. La méthode consiste à isoler les états appartenant au même groupe de transitions. Ainsi, chaque fois que le système se trouve dans un sous-treillis donné, il ne peut transiter que vers un état appartenant au même groupe. Les figures 3.7 et 3.8 représentent respectivement un treillis Radix-2 à 4 états et sa décomposition en deux sous-treillis Radix-2.

5. Si la phase d'initialisation est effectuée d'après l'équation (3.23), l'algorithme de Viterbi choisit la métrique la plus grande (cf. paragraphe 3.3.1). Si la phase d'initalisation s'effectue d'après l'équation (3.67), l'algorithme de Viterbi choisit la métrique la plus faible [66].

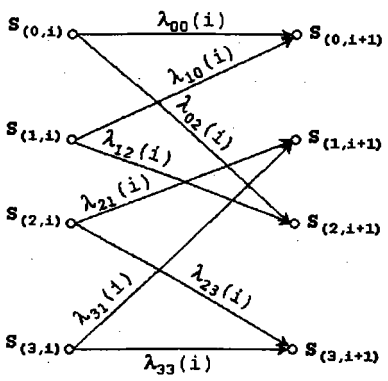



FIG. 3.7 – *Treillis Radix-2 à 4 états.*

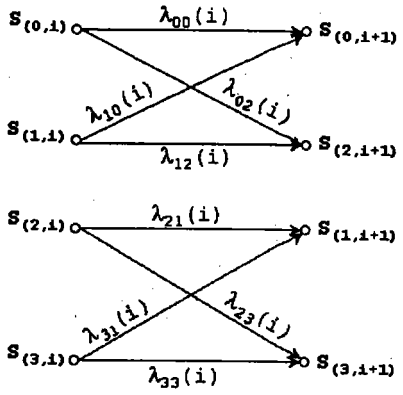



FIG. 3.8 – *Deux sous-treillis Radix-2.*

La décomposition du treillis permet de réduire la complexité architecturale du circuit en divisant celui-ci en un ensemble de circuits simples. Cette décomposition permet d'adapter l'algorithme de calcul à des architectures dédiées telles que, par exemple, le DSP TMS320C554x qui a une architecture interne adaptée au calcul des structures en papillon (Radix-2).

Une autre approche consiste à compresser le treillis de sorte à calculer les métriques $\Gamma(j,i+k)$ à partir des métriques $\Gamma(j,i)$ et des métriques des $k$ branches $\lambda_{j'j}(i+1), \cdots, \lambda_{j'j}(i+k)$, ce qui correspond au passage d'une structure Radix-2 à une structure Radix-$2^k$. Les figures 3.9 et 3.10 présentent un exemple de passage d'une structure Radix-2 à une structure Radix-4.

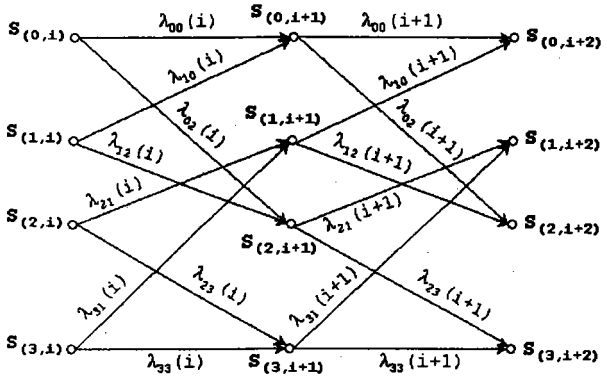



FIG. 3.9 – *Treillis Radix-2.*

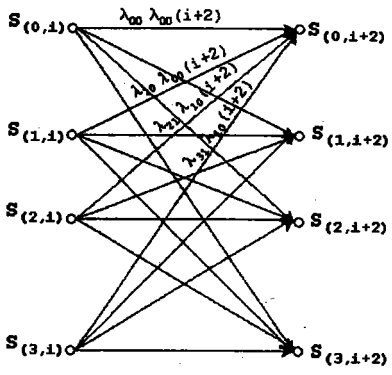



FIG. 3.10 – *Treillis équivalent Radix-4.*

Le processeur élémentaire qui permet le calcul de la métrique d'un nœud du treillis ($S_{(0,i+2)}$ dans l'exemple) est représenté sur la figure 3.11. En combinant quatre processeurs élémentaires Radix-4 on obtient un bloc ACS Radix-4 (fig. 3.12).

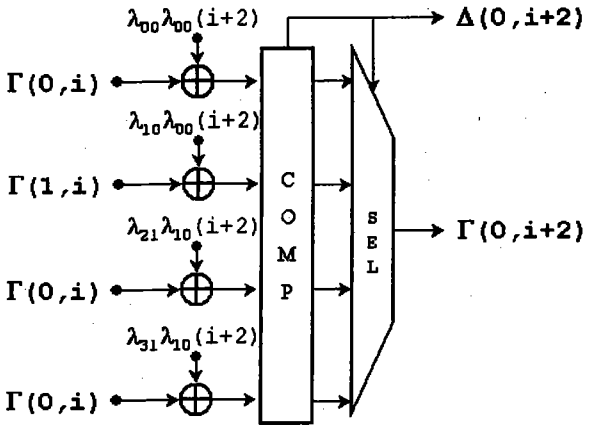



FIG. 3.11 – *Processeur élémentaire Radix-4.*

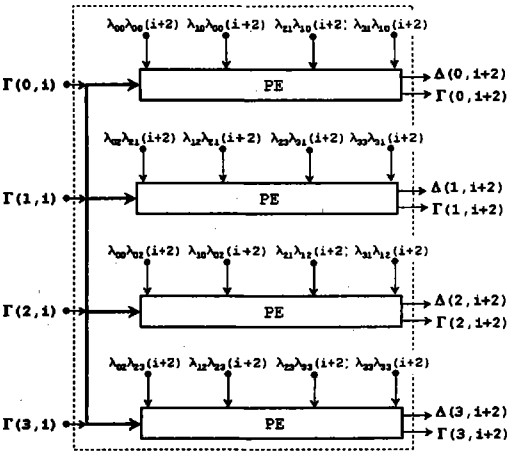



FIG. 3.12 – *ACS Radix-4.*

Dans le cas d'une structure Radix-2, $k$ itérations sont nécessaires pour calculer la métrique $\Gamma(j, i+k)$ alors que dans le cas Radix-$2^k$, seule une itération est nécessaire. La complexité combinatoire du bloc ACS Radix-$2^k$ dépend pour chaque itération du traitement de $2^k$ chemins possibles du treillis, alors que dans une structure Radix-2, seuls 2 chemins (fig. 3.5) sont pris en compte. En supposant le cas idéal où les processeurs élémentaires traitent respectivement $2^k$ chemins et où 2 chemins génèrent les mêmes délais de propagation (chemin critique) alors la vitesse du circuit est augmentée d'un facteur $k$ pour une augmentation de la complexité de $2^{k-1}$.

Le tableau 3.2, dans le cas idéal, donne les augmentations de vitesse et de la complexité de différentes structures Radix-$2^k$.

| Radix | k | Augmentation de la vitesse (cas idéal) | Augmentation de la complexité |
|---|---|---|---|
| 2 | 1 | 1 | 1 |
| 4 | 2 | 2 | 2 |
| 8 | 3 | 3 | 4 |
| 16 | 4 | 4 | 8 |

TAB. 3.2 – *Mesures de la vitesse & de la complexité des structures Radix-$2^k$.*

Le tableau 3.3 représente, pour une implantation sur un FPGA de type Flex10KE d'Altera ($f_{max}$ = 250 MHz), les fréquences de fonctionnement et les surfaces occupées par des processeurs élémentaires Radix-$2^k$ ayant des entrées à 4 bits.

| Radix | k | Fréquence(MHz) | Surface (CLs) |
|---|---|---|---|
| 2 | 1 | 149,25 | 35 |
| 4 | 2 | 99 | 78 |
| 8 | 3 | 49.01 | 202 |
| 16 | 4 | - | - |

TAB. 3.3 – *Implantation de processeurs élémentaires Radix-$2^k$.*

La figure 3.13 présente les fréquences de fonctionnement et les débits atteints par les différentes structures de processeurs élémentaires Radix-$2^k$ en fonction des surfaces occupées.

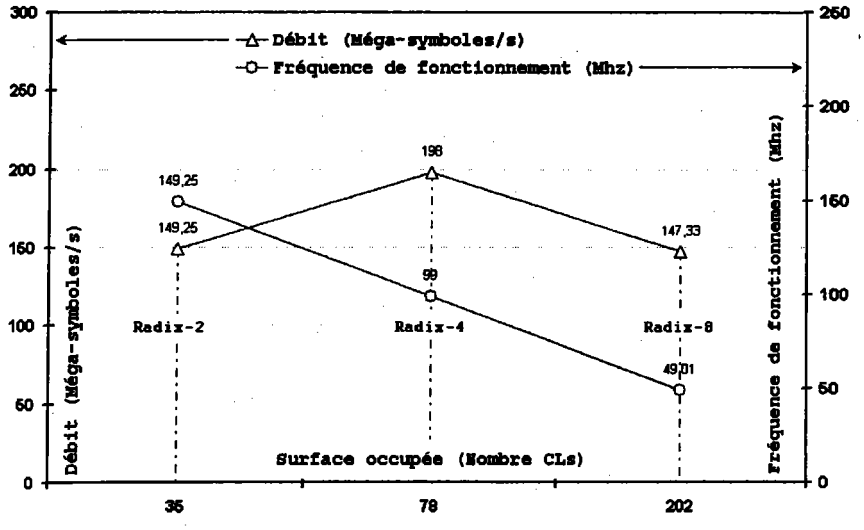



FIG. 3.13 – *Compromis entre (fréquence,débit) et surface des processeurs élémentaires Radix-$2^k$.*

Comme le montre la courbe, la fréquence de fonctionnement décroît en fonction de $k$. On remarque que la structure Radix-4 offre le meilleur compromis débit/surface.

## 3.6 Quelques réalisations de turbo-codeur

Il existe actuellement sur le marché des réalisations matérielles de turbo-codeurs en technologies ASIC ou FPGA. On trouve aussi des circuits virtuels (IP : *Intellectual Property*) permettant d'implanter des turbo-codes. Citons en quelques unes :

**Comatlas**

Le CAS5093, développé en 1995 par la firme Comatlas en collaboration avec l'ENST Bretagne à été implanté en technologie CMOS 0.8 $\mu m$. Il est basé sur un turbo-codeur à cinq étages. Il fonctionne avec un rendement fixe de 0.5 pour des blocs d'information de 1024 bits et peut atteindre un débit de

40 Mbits/s avec un temps de latence de 2318 bits. Ses performances vont de $9 \cdot 10^{-7}$ dB à 3 dB avec une amélioration de 0.5 dB par décade [103].

**Small World Communications**

Les circuits MAPO4T et MAP04B ont été conçus en 1996 par Small World Communications [132] en collaboration avec S. Pietrobon [66]. Ces circuits implantent des turbo-codes convolutifs avec un algorithme de décodage MAP SISO. Le MAPO4T ne fonctionne que sur 16 états tandis que le MAP04B peut fonctionner à 4, 8 ou 16 états. Leur rendement paramétrable est de 1/2, 1/3 ou 1/4. Les débits de décodage atteints sont de 31 Mbits/s avec une latence maximale de 400 bits pour le MAPO4T et 2,4 Mbits/s avec une latence comprise entre 66 et 257 bits pour le MAP04B [66, 104]. Plus récemment les améliorations apportées permettent au circuit MAPO4T d'atteindre des débits de décodage de 95 Mbit/s sur des FPGA de type Virtex (Xilinx) pour une latence comprise 400 et 784 cycles [105]. Le circuit MAP03T [106], quant à lui, permet des décodages jusqu'à 120 Mbit/s sur 8 états. Le circuit PCE03V [107] est un turbo-codeur convolutif à 8 états compatible avec les normes 3GPP/3GPP2 ($3^{rd}$ *Generation Partnership Project*)[6]. Il permet des rendements de 1/2, 1/3, 1/4 et 1/5 et fonctionne jusqu'à des débits de 42 Mbits/s. Selon la norme, les tailles des entrelaceurs varient de 40 à 5114 bits pour la norme 3GPP et 17 à 32768 pour la norme 3GPP2.

**Advanced Hardware Architectures, Inc.**

En 1998, Spectra Licensing Group (représentant de France Télécom en Amérique du Nord) a accordé une licence d'exploitation de la technologie des turbo-codes à la compagnie Comtech AHA Corporation [133]. Cette dernière a développé plusieurs circuits codeurs/décodeurs de turbo-codes produits. Le circuit AHA4501 [108] est implanté en technologie CMOS 0.35 $\mu m$ et offre des rendements de 0.325 à 0.793 pour des blocs d'information de 256 à 4096 bits. Il offre un débit de codage de 50 Mbits/s et de décodage de 36 Mbits/s. Le AHA4522 [109] et le AHA4524 [110] sont destinés aux applications sans fil et satellite. Ils ont un rendement compris entre 0,25 et 0,97 pour des blocs de 64 bits à 2 Kbits pour le premier et de 64 à 4 Kbits pour le deuxième. Leurs débits sont légèrement inférieurs à 30 Mbits/s. Le AHA4525 [111] supporte le standard IEEE 802.16a. Son rendement est compris entre 0,25 et 0,97. Il permet de coder des trames de 64 bits à 4 Kbits avec un débit de 50 Mbit/s. Le AHA4540 [112] fonctionne à un débit de 155 Mbits/s pour une fréquence interne de 86 MHz. Il apporte une amélioration de 2 à 4 dB par rapport à un codage convolutif ou RS conventionnel.

**STMicroelectronics**

Après l'acquisition d'une licence d'exploitation des turbo-codes, STMicroelectronics a développé

6. Le 3GPP est une organisation internationale qui réunit six sociétés (ARIB, CWTS, ETSI, T1, TTA et TTC.). Il est considéré comme le principal organisme de standardisation mondiale par le Japon, la Chine, l'Europe, les Etats-Unis et la Corée.

le STV0399 en technologie CMOS 0.18 $\mu m$ qui englobe une fonction de turbo-codage et de modulation 8PSK [113].

**TurboConcept**

La firme française TurboConcept [134] fournit des cœurs IP qui permettent d'implanter des turbo-codes sur différents supports technologiques. La fonction IP TC1000 est utilisée dans la norme DVB-RCS [114]. Elle existe en version optimisée pour les circuits APEX d'Altera et pour les Virtex de Xilinx. Elle offre des rendements de 1/3, 2/5, 1/2, 2/3, 3/4 et 4/5 pour des blocs de 12 à 216 octets. Le codeur implanté dans un Virtex-II permet d'atteindre un débit de 11 Mbits/s. Le TC2000 est une version améliorée du TC1000 permettant de coder des blocs de 1000 octets et d'atteindre[7] un BER de $10^{-9}$ pour des rapports signal sur bruit de 2 dB avec un rendement de 1/2, de 2,9 dB avec un rendement de 2/3 et de 3,6 dB avec un rendement de 3/4. Leurs algorithmes de décodage sont basés sur Log MAP. D'autres fonctions IP pour des codes produits sont aussi disponibles. Le TC3000 dédié aux applications spécifiques IEEE 802.16 est basé sur les codes de Hamming ou BCH. Il permet de coder des trames jusqu'à 65 Kbits. Ses débits de fonctionnement sont de 38 Mbits/s pour une implantation sur un Virtex-II de Xilinx et de 42 Mbits/s sur un Stratix d'Altera. Le TC3401 est un cœur IP développé spécifiquement pour les circuits FPGA à faible coût d'Altera et Xilinx. Les performances atteintes sont de 40 Mbits/s pour une implantation sur un FPGA Cyclone d'Altera et de 30 Mbits/s sur un Spartan 2e de Xilinx. Enfin, le TC3404 permet d'atteindre des débits maximaux de 180 Mbits/s sur FPGA de type Xilinx/Virtex-II Pro ou Altera/Stratix.

**iCoding technologie incorporated**

La firme américaine iCoding Technologie Incorporated [135] fournit différents cœurs IP pour des turbo-codes convolutifs implantables en ASIC ou sur FPGA de type Virtex-II de Xilinx [115]. La fonction IP S1000 destinée aux applications DVB, fibres optiques et enregistrement de données peut fonctionner à des débits de 125 Mbits/s en ASIC et de 40 Mbits/s sur FPGA. Le S2000, qui satisfait aux normes DVB-RCS et DVB-RCT, est destiné aux applications de télédiffusion numérique. Le circuit atteint 60 Mbits/s en ASIC et le tiers sur FPGA. La fonction S3000 est dédiée aux applications de téléphonie mobile et de transmission de la voix et de données (norme GPP) à des débits de 21 Mbits/s en ASIC et de 8 Mbits/s sur FPGA. Le S4000 satisfait aux standards 802.16 et HyperMan, et peut atteindre 320 Mbits/s en ASIC et 155 Mbits/s sur FPGA. Le S5000, normalisé au standard Inmarsat, est destiné aux applications de téléphonie mobile Satcomms. Son implantation sur FPGA atteint un débit de 2 Mbits/s. Le S6000 satisfait aux normes CCSDC et 101.0-B-5. Il fonctionne à 4 Mbits/s sur FPGA. Enfin, le S7000 destiné aux normes Echostar/Broadcom et Advanced Broadcast atteint 110 Mbits/s en ASIC et 40 Mbits/s sur FPGA.

7. Performances indiquées dans le cas de paquets de données de format MPEG avec modulation QPSK.

**Xilinx**

Xilinx [136] a développée un cœur IP pour codes produits optimisé pour des FPGA de type Virtex-II et Virtex-II Pro [116]. Ce cœur IP est normalisé suivant les standards IEEE 802.16 et 802.16a. Le circuit accepte des blocs de 64 bits à 4 Kbits. Les débits atteints sont compris entre 45 Mbits/s et 155 Mbits/s selon que l'on opte pour l'utilisation d'un seul décodeur SISO ou pour quatre. Il permet d'atteindre un gain de 7 dB pour un BER de $10^{-6}$ avec une modulation BPSK dans un canal AWGN. Les fréquences maximales de fonctionnement sont respectivement de 150 MHz et 170 MHz pour une implantation dans un Virtex-II ou un Virtex-II Pro.

**Amphion Semiconductor, Ltd.**

La société Amphion [138] offre plusieurs fonctions IP optimisées pour des applications SoC (*System-on-a-Chip*). Leurs IP sont dénommés ASVC (*Application Specific Virtual Components*). Dans leur gamme de produits, on trouve le turbo codeur CS3530 [117] et le décodeur CS3630 [118] destinés a être implantés en ASIC TSMC (*Taiwan Semiconductor Manufacturing Corporation*) 180 $nm$ ou sur FPGA de type Apex20KE d'Alera ou Virtex-E de Xilinx. Ils sont compatibles avec les normes 3GPP et CDMA2000 (TIA/EIA/IS-2000.2-A) et permettent un débit moyen de 2,048 Mbits/s sur ASIC. On trouve aussi le CS3520 et le CS3620, dédiés aux normes CDMA2000, et les CS3510 et CS3610 qui satisfont aux standards 3GPP.

**Autres réalisations**

D'autres turbo-codeurs/décodeurs ont été réalisés pour différentes technologies. Dans [119] par exemple, un turbo-décodeur à été développé sur le DSP ADSP-21061 SHARC d'Analog Device fonctionnant à 40 MIPS (*MIllion of Instructions Per Second*). Le circuit ne requiert aucune mémoire externe et fonctionne à un débit moyen de 70 Kbits/s. Dans [120], un turbo-décodeur basé sur l'algorithme de décodage Max-Log MAP a été implanté sur le DSP StarCore SC140 ayant une architecture SWP (*Sub-Word Parallel*) [121]. Le centre de recherches sur les communications du Canada [137] propose des modèles de turbo-codeurs/décodeurs simulables sur PC. Les turbo-codes utilisent deux codes constituants à 16 états. Ils atteignent une vitesse de décodage supérieure à 1 Mbits/s sur un ordinateur à processeur AMD MP 2100+ cadencé à 1,8 GHz, pour huit itérations fixes de décodage. Dans [122], l'implantation série des algorithmes de décodage SOVA et MAP sur circuits VLSI en technologie 8 $\mu m$ conduit respectivement à une complexité de 10 $mm^2$ et 13 $mm^2$. Deux circuits décodeurs pour un turbo-code série, de rendements 2/3 et 3/4, ont permis d'atteindre un débit de 2 Mbits/s sur une puce de 35 $mm^2$ dont 23,4 $mm^2$ sont occupés par de la RAM et seulement 3 $mm^2$ par le décodeur de rendement 2/3 et 5 $mm^2$ par celui de rendement 3/4.

## 3.7 Conclusion

Ce dernier chapitre de la première partie donne une description détaillée des algorithmes de décodage itératif des turbo-codes. Le circuit décodeur est composé de décodeurs élémentaires de type SISO fonctionnant de manière itérative. Le choix de l'algorithme détermine, d'une part, les performances de décodage et, d'autre part la complexité du circuit décodeur. Les algorithmes les plus simples à mettre en œuvre sont le SOVA et le SOVA amélioré alors que les plus coûteux en surface sont ceux dérivés du MAP, à savoir le Log Map et le Max-Log Map. Cependant, des versions moins onéreuses existent, comme par exemple le SMAP, le SLog-MAP et le SMax Log MAP qui s'exécutent sur des blocs partiels d'information. Tous les algorithmes SISO traités, à savoir SOVA, SOVA amélioré, Log Map et Max Log-Map, se basent sur des versions modifiées de l'algorithme de Viterbi.

D'un autre coté, les débits des décodeurs Viterbi sont principalement limités par la conception du bloc ACS. En effet, sa nature récursive empêche l'utilisation de structures pipelines destinées à en réduire le chemin critique [93]. Cependant, plusieurs solutions visant à optimiser ses performances existent [98, 99, 100, 101]. L'approche la plus utilisée consiste à compresser le treillis, de manière à traiter plus de bits reçus. Ceci conduit à des structures ACS Radix-$k$ dont la complexité augmente en fonction de $k$. L'implantation sur un FPGA de type Flex10KE d'Altera de différentes structures Radix-$k$ a permis de déduire qu'au-delà de $k = 4$, la fréquence se dégrade considérablement, rendant cette technique inefficace.

A ce stade de l'étude, on retient qu'à chaque configuration d'un turbo-code correspond une distance minimale de code, qui détermine les performances du circuit en terme de protection. Notre objectif est d'augmenter le débit de traitement des données sans dégrader la distance minimale du code, ce qui revient à augmenter le débit des codes constituants et des entrelaceurs. Dans le cas de l'entrelaceur, cela peut se faire aisément en augmentant par exemple le nombre d'entrées du générateur de permutations sans augmenter la complexité combinatoire du circuit. Par contre, dans le cas d'un codeur récursif, l'application d'un parallélisme combinatoire pur entraîne une dégradation considérable de la fréquence, ce qui conduit à considérer d'autre solutions architecturales. Dans la suite, notre étude porte sur deux architectures possibles. A travers le chapitre 4 nous explorons les possibilités et les limitations des architectures systoliques dédiées à la convolution. Dans le chapitre 5, nous proposons une architecture de codeur convolutif à haut débit qui combine les deux techniques du parallélisme et du pipelining, puis nous étudierons son application aux turbo-codes. Le chapitre 6 enfin, présente un décodeur de syndrome haut débit exploitant l'architecture rapide du codeur convolutif.

# II. ETUDE ARCHITECTURALE ET IMPLANTATION



# Chapitre 4

# Architectures systoliques pour la convolution

A travers ce chapitre nous aborderons les architectures systoliques appliquées au calcul de la convolution. Trois types de convolutions seront traités : la convolution non récursive, la convolution récursive et la convolution récursive générale. Du point de vue architectural, différentes structures systoliques sont capables de traiter un même type de convolution.

La première partie de ce chapitre présente l'approche systolique, ses avantages et inconvénients ainsi que ses domaines d'application. La deuxième partie sera consacrée à l'étude de quelques réseaux systoliques dédiés au calcul de la convolution. Nous commencerons par les réseaux systoliques bidirectionnels, en référence au sens de propagation des données à travers le réseau. Ce type de réseau est d'efficacité 2, c'est à dire qu'il fournit un résultat uniquement un cycle d'horloge sur deux. La deuxième structure présentée, dite unidirectionnelle, apporte une amélioration de l'efficacité au détriment de la surface et du temps de latence. Quant à la troisième technique, dite « diviser pour régner », elle permet de résoudre le problème de latence. Enfin, le réseau orthogonal permet le calcul de la convolution en flot continu.

Chaque type de convolution sera présenté sous la forme d'équation fonctionnelle et par la description architecturale systolique correspondante. Chaque modèle décrit a été simulé sous MAX+plus II pour des circuits FPGA de type Flex10KE d'Altera.

**Mots clés :** architecture systolique, convolution.

## 4.1 Introduction

Le concept systolique a été introduit en 1978 par KUNG et LEISERSON. Le principe consiste à décomposer un problème donné ou un calcul compliqué, lorsque cela est possible, en un ensemble d'opérations itératives simples de façon à en faciliter la résolution. De ce fait, une architecture systolique se présente sous la forme d'un réseau de processeurs simples (élémentaires) et identiques connectés localement selon une topologie bien définie. Chaque processeur élémentaire exécute un

calcul simple sur les données et transmet le résultat aux cellules voisines. Seules les cellules situées aux extrémités du réseau communiquent avec l'extérieur. Le réseau est piloté par une seule horloge, ce qui le rend totalement synchrone dans son fonctionnement d'où le terme systolique qui se rapporte à la notion de battement, ou fonctionnement par battement à la cadence d'une horloge : $clk_{sys}$ dans ce cas précis.

## 4.2 Avantages et inconvénients des architectures systoliques

### 4.2.1 Principaux avantages

En comparaison avec une architecture classique, une architecture systolique présente de nombreux avantages.

**Régularité et simplicité :** les architectures systoliques sont composées d'une ou plusieurs cellules de base, simples et identiques. Cette régularité permet, d'une part, une intégration à très haute échelle, notamment la conception de circuits spécialisés de type ASIC en technologie VLSI (*Very Large Scale Integration.*), et d'autre part, la diminution du coût de fabrication qui doit être assez faible pour pouvoir être amorti sur de faibles volumes de fabrication.

**Reconfigurabilité :** cette caractéristique découle directement de l'aspect modulaire de ce type d'architectures. En effet, elles sont adaptables à la taille et à la nature du problème traité. Pour résoudre par exemple, un problème de même type mais de plus grande taille, il suffit de rajouter des cellules élémentaires.

**Commande aisée :** en plus de l'aspect régulier et local des interconnexions entre cellules, celles-ci sont simples et en nombre réduit, ce qui rend la commande d'un système systolique aisée.

**Concurrence et communication :** dans les systèmes séquentiels classiques, les accès à la mémoire sont fréquents. A chaque utilisation d'une donnée correspond un accès à la mémoire. Dans une architecture systolique, une fois que la donnée intègre le réseau, elle circule d'une cellule à une autre pour subir des traitements successifs.

**Testabilité et résistance aux pannes :** le fait que les cellules soient identiques en facilite le test : il suffit d'en tester une de chaque groupe identique. De plus, l'association au réseau systolique d'un système permettant de le reprogrammer en cas de pannes de cellules est relativement aisé. En effet, le caractère local des interconnexions permet de contourner facilement les cellules défectueuses.

**Adaptation aux calculs intensifs :** pour la plupart des algorithmes de type *compute-bound*[1], les réseaux systoliques permettent un traitement en temps réel, ce qui les rend très adaptés aux problèmes nécessitant un traitement régulier avec de gros volumes de calcul.

### 4.2.2 Principaux inconvénients

Les deux principales restrictions liées à l'utilisation d'un processeur systolique sont :

**Spécialisation :** les processeurs systoliques sont des systèmes spécialisés, d'où la nécessité de les associer à un processeur maître de structure conventionnelle.

**Domaine d'application :** ce type d'architecture trouve son efficacité optimale dans les applications où le volume de calcul prime largement sur les transferts de données à réaliser. Néanmoins, son champ d'application reste vaste.

## 4.3 Domaine d'applications des réseaux systoliques

Comme dit précédemment, les algorithmes de calcul existants ne sont pas tous adaptés aux structures systoliques. Les architectures systoliques peuvent cependant être employées dans de nombreux domaines [124, 125, 126] tels que :

**Traitement du signal :**

- filtrage, convolutions 1D et 2D, corrélation, transformée de Fourier discrète, projections géométriques, etc ;
- codage canal, notamment codes correcteurs d'erreurs (FEC).

**Calcul matriciel :**

- multiplication matricielle : matrice-vecteur, matrice-matrice, etc. ;
- triangularisation : résolution de systèmes linéaires, inversion de matrice, etc.

**Traitement non numérique :**

- structures de données : piles, files d'attente, recherche dans un dictionnaire, etc. ;
- manipulation de chaînes de caractères ;
- algèbre des polynômes : division euclidienne, multiplication, pgcd[2], etc. ;
- graphes et algorithmes géométriques.

1. Un système *compute-bound* est un système dont le nombre de calculs élémentaires est plus grand que le nombre de données en entrée et en sortie du système considéré. Dans le cas inverse le système est dit *I/O-bound : input/output-bound*

2. pgcd : Plus Grand Commun Dénominateur.

## 4.4 Convolution non récursive

**Notations et terminologies** Avant de commencer la présentation des descriptions fonctionnelles, voici les notations et terminologies qui seront utilisées :

- la notation $(\aleph)_t$ représente le contenu du registre (ou bascule) $\aleph$ à l'instant $t$ ;
- le paramètre $t$ représente le temps. Si l'on note l'instant initial $t_0$ et que $T = 1/f_{clk}$ représente la période de l'horloge du système, en supposant que le contenu du registre $R$ à l'instant $t_0$ est $x_0$ et que son contenu à l'instant $t_0 + 1 \cdot T$ est $x_1$, etc., on a alors de manière générale :

$$\begin{array}{lllll} (R)_{t_0+0\cdot T} & = & (R)_{t_0} & = & x_0 \\ (R)_{t_0+1\cdot T} & = & (R)_{t_1} & = & x_1 \\ \cdots & \cdots & \cdots & \cdots & \\ (R)_{t_0+i\cdot T} & = & (R)_{t_i} & = & x_i \\ (R)_{t_0+(i+1)\cdot T} & = & (R)_{t_{i+1}} & = & x_{i+1} \\ \cdots & \cdots & \cdots & & \end{array}$$

où $(R)_{t_{i+1}}$ est la représentation simplifiée du contenue du registre $R$ à l'instant $(t_0 + (i+1) \cdot T)$.

Deux types d'équations seront utilisées :

- les équations faisant intervenir des entités telles que les variables $x_i$ ou $y_i$, indépendamment d'une architecture particulière seront appelées équations fonctionnelles ;
- les équations faisant intervenir des entités matérielles telles que les registres, décrivant le fonctionnement d'une structure seront appelées équations architecturales.

### 4.4.1 Calcul de la convolution non récursive

La formule générale de la convolution non récursive est décrite par l'équation fonctionnelle suivante :

$$y_i = \sum_{j=0}^{q} x_{i-j} \cdot a_j \tag{4.1}$$

où les $x_i$ représentent les données et les $a_i$ sont des coefficients fixes. L'équation (4.1) peut être décomposée de la façon suivante :

| **cycle** | **valeur $y_i$** |
|---|---|
| $t+0$ | $\rightarrow \quad y_0 = a_0 x_0$ |
| $t+1$ | $\rightarrow \quad y_1 = a_0 x_1 + a_1 x_0$ |
| $t+2$ | $\rightarrow \quad y_2 = a_0 x_2 + a_1 x_1 + a_2 x_0$ |
| $\cdots$ | $\rightarrow \quad \cdots$ |
| $t+i$ | $\rightarrow \quad y_i = a_0 x_i + a_1 x_{i-1} + \cdots + a_q x_{i-q}$ |

A partir de l'équation (4.2) on déduit les règles de calcul suivantes [123] :

- tous les $a_i$ et $x_j$ doivent se croiser en même temps dans la cellule qui calcule le résultat $y_{i+j}$ ;
- Les couples $(a_i, x_j)$ ayant la même somme d'indices $(i+j)$ ne peuvent jamais être traités simultanément dans la cellule $q = i + j$.

Le calcul de la convolution se fait en utilisant deux flots de données. Les $x_i$ se propagent de gauche à droite du réseau sans que leur valeur ne soit modifiée. Les $y_i$ se propagent à la même vitesse mais en sens inverse. Leur calcul s'effectue par accumulation successive du résultat de chaque cellule élémentaire. La cellule élémentaire $q$ effectue la multiplication du $x_i$ entrant avec le coefficient correspondant $a_q$ et le cumule au résultat calculé par la cellule précédente $q-1$. La dernière cellule du réseau fournit le résultat $y_i$.

### 4.4.2 Description de la cellule élémentaire

Comme décrit précédemment la cellule élémentaire doit effectuer deux opérations simples : la multiplication et l'accumulation, d'où son nom de cellule MAC (*Multiply And Accumulate*) ; son fonctionnement est décrit par l'équation architecturale (4.2). Les explications permettant de comprendre le fonctionnement global sont données, plus loin, dans le paragraphe 4.4.3.

$$Cellule(q) \rightarrow \begin{cases} (R^x{}_q)_{t+1} & = & (R^x{}_{q+1})_t \\ (R^y{}_q)_{t+1} & = & (R^y{}_{q-1})_t + a_q \cdot (R^x{}_{q+1})_t \end{cases} \tag{4.2}$$

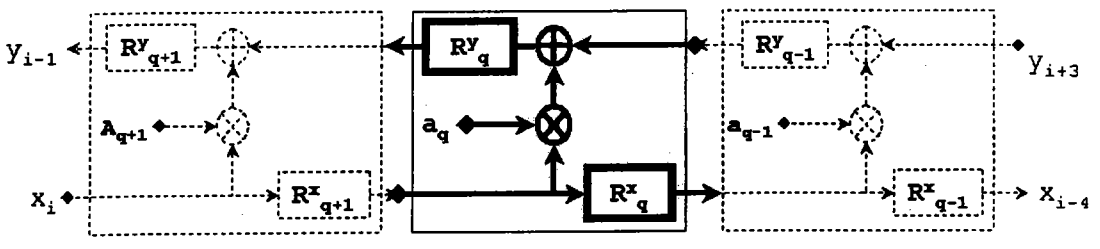



FIG. 4.1 – *Architecture interne de la cellule MAC.*

La description architecturale de la cellule MAC est représentée sur la figure 4.1. La complexité combinatoire de cette dernière est celle de la mise en cascade d'un multiplieur et d'un additionneur. Dans notre cas, le domaine de travail se situe dans le corps de Galois $CG(2)$ à deux éléments binaires $\{0,1\}$, ce qui ramène la complexité combinatoire de la cellule MAC à la mise en cascade d'une porte AND et d'une porte XOR. Vu que deux flots de données la traversent en sens inverse, la cellule est dite bidirectionnelle.

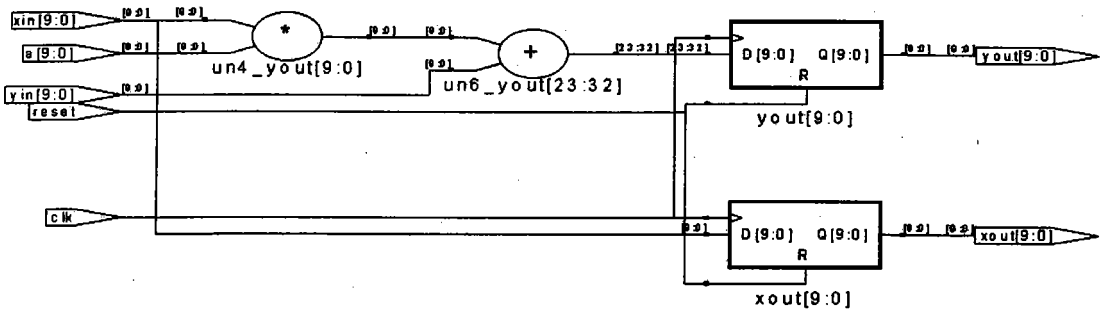



FIG. 4.2 – *Synthèse de la cellule MAC.*

La cellule MAC (fig. 4.2) à été décrite en VHDL sous MAX+plus II. Pour faciliter l'interprétation des résultats, le modèle (figure 4.4) a été décrit de la manière suivante :

- trois bus de 10 bits, $xin$, $yin$ et $a$ qui correspondent aux entrées des signaux $x_i$, $y_i$ et du coefficient $a_q$ ;
- deux bus de 10 bits $yout$ et $xout$ qui correspondent aux sorties de la cellules MAC ;
- une entrée $clk$ pour l'horloge $clk_{sys}$ et une entrée de réinitialisation $reset$.

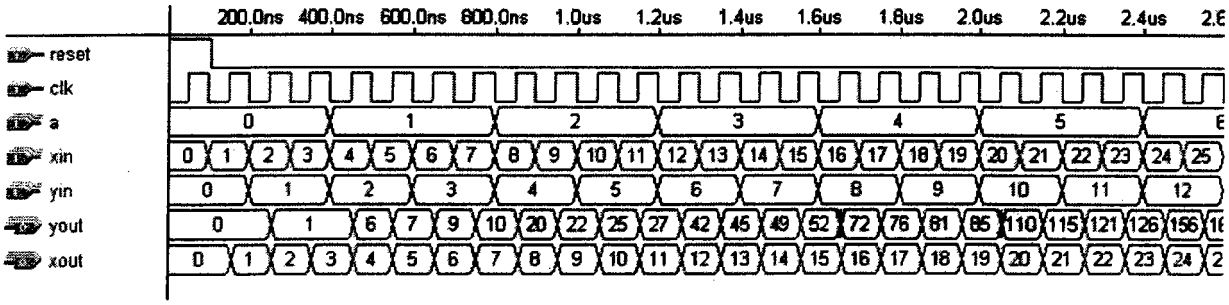



FIG. 4.3 – *Simulation de la cellule MAC.*

La figure 4.3 présente la simulation de la cellule MAC. On peut vérifier le bon fonctionnement de la cellule, en prenant par exemple, les valeurs en entrée de la cellule aux instants respectifs t =1 $\mu s$

(équ. (4.3)) et 2.2 $\mu s$ (équ. (4.4)).

$$\text{à t=1.0 } \mu s \mapsto \left\{ \begin{array}{rcl} x_{in} & = & 10 \\ y_{in} & = & 5 \\ a & = & 2 \end{array} \right\} \Rightarrow \left\{ \begin{array}{rcl} x_{out} & = & 10 \\ y_{out} & = & 10 \times 2 + 5 = 25 \end{array} \right. \tag{4.3}$$

$$\text{à t=2.2 } \mu s \mapsto \left\{ \begin{array}{rcl} x_{in} & = & 22 \\ y_{in} & = & 11 \\ a & = & 5 \end{array} \right\} \Rightarrow \left\{ \begin{array}{rcl} x_{out} & = & 22 \\ y_{out} & = & 22 \times 5 + 11 = 121 \end{array} \right. \tag{4.4}$$

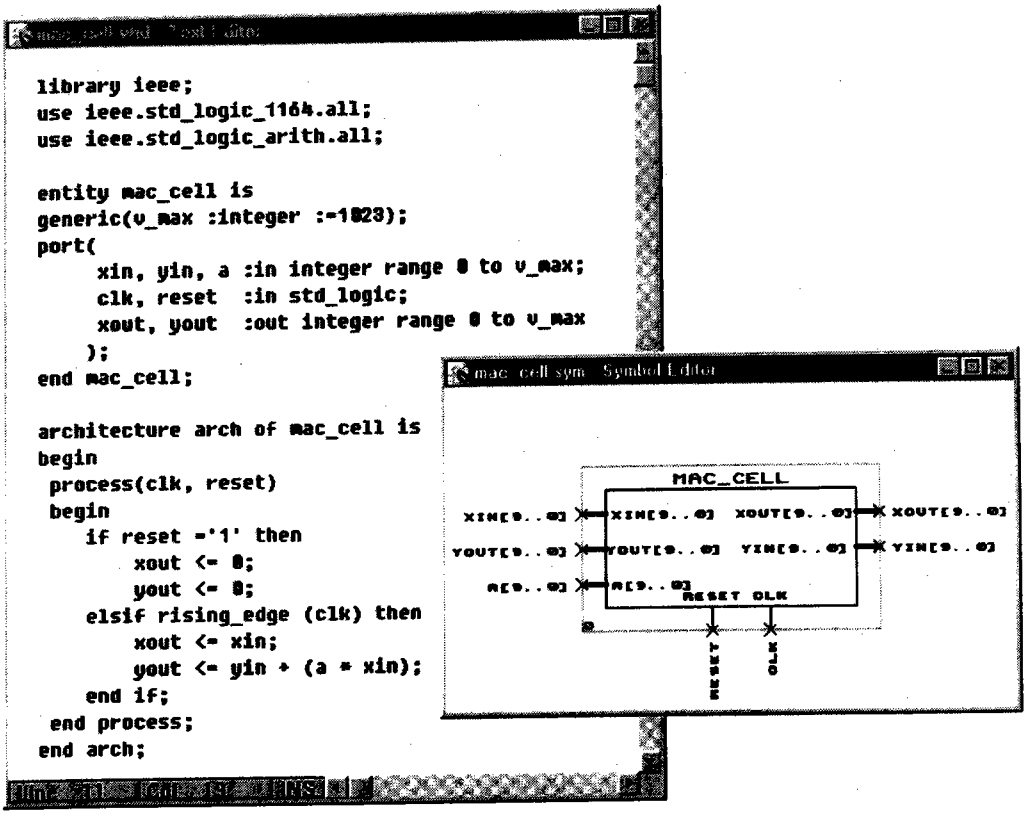



FIG. 4.4 – *Description VHDL de la cellule MAC.*

### 4.4.3 Description du réseau systolique

Après la définition de la cellule élémentaire MAC, la construction du réseau systolique permettant le calcul de la convolution est simple : il suffit d'interconnecter des cellules élémentaires entre elles. Le nombre de cellules depend uniquement du degré $q$ du polynôme générateur $P(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_q x^q$. Dans le cas présent, $q + 1$ cellules seront nécessaires pour l'implantation du réseau. La figure 4.5 représente un réseau de convolution non récursive correspondant à un polynôme

générateur $P(x)$ de degré $q = 3$.

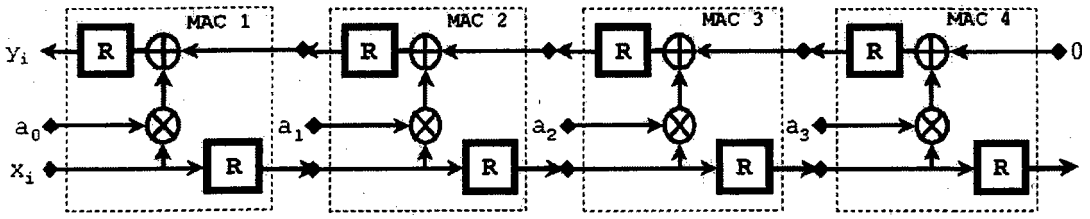



FIG. 4.5 – *Réseau de convolution non récursive bidirectionelle.*

Seule la cellule MAC1 est connectée à l'unité de contrôle. Les deux ports de la cellule MAC4 sont fictifs, les $y_i$ entrant sont des 0. Pour que le calcul de la convolution se fasse correctement, les $x_i$ sont injectés dans le réseau un cycle d'horloge sur deux. Par conséquent, un résultat est fourni un cycle d'horloge sur deux.

Le tableau 4.1 permet de suivre, étape par étape, la propagation des $y_i$ à travers le réseau : chaque cellule élémentaire calcule un résultat partiel $y_{MACi}$ et l'ajoute au précédent. Ainsi, le résultat final est généré par la première cellule (MAC1 dans l'exemple).

| Cycle | $y_i$ | $x_i$ | MAC1 | MAC2 | MAC3 | MAC4 |
|---|---|---|---|---|---|---|
| t | - | $x_0$ | 0 | 0 | 0 | 0 |
| t+1 | - | - | $y_0 = x_0a_0$ | 0 | 0 | 0 |
| t+2 | $y_0$ | $x_1$ | - | $x_0a_1$ | 0 | 0 |
| t+3 | - | - | $y_1 = x_1a_0 + x_0a_1$ | - | $x_0a_2$ | 0 |
| t+4 | $y_1$ | $x_2$ | - | $x_1a_1 + x_0a_2$ | - | $x_0a_3$ |
| t+5 | - | - | $y_2 = x_2a_0 + x_1a_1 + x_0a_2$ | - | $x_1a_2 + x_0a_3$ | - |
| t+6 | $y_2$ | $x_3$ | - | $x_2a_1 + x_1a_2 + x_0a_3$ | - | $x_1a_3$ |
| t+7 | - | - | $y_3 = x_3a_0 + x_2a_1 + x_1a_2 + x_0a_3$ | - | $x_2a_2 + x_1a_3$ | - |
| t+8 | $y_3$ | $x_4$ | - | $x_3a_1 + x_2a_2 + x_1a_3$ | - | $x_2a_3$ |
| t+9 | - | - | $y_4 = x_4a_0 + x_3a_1 + x_2a_2 + x_1a_3$ | - | $x_3a_2 + x_2a_3$ | - |
| t+10 | $y_4$ | $x_5$ | - | $x_4a_1 + x_3a_2 + x_2a_3$ | - | $x_3a_3$ |

TAB. 4.1 – *Propagation des données dans le réseau.*

Pour le bon fonctionnement du circuit, les $x_i$ doivent être injectés un cycle sur deux. Cette restriction réduit l'efficacité du circuit d'un facteur deux. Une première constatation permet de dire que l'architecture systolique consomme, pour le calcul d'un même produit de convolution, deux fois plus de cycles d'horloge qu'une architecture classique (équ. (4.5)).

$$\text{Nbre } Cycle_{systolique} = 2 \times \text{Nbre } Cycle_{classique} \tag{4.5}$$

Le principal inconvénient d'une architecture classique (de type MTO) réside dans le fait que le chemin critique est proportionnelle au degré $q$ de son polynôme générateur[3]. Alors que dans une architecture systolique, quel que soit ce degré, le chemin critique du réseau reste celui de la cellule élémentaire

3. Plus précisément le chemin critique dépend du nombre de coefficients non nuls du polynôme générateur.

MAC. Comme le montre la figure 4.1, le chemin critique de la cellule MAC est équivalent, dans le cas binaire, à celui de portes AND et XOR mises en série. Par conséquent, pour de grandes valeurs de $q$, le réseau systolique est plus performant (équ. (4.6)).

$$q \rightarrow grand \Rightarrow 2 \times T_{sys} < T_{clas} \tag{4.6}$$

où $T_{sys}$ et $T_{clas}$ représentent les périodes d'horloge (durée d'un cycle) des architectures systolique et classique respectivement.

Le réseau systolique (fig. 4.6) simulé correspond au polynôme générateur $P(x) = 2 + x + 5x^2 + 3x^3$, ($a_0 = 2$, $a_1 = 1$, $a_2 = 5$, $a_3 = 3$). Comme précisé précédemment, la sortie $x_{out}$ est fictive et l'entrée $y_{in}$ mise à zéro « GND[4] ».

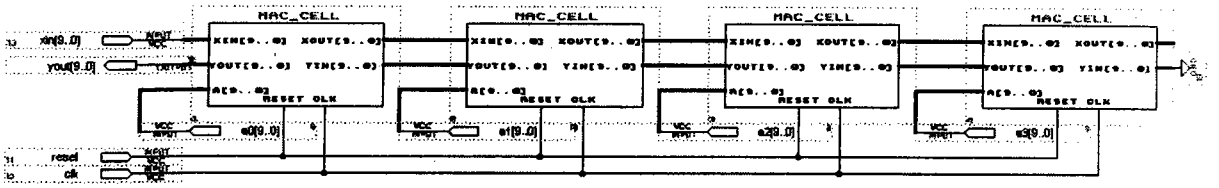



FIG. 4.6 – *Implantation du réseau bidirectionnel de la convolution non récursive.*

La simulation du réseau bidirectionel (fig. 4.7) permet de mettre en évidence la réduction de l'efficacité du circuit par deux. Le « × » représente l'état indéterminé.

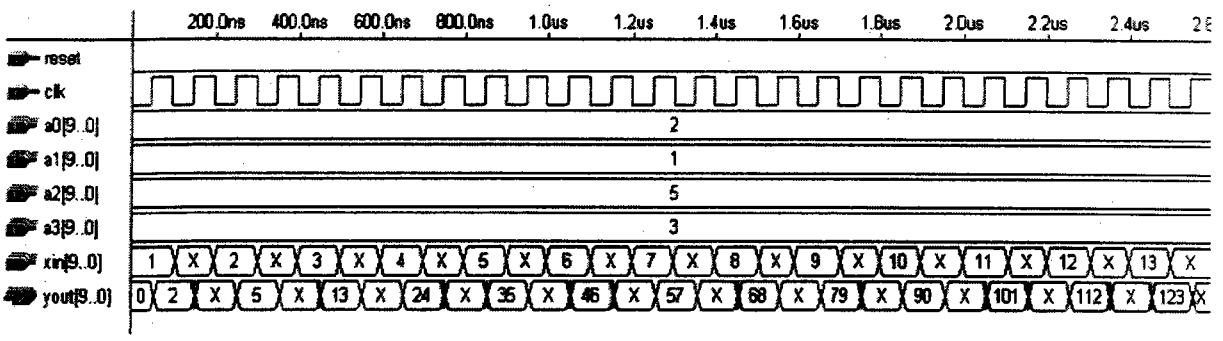



FIG. 4.7 – *Simulation du réseau bidirectionnel de la convolution non récursive.*

L'ensemble d'équations (4.7) met en évidence les premières itérations du calcul de la convolution.

4. GND : *ground* (masse).

Ceci permet de vérifier que le réseau fonctionne correctement :

$$\left\| \begin{array}{rcl} y_0 & = & (2)\cdot 1+(1)\cdot 0+(5)\cdot 0+(3)\cdot 0=2 \\ y_1 & = & (2)\cdot 2+(1)\cdot 1+(5)\cdot 0+(3)\cdot 0=5 \\ y_2 & = & (2)\cdot 3+(1)\cdot 2+(5)\cdot 1+(3)\cdot 0=13 \\ y_3 & = & (2)\cdot 4+(1)\cdot 3+(5)\cdot 2+(3)\cdot 1=24 \\ y_4 & = & (2)\cdot 5+(1)\cdot 4+(5)\cdot 3+(3)\cdot 2=35 \end{array} \right. \tag{4.7}$$

## 4.5 Convolution récursive

La convolution récursive peut être vue comme un filtre à réponse impulsionelle infinie. Son mode opératoire est décrit par l'équation fonctionnelle (4.8).

$$\left\{ \begin{array}{rcl} y_{-1} & = & x_0 \\ y_i & = & \sum_{j=0}^{q} a_j \cdot y_{i-j-1} = a_0 y_{i-1} + a_1 y_{i-2} + \cdots + a_q y_{i-q-1} \\ & & i \in \{0,1,\cdots\} \end{array} \right. \tag{4.8}$$

Le réseau systolique qui réalise la convolution récursive est simple à construire. En effet, à partir du réseau de convolution non récursive, il suffit de rajouter une cellule pour réinjecter le résultat dans le réseau afin qu'il soit pris en compte dans le calcul.

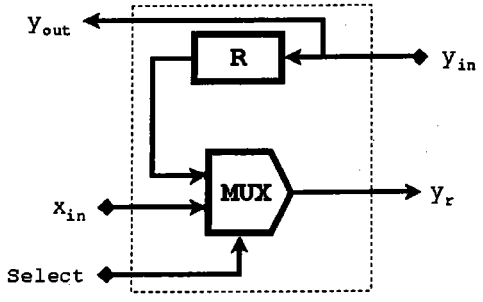



FIG. 4.8 – *Cellule de retard.*

La cellule de retard implantée (fig. 4.8) a pour rôle :

- d'amorcer le réseau de convolution ;
- de réinjecter les $y_i$ calculés dans le réseau.

Ces deux actions sont réalisées à l'aide d'un multiplexeur à deux entrées qui, en fonction de la commande `select` permet, soit de sélectionner l'entrée extérieure $x_{in}$, soit de réinjecter les $y_i$ calculés dans le réseau. Dans le premier cas, le réseau effectue la convolution non récursive. Dans le deuxième cas, c'est la convolution récursive qui est réalisée. Dans ce mode de fonctionnement toutefois, l'entrée $x_{in}$ sera maintenue durant un cycle d'horloge afin d'amorcer le circuit.

### 4.5.1 Description du réseau systolique

La figure 4.9 représente le réseau de convolution correspondant à un polynôme générateur de degré $q = 3$. Il présente la même efficacité que le réseau calculant la convolution non récursive, c'est à dire un résultat tous les deux cycles d'horloge.

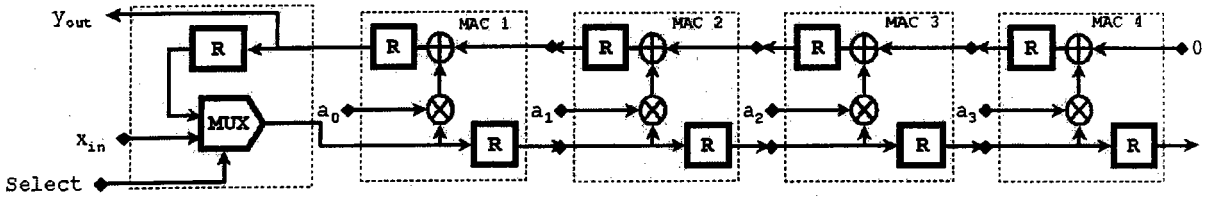



FIG. 4.9 – *Réseau bidirectionnel de la convolution récursive.*

Le tableau 4.2 visualise les premières itérations du calcul de la convolution récursive. Le réseau fonctionne de la même manière que celui décrit précédemment (calcul de la convolution non récursive). $E_{MUX}$ dans le tableau représente l'entrée sélectionnée du multiplexeur.

| Cycle | $y_i$ | $E_{MUX}$ | MAC1 | MAC2 | MAC3 | MAC4 |
|---|---|---|---|---|---|---|
| t | - | $x_0$ | 0 | 0 | 0 | 0 |
| t+1 | - | - | $y_0 = x_0a_0$ | 0 | 0 | 0 |
| t+2 | $y_0$ | $y_0$ | - | $x_0a_1$ | 0 | 0 |
| t+3 | - | - | $y_1 = y_0a_0 + x_0a_1$ | - | $x_0a_2$ | 0 |
| t+4 | $y_1$ | $y_1$ | - | $y_0a_1 + x_0a_2$ | - | $x_0a_3$ |
| t+5 | - | - | $y_2 = y_1a_0 + y_0a_1 + x_0a_2$ | - | $y_0a_2 + x_0a_3$ | - |
| t+6 | $y_2$ | $y_2$ | - | $y_1a_1 + y_0a_2 + x_0a_3$ | - | $y_0a_3$ |
| t+7 | - | - | $y_3 = y_2a_0 + y_1a_1 + y_0a_2 + x_0a_3$ | - | $y_1a_2 + y_0a_3$ | - |
| t+8 | $y_3$ | $y_3$ | - | $y_2a_1 + y_1a_2 + y_0a_3$ | - | $y_1a_3$ |
| t+9 | - | - | $y_4 = y_3a_0 + y_2a_1 + y_1a_2 + y_0a_3$ | - | $y_2a_2 + y_1a_3$ | - |
| t+10 | $y_4$ | $y_4$ | - | $y_3a_1 + y_2a_2 + y_1a_3$ | - | $y_2a_3$ |

TAB. 4.2 – *Propagation des données dans le réseau.*

– : signifie aucune valeur introduite dans le circuit.

Le réseau systolique simulé est représenté dans la figure 4.10. La commande `select` de la cellule de retard $DELAY_CELL$ permet de sélectionner le mode de fonctionnement du réseau (voir tableau 4.3).

| select | $E_{MUX}$ | type de convolution |
|---|---|---|
| 0 | $x_{in}$ | non récursive |
| 1 | $y_{in}$ | récursive |

TAB. 4.3 – *Configuration du réseau bidirectionnel.*

La simulation du réseau (fig. 4.11) montre les deux modes de fonctionnement. En mode récursif, le circuit effectue le calcul de l'équation (4.9).

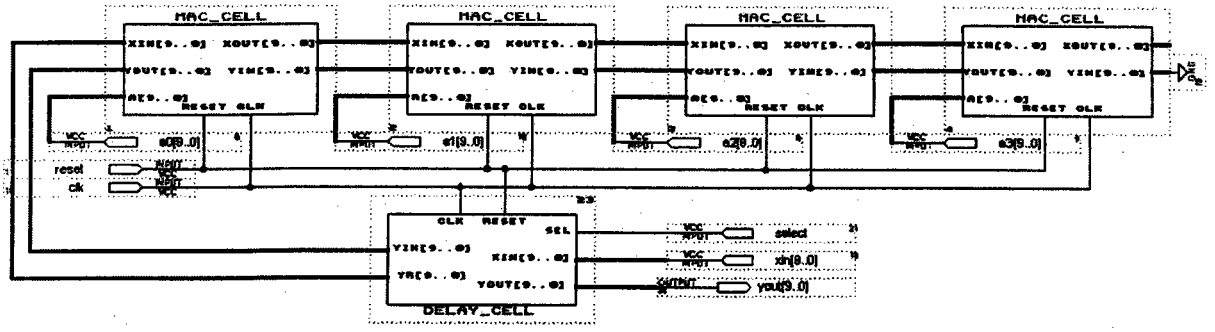



FIG. 4.10 – *Implantation du réseau bidirectionnel de la convolution récursive.*

Le polynôme générateur choisi est $P(x) = 2 + x + 5x^2 + 3x^3$.

$$\begin{cases} y_{-1} &= x_0 \\ y_i &= a_0 y_{i-1} + a_1 y_{i-2} + a_2 y_{i-3} + a_3 y_{i-4} \\ & \quad i \in \{0,1,\cdots\} \end{cases} \tag{4.9}$$

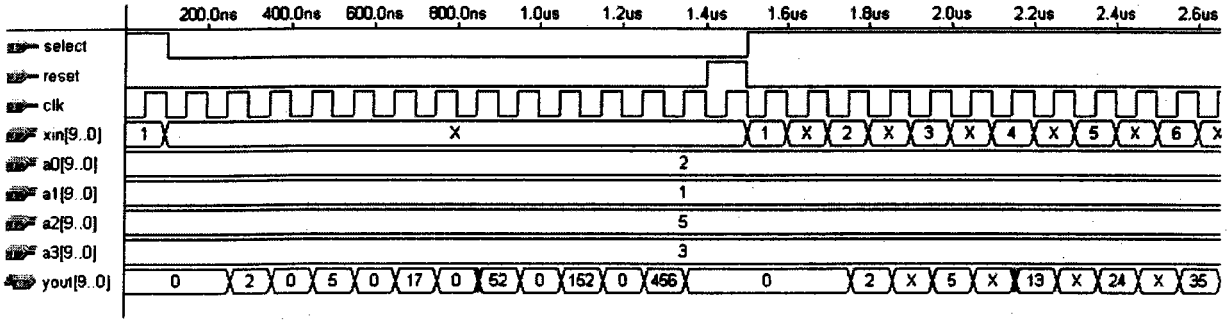



FIG. 4.11 – *Simulation du réseau bidirectionnel de la convolution récursive & non récursive.*

En effectuant quelques itérations on obtient :

$$\begin{array}{lcl} y_{-1} &=& x_0 = 1 \\ y_0 &=& (2)\cdot 1 + (1)\cdot 0 + (5)\cdot 0 + (3)\cdot 0 = 2 \\ y_1 &=& (2)\cdot 2 + (1)\cdot 1 + (5)\cdot 0 + (3)\cdot 0 = 5 \\ y_2 &=& (2)\cdot 5 + (1)\cdot 2 + (5)\cdot 1 + (3)\cdot 0 = 17 \\ y_3 &=& (2)\cdot 17 + (1)\cdot 5 + (5)\cdot 2 + (3)\cdot 1 = 52 \\ y_4 &=& (2)\cdot 52 + (1)\cdot 17 + (5)\cdot 5 + (3)\cdot 2 = 152 \end{array} \tag{4.10}$$

## 4.6 Convolution récursive générale

La convolution récursive générale est décrite par l'équation fonctionnelle (4.11).

$$y_i = \sum_{j=0}^{q} a_j \cdot x_{i-j} + \sum_{j=0}^{h} g_j \cdot y_{i-j-1} \tag{4.11}$$

où $P(x) = a_0 + a_1x + a_2x^2 + \cdots + a_qx^q$ et $G(x) = g_0 + g_1x + g_2x^2 + \cdots + g_hx^h$.
où $(q+1)$ représente la longueur de la fenêtre de convolution et $(h+1)$ l'ordre de récurrence.

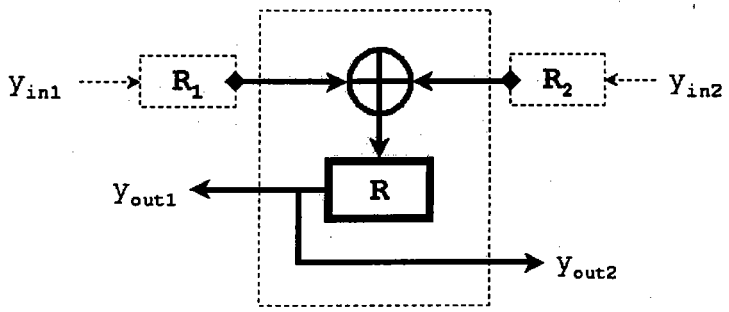



FIG. 4.12 – *Cellule d'addition.*

Dans ce cas-ci le réseau de convolution est composé de 2 blocs, $B_q$ et $B_h$, et d'une cellule, décrite par l'équation architecturale (4.12), pour additionner les deux convolutions partielles (fig. 4.12). Les blocs sont eux mêmes constitués de $(q+1)$ et $(h+1)$ cellules MAC, respectivement.

$$(R)_{t+1} = (R_1)_t + (R_2)_t \tag{4.12}$$

Le réseau bidirectionnel de la convolution générale est représenté sur la figure 4.13. Comme pour les réseaux précédents, les $y_i$ sont calculés un cycle d'horloge sur deux.

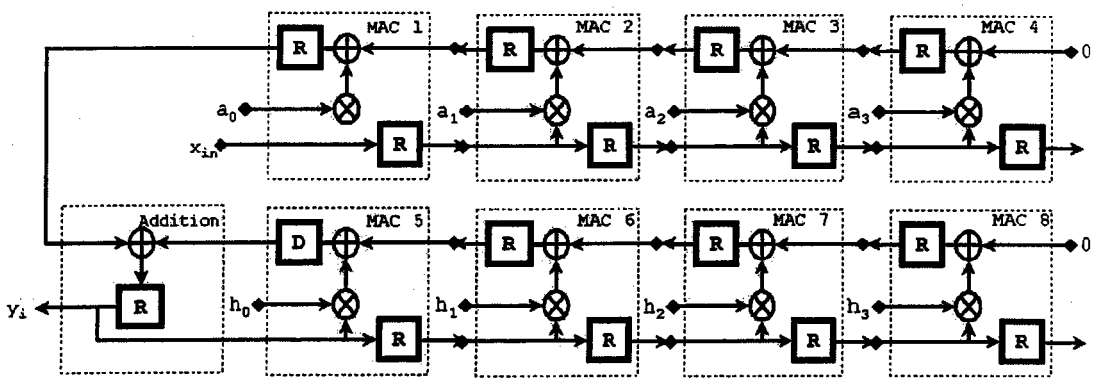



FIG. 4.13 – *Réseau bidirectionnel de la convolution récursive générale.*

La figure 4.14 représente l'implantation d'un réseau de convolution pour $q = h = 3$, ce qui correspond aux polynômes générateurs $P(x) = a_0 + a_1x + a_2x^2 + a_3x^3$ et $G(x) = g_0 + g_1x + g_2x^2 + g_3x^3$. La figure 4.15 présente la simulation du réseau pour des valeurs de coefficients suivantes : ($a_0$=2, $a_1$=1, $a_2$=5, $a_3$=3) et ($g_0$=2, $g_1$=1, $g_2$=3, $g_3$=4).

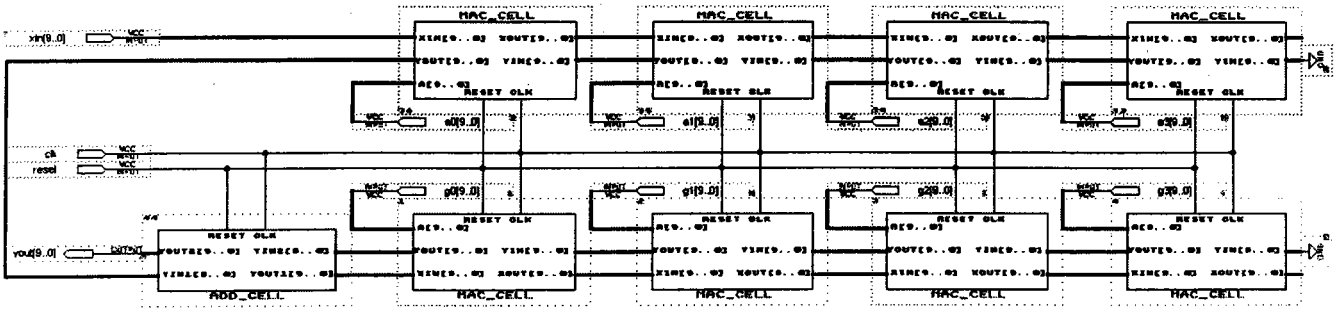



FIG. 4.14 – *Implantation du réseau bidirectionnel de la convolution récursive générale.*

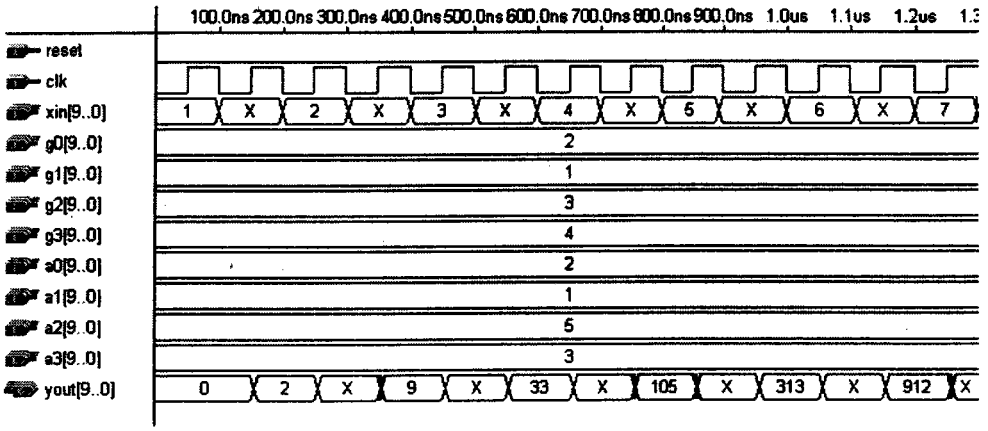



FIG. 4.15 – *Simulation du réseau bidirectionnel de la convolution récursive générale.*

Le groupe d'équations (4.13) décrit les premières itérations du calcul de la convolution.

$$
\begin{array}{rcrcrcrcrcl}
y_0 & = & (2)\cdot 1 & + & (1)\cdot 0 & + & (5)\cdot 0 & + & (3)\cdot 0 & & \\
 & + & (2)\cdot 0 & + & (1)\cdot 0 & + & (3)\cdot 0 & + & (4)\cdot 0 & = & 2 \\
y_1 & = & (2)\cdot 2 & + & (1)\cdot 1 & + & (5)\cdot 0 & + & (3)\cdot 0 & & \\
 & + & (2)\cdot 2 & + & (1)\cdot 0 & + & (3)\cdot 0 & + & (4)\cdot 0 & = & 9 \\
y_2 & = & (2)\cdot 3 & + & (1)\cdot 2 & + & (5)\cdot 1 & + & (3)\cdot 0 & & \\
 & + & (2)\cdot 9 & + & (1)\cdot 2 & + & (3)\cdot 0 & + & (4)\cdot 0 & = & 33 \\
y_3 & = & (2)\cdot 4 & + & (1)\cdot 3 & + & (5)\cdot 2 & + & (3)\cdot 1 & & \\
 & + & (2)\cdot 33 & + & (1)\cdot 9 & + & (3)\cdot 2 & + & (4)\cdot 0 & = & 105 \\
y_4 & = & (2)\cdot 5 & + & (1)\cdot 4 & + & (5)\cdot 3 & + & (3)\cdot 2 & & \\
 & + & (2)\cdot 105 & + & (1)\cdot 33 & + & (3)\cdot 9 & + & (4)\cdot 2 & = & 313
\end{array}
\qquad (4.13)
$$

## 4.7 Optimisation des structures systoliques

Les réseaux systoliques présentés, permettant le calcul de la convolution, ont l'avantage majeur d'être simples à mettre en œuvre. Ils ne sont cependant pas optimaux. En effet, pour une fréquence de fonctionnement donnée $f_{clk_{sys}}$, le circuit a un débit réel de $1/2 \cdot f_{clk_{sys}}$, soit un résultat pour deux cycles d'horloge. On parle d'efficacité 2 pour ce type de circuit.

Dans ce qui suit, deux méthodes sont présentées qui permettent d'optimiser le fonctionnement de ces réseaux. Le but est d'obtenir des réseaux d'efficacité 1 (un résultat chaque cycle d'horloge). Le prix à payer dans ce cas, est l'augmentation de la surface utilisée lors de l'implantation. Une première approche consiste à changer la direction de propagation de telle façon qu'elle soit la même pour $x_i$ et $y_i$. Le réseau ainsi obtenu est dit unidirectionnel, en référence au sens de propagation des deux flots $x_i$ et $y_i$. Une autre méthode utilise deux sous-réseaux bidirectionnels, calculant chacun une convolution partielle. De cette manière, la somme des calculs partiels fournit le résultat final de la convolution. Cette approche est appelée « diviser pour régner ».

### 4.7.1 Réseau systolique unidirectionnel pour la convolution

Le réseau unidirectionnel est construit à partir des mêmes cellules MAC utilisées dans le réseau bidirectionnel. Du point de vue fonctionnel les cellules utilisées dans les deux types de réseaux sont identiques. Cependant deux modifications sont apportées :

- afin d'obtenir deux flots de données de même sens de propagation, les entrées et les sorties de la cellule MAC sont orientées dans la même direction (fig. 4.16) ;

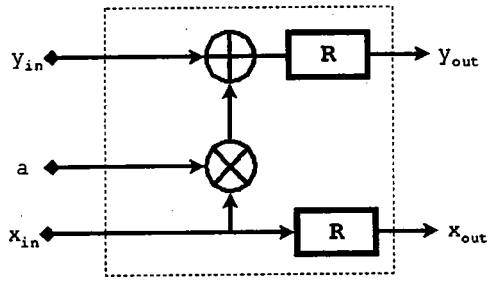



FIG. 4.16 – *Cellule MAC unidirectionnelle.*

- pour que le réseau calcule correctement la convolution, des cellules de retard (registres) sont insérées dans le parcours des $x_i$ entre les cellules MAC (fig. 4.17). De cette manière, les $y_i$ circuleront dans le réseau deux fois plus vite que les $x_i$.

Le réseau de convolution simulé est représenté sur la figure 4.18. Afin de comparer les performances des deux réseaux, bidirectionnel et unidirectionnel, les coefficients de convolution $a_i$ choisis

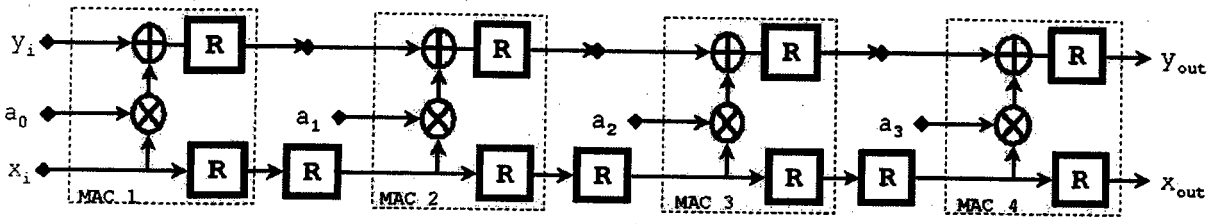



FIG. 4.17 – *Réseau unidirectionnel de convolution non récursive.*

sont identiques aux précédents. Du point de vue de la surface, 3 registres supplémentaires sont utilisés pour réaliser le réseau unidirectionnel. Dans le cas général, pour $q$ cellules MAC, $q-1$ registres supplémentaires sont requis.

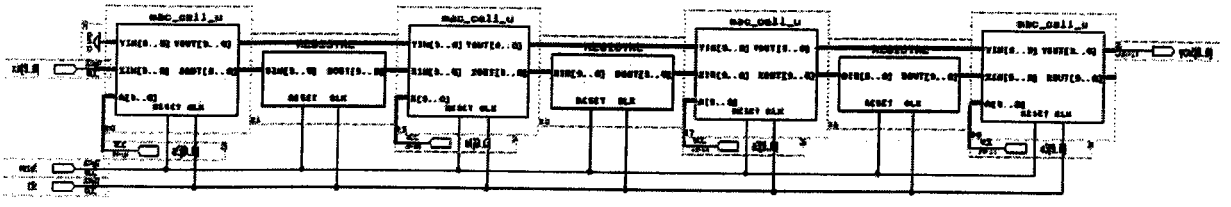


FIG. 4.18 – *Implantation du réseau unidirectionnel de convolution non récursive.*

La figure 4.19 présente les résultats de simulation du réseau unidirectionnel. On voit bien que le réseau fournit un résultat $y_i$ à chaque cycle d'horloge. Le réseau est donc d'efficacité 1. On constate par contre, l'apparition d'un temps de latence dont la durée est égale à 4 cycles d'horloge. Cette latence est due au cumul des temps de propagation des $y_i$ à travers les 4 cellules MAC du réseau. Plus généralement, pour un réseau unidirectionnel de $q$ cellules élémentaires, la durée du temps de latence est de $q$ cycles d'horloge.

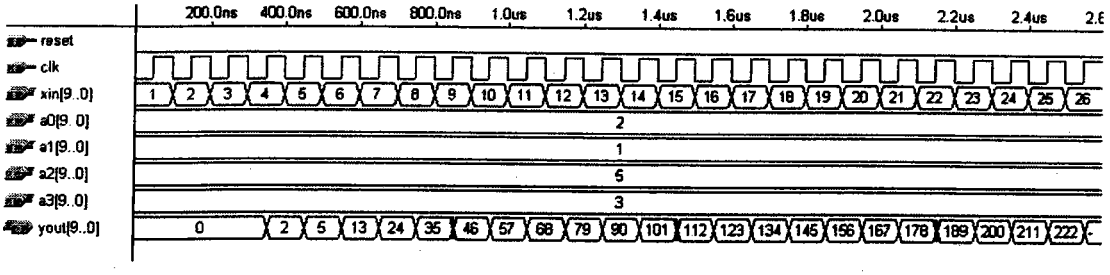



FIG. 4.19 – *Simulation du réseau unidirectionnel de convolution non récursive.*

Le tableau 4.4 donne de manière succincte un comparatif des deux réseaux systoliques bidirectionnels et unidirectionnels pour le calcul de la convolution non récursive ; $q$ représente le degré du polynôme générateur $P(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_q x^q$.

Les réseaux unidirectionnels ne sont pas adaptés au calcul de la convolution récursive [123].

| Réseau | Efficacité | Latence | Nbre de MAC | Nbre de registre |
|---|---|---|---|---|
| Bidirectionnel | 2 | 0 | q+1 | 0 |
| Unidirectionel | 1 | q+1 | q+1 | q |

TAB. 4.4 – *Comparatif des réseaux bidirectionnels et unidirectionnels.*

### 4.7.2 Technique « Diviser pour Régner »

L'approche « Diviser pour Régner » (DR) consiste à diviser un problème donné, quand celui-ci le permet, en $n$ sous-problèmes que l'on résout séparément. La solution du problème initial est obtenue en combinant les $n$ solutions partielles. A l'inverse du réseau unidirectionnel, cette technique est adaptable aux deux types de convolutions, la non récursive et la récursive.

**Application à la convolution non récursive**

L'équation (4.1) peut être décomposée de la manière suivante, (équ. (4.14)) :

$$\begin{aligned} y_i &= \sum_{j=0}^{q} x_{i-j} \cdot a_j \\ &= \sum_{u=0}^{q_1} x_{i-2u} \cdot a_{2u} + \sum_{v=0}^{q_2} x_{i-2v+1} \cdot a_{2v+1} = {y_i}^1 + {y_i}^2 \end{aligned} \tag{4.14}$$

Les valeurs de $q_1$ et $q_2$ sont déterminées suivant la relation (4.15), en fonction de :

$$\begin{cases} q \text{ pair} & \Rightarrow \quad q_1 = \frac{q}{2} \quad et \quad q_2 = \frac{q}{2} - 1 \\ q \text{ impair} & \Rightarrow \quad q_1 = \frac{q-1}{2} \quad et \quad q_2 = \frac{q-1}{2} \end{cases} \tag{4.15}$$

D'après la relation (4.14), le réseau systolique $B$, représenté sur la figure 4.20 est composé :

- d'un sous réseau $B_1$ qui calcule les valeurs partielles ${y_i}^1$ ;
- d'un sous réseau $B_2$ qui calcule les valeurs partielles ${y_i}^2$ ;
- d'une cellule qui calcule $y_i = {y_i}^1 + {y_i}^2$.

Le réseau systolique d'efficacité 1 est obtenu en combinant les deux sous-réseaux bidirectionnels $B1$ et $B2$ calculant chacun une convolution partielle. L'augmentation de la surface, par rapport au réseau bidirectionnel, est celle d'un additionneur.

Le réseau $B$ simulé, représenté sur la figure 4.21, a pour polynôme générateur $P(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$.

- on déduit les valeurs de $q1$ et $q2$ d'après (4.15) :

$$q = 3 \Rightarrow q \text{ est impair} \Rightarrow \begin{cases} q_1 = \frac{q-1}{2} = 1, \\ q_2 = \frac{q-1}{2} = 1. \end{cases} \tag{4.16}$$

– d'après (4.14), on construit les réseaux $B_1$ et $B_2$ :

$$\begin{aligned} (B_1) : {y_i}^1 &= \sum_{u=0}^{q_1=1} x_{i-2u} \cdot a_{2u} = x_i \cdot a_0 + x_{i-2} \cdot a_2 \\ (B_2) : {y_i}^2 &= \sum_{v=0}^{q_2=1} x_{i-2v+1} \cdot a_{2v+1} = x_{i-1} \cdot a_1 + x_{i-3} \cdot a_3 \end{aligned} \tag{4.17}$$

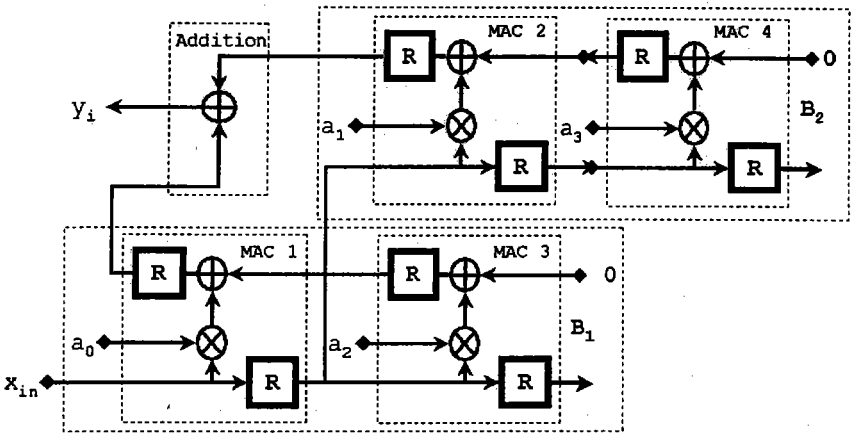



FIG. 4.20 – *Réseau de convolution non récursive DR.*

De l'équation (4.17), on déduit que $B_1$ et $B_2$ sont constitués chacun de 2 cellules MAC. Les résultats de simulation du réseau de la figure 4.21 sont donnés sur la figure 4.22. On voit bien que le réseau est d'efficacité 1 et que, de plus, cette technique n'introduit aucun temps de latence.

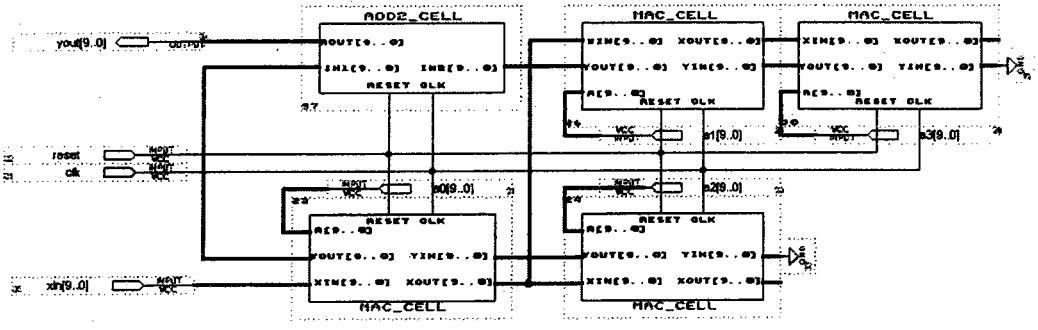



FIG. 4.21 – *Implantation du réseau de convolution non récursive DR.*

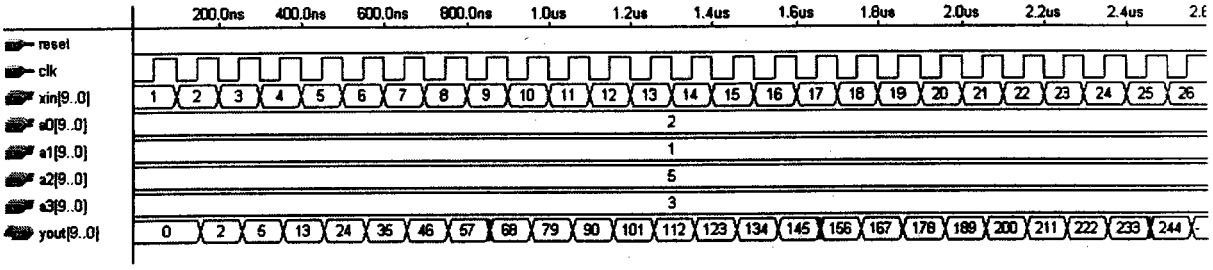



FIG. 4.22 – *Simulation du réseau de convolution non récursive DR.*

### Application à la convolution récursive

Comme pour la convolution non récursive, la technique DR s'applique à la convolution récursive. L'équation (4.9) est divisée de la manière suivante :

$$\begin{aligned} y_i &= \sum_{j=0}^{q} a_j \cdot y_{i-j-1} = a_0 \cdot y_{i-1} + a_1 \cdot y_{i-2} + \cdots + a_q \cdot y_{i-q-1} \\ &= \sum_{u=0}^{q_1} a_{2u} \cdot y_{i-(2u)-1} + \sum_{v=0}^{q_2} a_{2v+1} \cdot y_{i-(2v+1)-1} = {y_i}^1 + {y_i}^2 \\ &\text{avec } y_{-1} = x_0 \end{aligned} \tag{4.18}$$

La seule modification par rapport au réseau unidirectionnel est l'addition de la cellule de rebouclage (fig. 4.23).

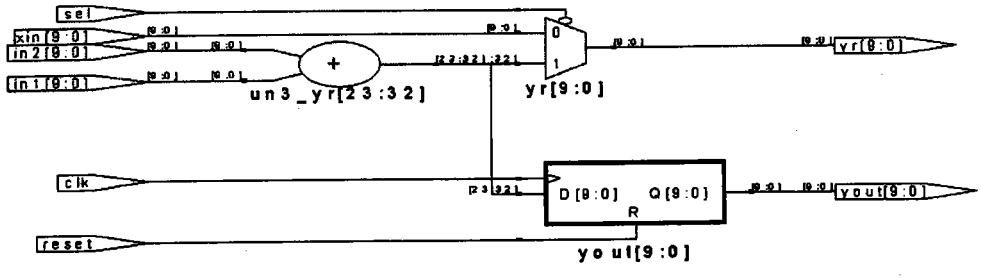



FIG. 4.23 – *Synthèse de la cellule de rebouclage DR.*

La cellule de rebouclage fournit la sortie $y_i = {y_i}^1 + {y_i}^2$. Elle permet aussi l'amorçage du circuit dans la phase d'initialisation, où les $y_i$ sont directement injectés dans le réseau. La simulation du circuit est représentée sur la figure 4.25, l'apparition d'une latence qui d'un cycle d'horloge est uniquement due à l'introduction du registre de sortie *yout* comme le montre la figure 4.23.

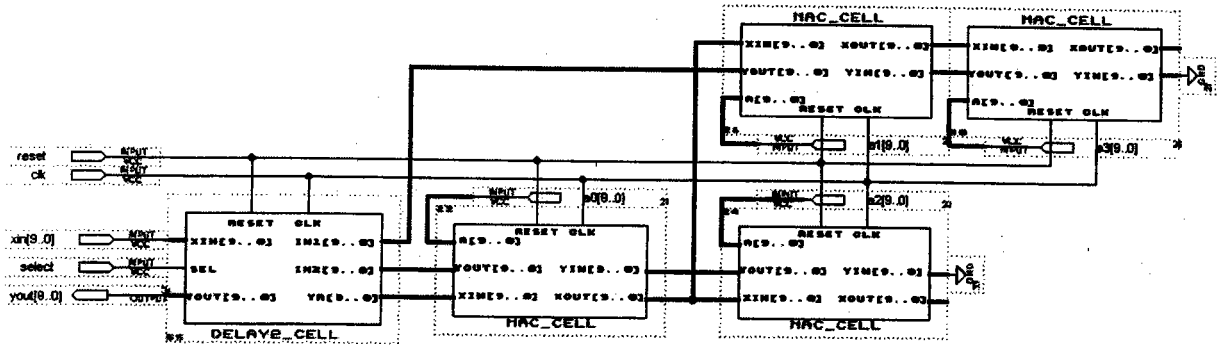



FIG. 4.24 – *Implantation du réseau de convolution récursive DR.*

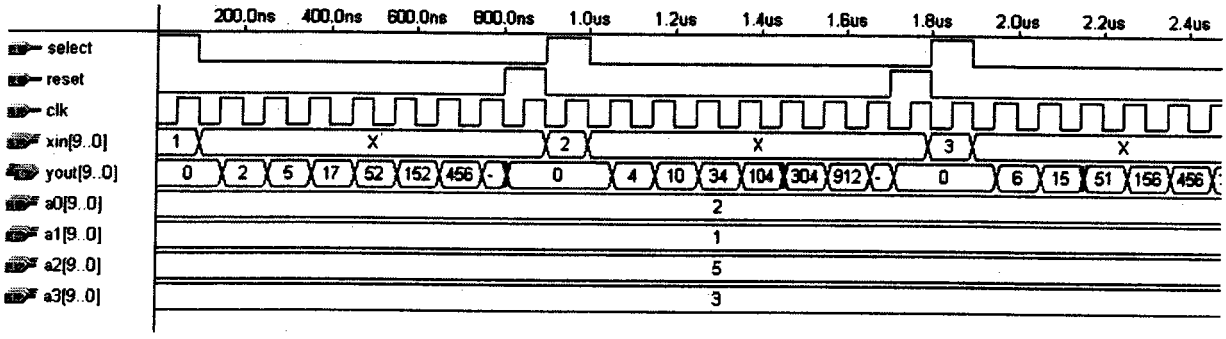



FIG. 4.25 – *Simulation du réseau de convolution récursive DR.*

### Application à la convolution récursive générale

De la même façon que pour la convolution récursive, l'équation (4.14) est appliquée à chacun des blocs $B_q$ et $B_h$ du réseau bidirectionnel (fig. 4.13).

$$\begin{aligned} y_i &= \sum_{j=0}^{q} a_j \cdot x_{i-j} + \sum_{j=0}^{h} g_j \cdot y_{i-j-1} \\ &= \left( \sum_{u=0}^{q_1} a_{2u} \cdot x_{i-2u} + \sum_{v=0}^{q_2} a_{2v+1} \cdot x_{i-2v+1} \right) \\ &+ \left( \sum_{u=0}^{h_1} g_{2u} \cdot y_{i-(2u)-1} + \sum_{v=0}^{h_2} g_{2v+1} \cdot y_{i-(2v+1)-1} \right) \\ &= \left( {y_i}^1 + {y_i}^2 \right) + \left( {z_i}^1 + {z_i}^2 \right) \end{aligned} \tag{4.19}$$

Les valeurs de $q_1$, $q_2$, $h_1$ et $h_2$ sont déterminées suivant la relation (4.15).

De la relation (4.19), nous pouvons déduire que le réseau est composé de 4 blocs (fig. 4.26) :

$$
\begin{aligned}
(B_q{}^1) &: \sum_{u=0}^{q_1} a_{2u} \cdot x_{i-2u} = y_i{}^1 \\
(B_q{}^2) &: \sum_{v=0}^{q_2} a_{2v+1} \cdot x_{i-2v+1} = y_i{}^2 \\
(B_h{}^1) &: \sum_{u=0}^{h_1} g_{2u} \cdot y_{i-(2u)-1} = z_i{}^1 \\
(B_h{}^2) &: \sum_{v=0}^{h_2} g_{2v+1} \cdot y_{i-(2v+1)-1} = z_i{}^2.
\end{aligned}
$$

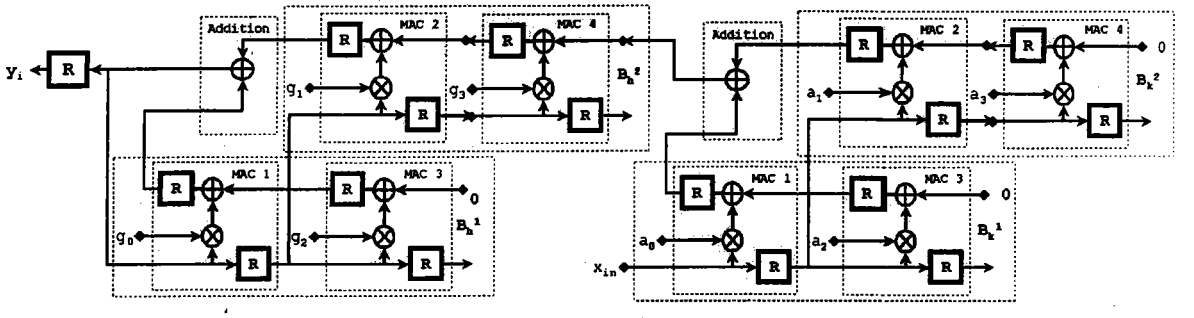



FIG. 4.26 – *Réseau de convolution récursive générale DR.*

Le réseau implanté est représenté sur la figure 4.27. Les polynômes choisis $P(x)$et $G(x)$ sont de degré 3.

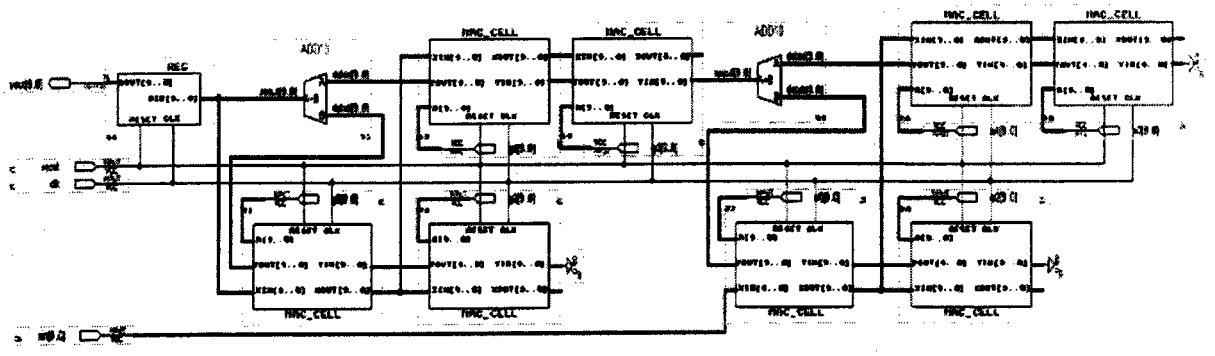


FIG. 4.27 – *Implantation du réseau de convolution récursive générale DR.*

Les courbes de simulation de la figure 4.28 représentent la convolution récursive générale dans les cas des polynômes générateurs ($P_1(x)$, $G_1(x)$) et ($P_2(x)$, $G_2(x)$), et des intervalles de temps

$[0\ \mu\ s,\ 1.1\ \mu\ s]$ et $[1.2\ \mu\ s,\ 2.3\ \mu\ s]$, respectivement.

$$\textbf{Intervalle [0 } \mu s\textbf{, 1.1 } \mu s\textbf{]} \quad \mapsto \quad \begin{cases} P_1(x) &= 2 + 1x + 5x^2 + 3x^3 \\ G_1(x) &= 2 + 1x + 3x^2 + 4x^3 \end{cases}$$

$$\textbf{Intervalle [1.2 } \mu s\textbf{, 2.3 } \mu s\textbf{]} \quad \mapsto \quad \begin{cases} P_2(x) &= 2 + 1x + 2x^2 + 4x^3 \\ G_2(x) &= 1 + 0x + 3x^2 + 2x^3 \end{cases}$$

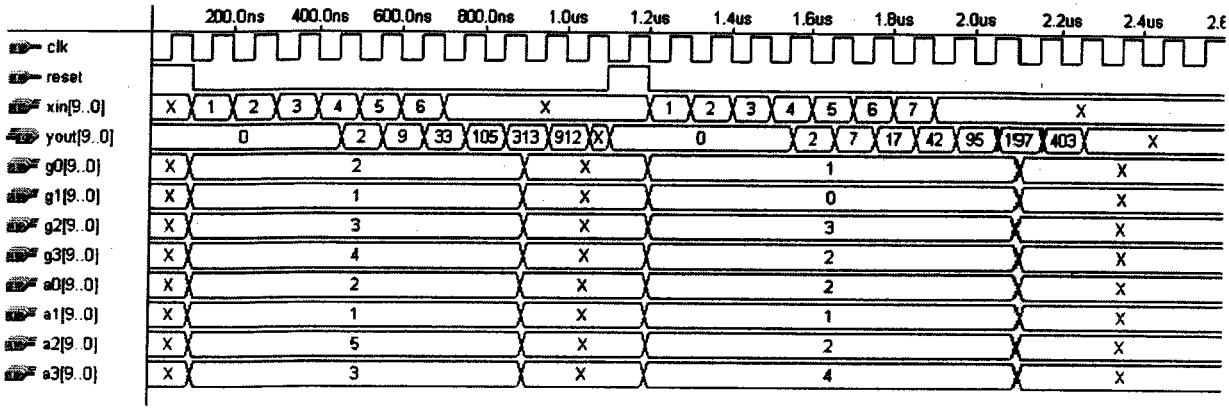



FIG. 4.28 – *Simulation du réseau de convolution récursive générale DR.*

## 4.8 Résistance aux pannes

De par leur conception, les réseaux systoliques peuvent être facilement modifiés pour les rendre résistants aux éventuelles pannes survenant au niveau des cellules élémentaires. Le principe est de concevoir chaque cellule élémentaire avec des éléments redondants permettant de la contourner en cas de défaillance. De cette manière, en cas de panne, il suffira de détecter la cellule défectueuse afin de la contourner. Bien sûr, dans ce cas le réseau sera reconfiguré pour fonctionner sans les cellules défectueuses. Dans le cas de la convolution, le réseau résultant de taille inférieure à l'originel calculera la convolution sur des polynômes de moindre degré.

### 4.8.1 Réseaux linéaires unidirectionnels

Dans le cas du réseau linéaire unidirectionnel la solution est simple : il suffit d'associer à chaque processeur élémentaire deux registres supplémentaires permettant de le contourner en cas de défaillance (fig. 4.29).

Dans le cas par exemple d'une panne dans un réseau de convolution de 5 cellules élémentaires

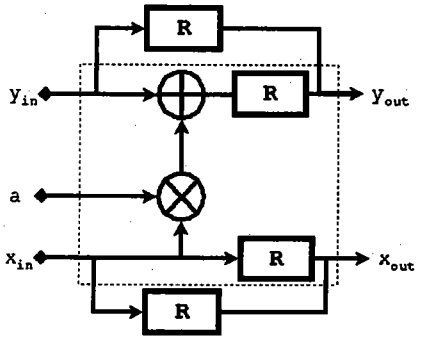



FIG. 4.29 – *Cellule MAC unidirectionnelle avec cellules de contournement.*

($q = 4$), celui-ci se comportera comme un réseau de convolution de 4 cellules, avec un retard d'un cycle d'horloge.

Il est nécessaire d'autre part, de charger les processeurs encore valides avec les poids adéquats, ce qui peut se faire en adjoignant une entrée de chargement dans chaque cellule et en sélectionnant les coefficients dans une mémoire préchargée.

De manière générale, un réseau linéaire unidirectionnel de $n$ cellules dont $q$ sont défaillantes peut fonctionner comme un réseau de $(n-q)$ cellules à condition de pouvoir détecter les cellules valides et les reconfigurer. Bien sûr, dans ce cas, les résultats du calcul seront fournis avec un temps de latence de $q$ cycles d'horloge.

### 4.8.2 Réseaux linéaires bidirectionnels

En insérant des registres de contournement, les flots de données $x_i$ et $y_i$ qui circulent en sens inverse ne sont plus synchronisés. Une solution consiste à diviser le taux d'activité du système par un facteur $(q+1)$, où $q$ représente le nombre de cellules susceptibles de tomber en panne. Cette solution dégrade cependant considérablement l'efficacité du réseau en la ramenant à une valeur de $2(q+1)$.

## 4.9 Réseaux orthogonaux pour le calcul de la convolution

La topologie orthogonale permet le calcul de la convolution de deux polynômes $A(x)$ et $B(x)$ chacun de degré $q$. Le réseau systolique dans ce cas est composé de $(q+1) \times (q+1)$ cellules élémentaires et les $(2q+1)$ résultats de la convolution sont donnés après $(q+1)$ cycles d'horloge.

Trois flots de données circulent dans le réseau : les coefficients $a_i$ et $b_i$ correspondant aux polynômes respectifs $A(x)$ et $B(x)$ et les résultats partiels $y_i$ de la convolution.

Chaque processeur élémentaire (fig. 4.30) est connecté à son voisin par ligne, par colonne et par

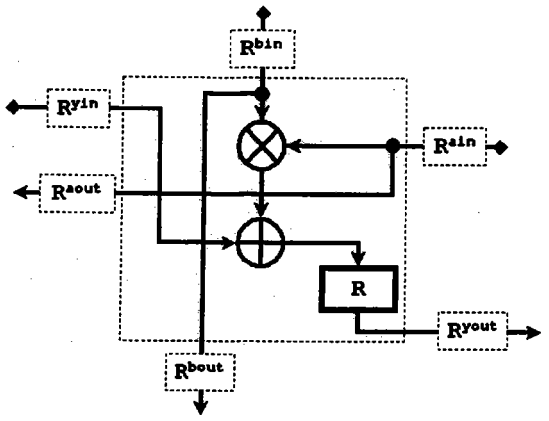



FIG. 4.30 – *Architecture interne de la cellule élémentaire dans un réseau orthogonal.*

diagonale. Son fonctionnement est décrit par l'équation architecturale (4.20).

$$\begin{cases} (R^{aout})_{t+1} = (R^{ain})_t \\ (R^{bout})_{t+1} = (R^{bin})_t \\ (R^{yout})_{t+1} = (R^{ain})_t \cdot (R^{bin})_t + (R^{yin})_t \end{cases} \tag{4.20}$$

Le réseau orthogonal de la figure 4.31 est constitué de $4 \times 4$ processeurs élémentaires et permet le calcul de la convolution de deux polynômes de degré $q = 3$. Les entrées des coefficients $a_i$ et $b_i$ de chaque polynôme ainsi que les sorties $y_i$ sont situées aux extrémités du réseau.

Les $2q + 1$ résultats de la convolution sont disponibles après un temps de latence de $q + 1$ cycles d'horloge. Les repères $d_i$ et les $f_i$ sur les courbes de simulation représentent respectivement les débuts du calcul de la convolution et les instants de disponibilité des résultats. On peut constater que le nombre de cycles d'horloge entre un $d_i$ et le $f_i$ correspondant est de quatre, ce qui correspond au temps de latence. Le principal avantage de cette structure est qu'elle permet de traiter les données en flot continu.

Les résultats de la simulation du réseau pour une implantation sur un FPGA de type Flex10KE d'Altera sont présentés sur la figure 4.32.

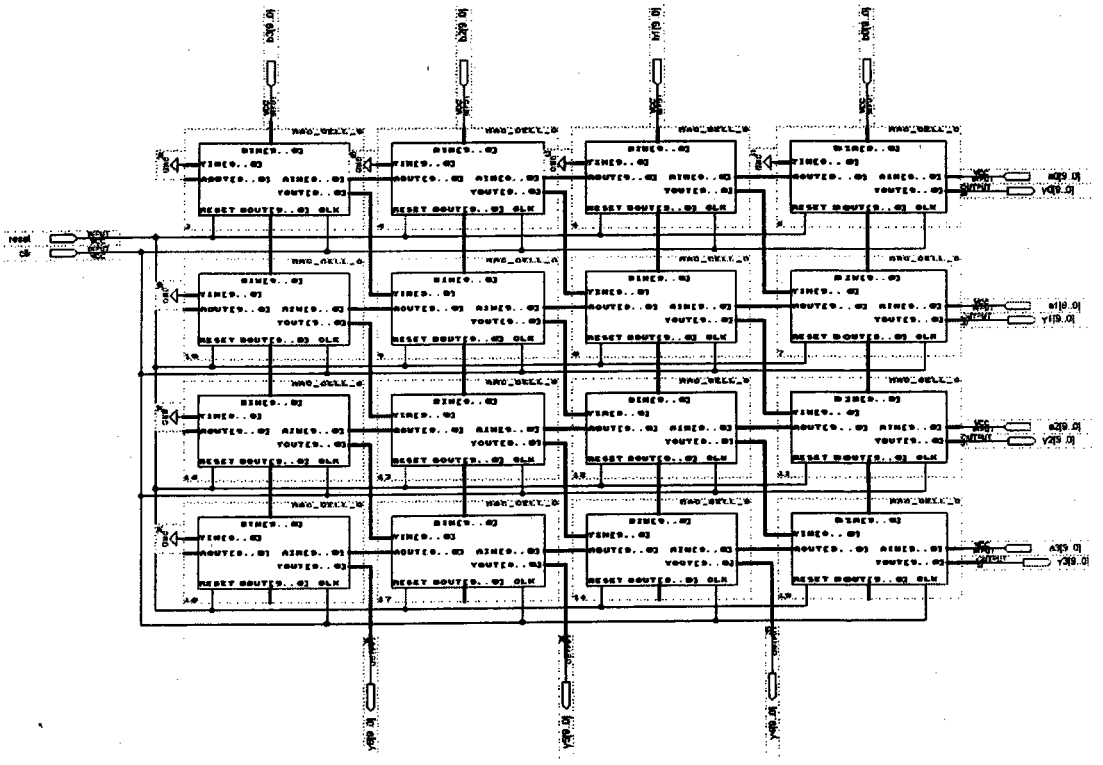


FIG. 4.31 – *Réseau orthogonal pour le calcul de la convolution non récursive.*

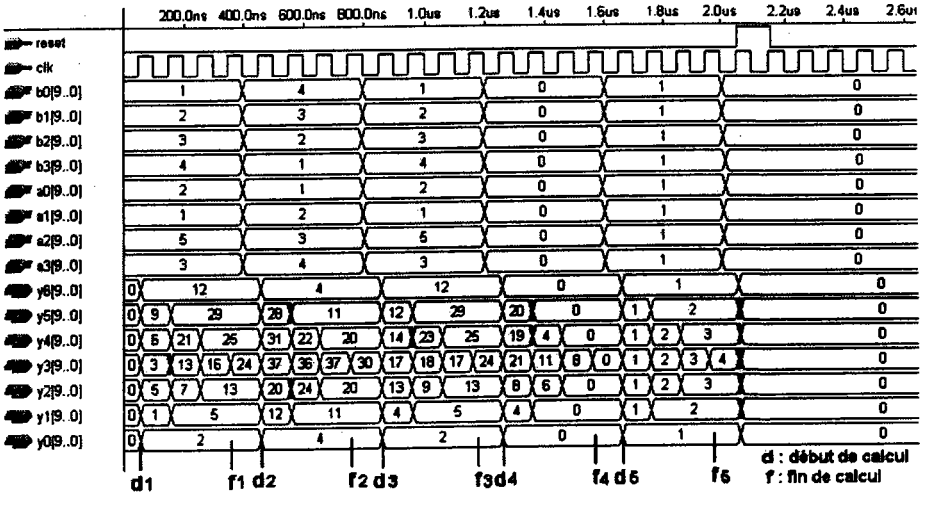



FIG. 4.32 – *Simulation du réseau orthogonal.*

## 4.10 Conclusion

Ce chapitre a présenté les différentes structures systoliques destinées au calcul de la convolution. Nous avons commencé par les réseaux systoliques bidirectionnels qui sont d'efficacité 2. Ensuite deux solutions architecturales qui permettent d'améliorer l'efficacité des réseaux bidirectionnels ont été abordées. La première solution - réseaux unidirectionnels - modifie le sens de propagation des données et introduit un temps de latence, dépendant du nombre de cellules élémentaires mises en cascade, dû aux registres de synchronisation additionnels. La deuxième solution - diviser pour régner - réduit le temps de latence en divisant la structure complète en deux sous-réseaux dont les résultats sont additionnés. Le tableau 4.5 résume les différentes architectures de réseaux systoliques permettant le calcul des différents types de convolution décrits dans ce chapitre.

| Réseau | Convolution | Efficacité | Nombre de cellules |
|---|---|---|---|
| Bidirectionnel | Non récursive | 2 | (q+1) $PE_b$ |
| | Récursive | 2 | (q+1) $PE_b$+ $C_r$ |
| | Récursive générale | 2 | (q+1) $PE_b$+(h+1) $PE_b$+ $C_a$ |
| Unidirectionnel | Non récursive | 1 | (q+1) $PE_u$+q $R_g$ |
| Diviser pour régner | Non récursive | 1 | (q+1) $PE_b$+ $C_a$ |
| | Récursive | 1 | (q+1) $PE_b$+ $C_r$ |
| | Récursive générale | 1 | (q+1) $PE_b$+(h+1) $PE_b$+2 $C_a$ |
| Orthogonal | Non récursive | 1 | (2q+1) $PE_o$ |

TAB. 4.5 – *Comparatif des réseaux systoliques pour le calcul de la convolution.*

avec
- $PE_b$ : Processeur élementaire bidirectionnel
- $PE_u$ : Processeur élementaire unidirectionnel
- $PE_o$ : Processeur élementaire orthogonal
- $C_r$ : Cellule de retard
- $C_a$ : Cellule d'addition
- $R_g$ : Registre

Cette étude a permis de mettre en évidence la simplicité de mise en œuvre des réseaux systoliques due à leurs structures régulières et à leur chemin critique équivalent à celui d'une cellule élémentaire. Leur capacité à résister aux pannes, d'être reconfigurés en ligne et d'effectuer les traitements en flot continu, les rendent particulièrement intéressants. Cependant, le traitement des données s'effectuant de manière sérielle, ces structures permettent d'atteindre, dans le meilleur des cas, un débit équivalent à la fréquence maximale de fonctionnement du composant cible. D'où la nécessité de développer une architecture parallèle permettant un traitement plus rapide. Il est évident qu'une augmentation effective de la vitesse dépend fortement de la fréquence de fonctionnement globale qui doit être maintenue à des valeurs proches de celles du circuit série pour que le parallélisme soit efficace. De plus le maintien d'une certaine régularité structurelle est souhaitable pour faciliter la mise en œuvre du circuit codeur. C'est ce que l'on se propose de faire dans le chapitre suivant.

# Chapitre 5

# Codeur convolutif et turbo-codeur haut débit

Dans ce chapitre, nous proposons une architecture parallèle avancée pour les codes convolutifs. La première partie introduit le principe général de fonctionnement du codeur rapide. La deuxième partie présente la mise en forme des équations fonctionnelles de deux architectures série de codeurs convolutifs : l'architecture *Many To One* (MTO) et l'architecture *One To Many* (OTM). A partir des deux descriptions fonctionnelles des codeurs MTO et OTM, seront déduites les équations qui régissent le fonctionnement du codeur parallèle. A partir des équations résultantes, l'architecture du codeur rapide sera décrite. Afin d'évaluer les performances du codeur haut débit, un modèle générique décrit en VHDL au niveau d'abstraction RTL (*Register Transfert Level*) a été synthétisé sur des FPGA de type Flex10KE d'Altera. La troisième partie enfin, sera dédiée à la présentation et à l'interprétation des résultats expérimentaux de l'implantation, en termes de débit (nombre de bits traités par seconde) et de surface consommée (nombre de cellules logiques utilisées).

**Mots clés :** codes convolutifs, architecture parallèle et pipeline, modélisation VHDL, implantation haut débit, FPGA.

## 5.1 Introduction

Avec une architecture synchrone fonctionnant à la fréquence d'horloge de $f_{clk}$ , l'approche série permet de traiter un bit par cycle d'horloge. Le débit $D_S$ du codeur vaut donc ($1\ bit \cdot f_{clk,ser}$). La parallélisation de l'architecture doit permettre de traiter $\wp$ bits par cycle d'horloge. Le débit $D_\wp$ est alors égal à ($\wp\ bits \cdot f_{clk,par}$). Pour avoir une augmentation effective du débit, il est souhaitable que la fréquence $f_{clk,par}$ de l'architecture parallèle soit aussi proche que possible :

$$f_{clk,par} \simeq f_{clk,ser} \tag{5.1}$$

Avant de poursuivre, nous introduisons, par un exemple simple, les notions de parallélisme combinatoire et pipeline.

- l'additionneur de la figure 5.1 permet d'effectuer l'addition de deux variables, la longueur de son chemin critique est $\ell_a$ ;

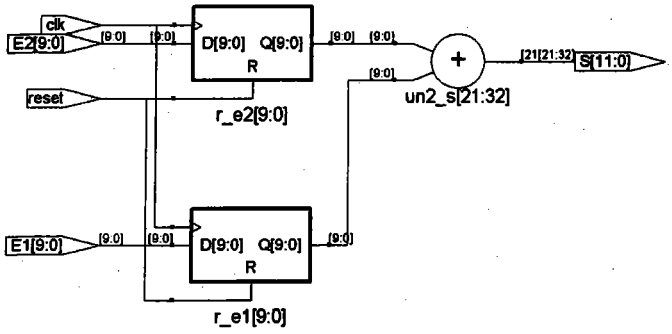



FIG. 5.1 – *Additionneur à 2 entrées.*

- la figure 5.2 représente une structure combinatoire d'un additionneur à 4 entrées, il permet de calculer en parallèle 4 variables. Cependant la longueur $\ell_c$ de son chemin critique est : $\ell_c > \ell_a$. Dans ce cas on parle d'implantation parallèle purement combinatoire ou de parallélisme spatial ;

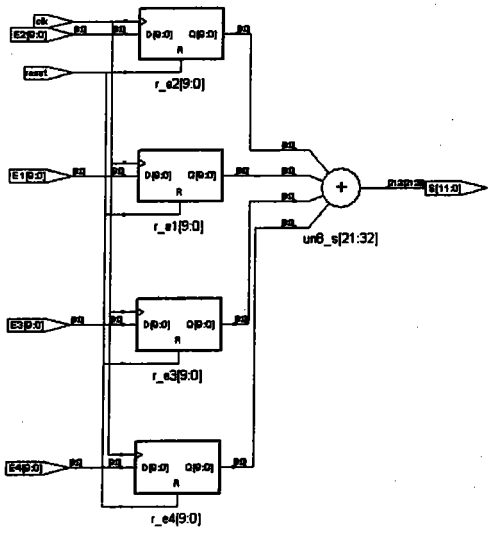


FIG. 5.2 – *Structure combinatoire d'un additionneur à 4 entrées.*

- la structure pipeline de l'additionneur représentée dans la figure 5.3 permet de calculer en parallèle 4 variables, la longueur $\ell_p$ de son chemin critique est : $\ell_p = \ell_a$. Dans ce cas on parle d'implantation parallèle pipeline ou de parallélisme temporel.

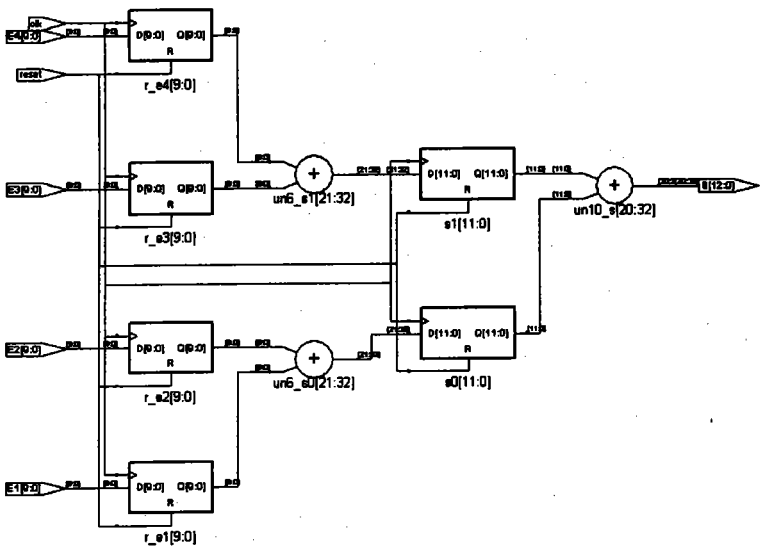



FIG. 5.3 – *Structure pipeline d'un additionneur à 4 entrées.*

Lors d'une implantation parallèle purement combinatoire, deux limitations sont à considérer:

- le chemin critique généré par le bloc combinatoire dégrade considérablement la fréquence de fonctionnement $f_{clk,par}$, d'une manière générale :

$$\wp \nearrow \Rightarrow f_{clk,par} \searrow \Rightarrow f_{clk,par} \ll f_{clk,ser} \tag{5.2}$$

- la surface occupée $S_o$ par le modèle parallèle augmente en fonction du degré de parallélisme.

$$\wp \nearrow \Rightarrow S_o \nearrow \tag{5.3}$$

Notre objectif étant d'augmenter le débit, la première contrainte rend une approche purement combinatoire inefficace dans notre cas. En effet, le principal objectif d'une parallélisation est d'augmenter le débit d'un circuit en augmentant le nombre de bits traités par cycle d'horloge.

L'idéal est d'avoir une augmentation proportionnelle du débit en fonction du degré de parallélisme (fig. 5.4, cas idéal). Malheureusement, dans le cas général la relation proportionnelle se dégrade rapidement après une certaine limite du degré de parallélisme $\wp_{limite}$ (fig. 5.4, approche combinatoire). Le débit ne pouvant augmenter de manière effective au delà de $\wp_{limite}$, le parallélisme combinatoire devient sans intérêt au delà de cette valeur.

Notre approche consiste à combiner l'approche combinatoire avec une technique pipeline afin de maximiser la valeur de $\wp_{limite}$ en minimisant d'une part, la longueur du chemin critique généré par le circuit combinatoire et d'autre part, en le rendant indépendant du degré de parallélisme. De cette manière, toute augmentation du degré de parallélisme $\wp$ entraînera automatiquement une augmentation du débit.

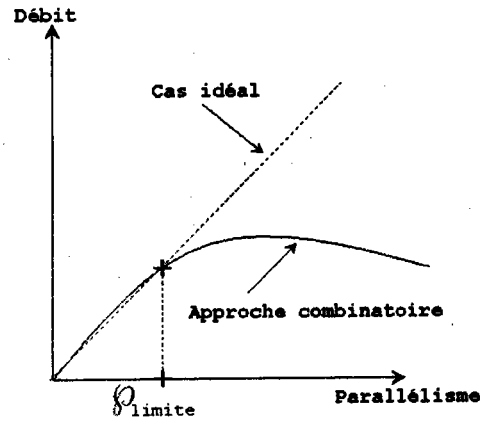



FIG. 5.4 – *Courbes caractéristiques du débit en fonction du degré de parallélisme.*

Du point de vue expérimental, quatre modèles de codeurs parallèles modélisés en VHDL ont été synthétisés sur FPGA. Deux architectures parallèles sont proposées pour chacune des variantes MTO et OTM du codeur.

$$CP_{mto} \rightarrow \begin{cases} CP1_{mto} \\ CP2_{mto} \end{cases} \text{ et } CP_{otm} \rightarrow \begin{cases} CP1_{otm} \\ CP2_{otm} \end{cases}$$

Après l'implantation, les performances de chaque codeur seront évaluées en fonction de deux critères directement liés aux paramètres intrinsèques du composant cible. Dans notre cas, la cible est un FPGA de type FLEX10KE d'Altera dont la fréquence maximale de fonctionnement est $f_{max} = 250\ MHz$. Par conséquent, l'évaluation du codeur se fera en fonction de :

- la fréquence de fonctionnement $f_{clk,par}$, qui permettra de calculer son débit :

$$D_{\wp} \text{ (unité)} = \wp \times f_{clk,par} \tag{5.4}$$

- la surface $S_o$ occupée, mesurée en nombre de cellules logiques utilisées dans le FPGA.

A partir de ces deux paramètres, les courbes de compromis ($D_{\wp}$, $S_o$) permettront de comparer les performances de chaque codeur parallèle. Ainsi, pour une surface occupée $S'_o$ le codeur le plus performant aura le débit le plus élevé. Vis versa, pour un débit $D'_{\wp}$ le meilleur codeur aura la plus petite surface dans le FPGA.

Après l'évaluation des performances du codeur convolutif rapide, la structure RSC du codeur sera implantée comme code constituant dans un turbo-code parallèle à deux niveaux.

## 5.2 Principe

Pour un codeur convolutif non systématique de rendement $1/n$, le calcul des bits de parité $(c_0, c_1, \cdots, c_n)_j = C_j$ correspondant à la sortie $C^{(j)}$ du codeur s'effectue à partir des deux paramètres $s_j$ et $d_j$ selon

l'équation (5.5), où $s_j$ représente l'état courant du codeur et $d_j$ le bit d'information présent à l'entrée.

$$C_j = F(s_j,d_j) \tag{5.5}$$

$F$ est une fonction qui dépend des polynômes générateurs correspondant à chaque sortie $C^{(j)}$ ($j = \{0,1,\cdots,n-1\}$). Soit un codeur à 4 états ($S_0$,$S_1$,$S_2$,$S_3$) dont le fonctionnement est décrit par le treillis de la figure (5.5). Comme vu dans le paragraphe 2.3.1, chaque nœud $S_{(j,i)}$ du treillis représente un état $S_j$ possible que peut prendre le codeur à l'instant $i$.

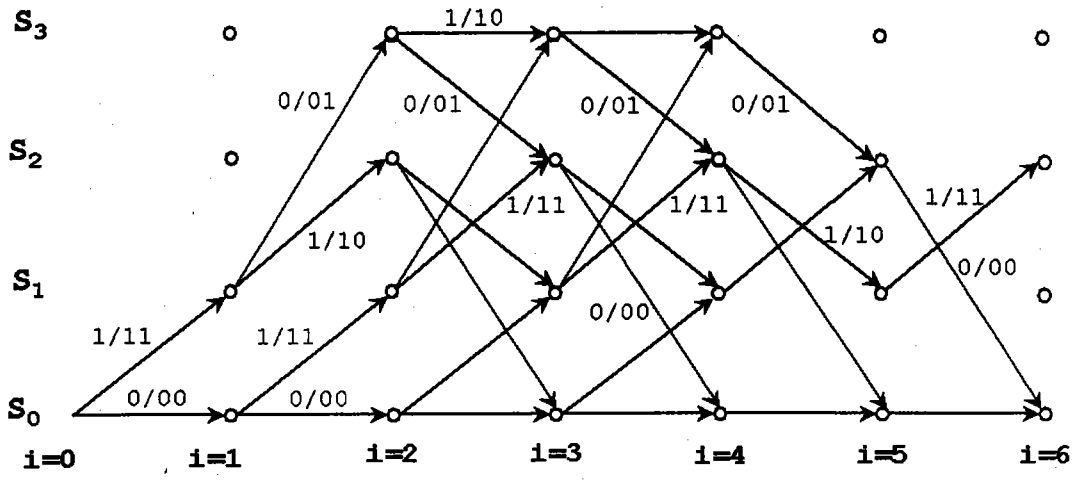



FIG. 5.5 – *Représentation en treillis du fonctionnement d'un codeur de rendement* $1/2$.

Si $(s_0,s_1,\cdots,s_{k-1})$ représente la séquence d'états du codeur correspondant à la lecture d'une trame d'information jusqu'à l'instant $k$, alors chaque état $s_i$ représente l'état $S_j$ du codeur au $i^{eme}$ cycle d'horloge (équ. (5.6)) :

$$S_{(j,i)} = s_i \tag{5.6}$$

En prenant par exemple, les états initiaux $s_0 = S_0$ et $i = 0$, et la séquence de données (101001), le chemin suivi par le codeur est donné en gras sur le treillis de la figure 5.6.

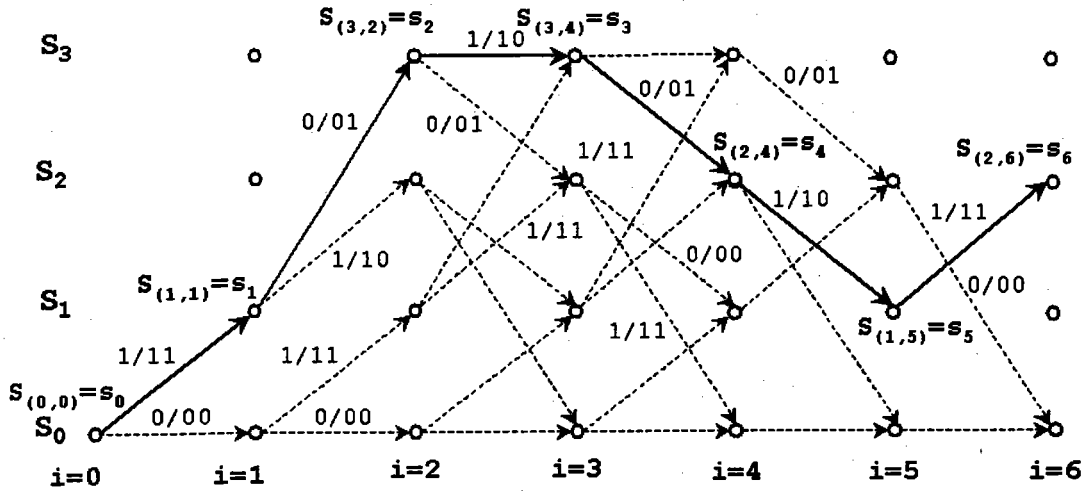



FIG. 5.6 – *Branche du treillis pour la séquence de donnée* (101001).

La séquence d'états correspondant à la séquence d'information (101001) est donnée dans le tableau 5.1, en appliquant l'équation (5.6).

| i | d | état |
|---|---|---|
| 0 | - | $S_{(0,0)} = s_0$ |
| 1 | 1 | $S_{(1,1)} = s_1$ |
| 2 | 0 | $S_{(3,2)} = s_2$ |
| 3 | 1 | $S_{(3,4)} = s_3$ |
| 4 | 0 | $S_{(2,4)} = s_4$ |
| 5 | 0 | $S_{(1,5)} = s_5$ |
| 6 | 0 | $S_{(2,6)} = s_5$ |
| ... | ... | ... |

TAB. 5.1 – *Etats du codeur générés par la séquence de données* (101001).

L'approche consiste à calculer des états particuliers du codeur qui seront appelés état anticipé (EA) et représentés par les valeurs $\Phi_k$. Chaque état anticipé $\Phi_k$ est l'état du codeur $s_{k\cdot\wp}$, (équ. (5.7)).

$$\Phi_k = s_{k\cdot\wp}, \quad k \in \{0,1,\cdots\} \tag{5.7}$$

A chaque cycle d'horloge, un nouvel état anticipé $\Phi_{k+1}$ est calculé à partir du précédent $\Phi_k$ et des $\wp$ nouveaux bits de données en entrée du codeur. Les états intermédiaires (EI) $(s_{k\cdot\wp+1}, s_{k\cdot\wp+2}, \cdots, s_{k\cdot\wp+\wp})$ entre deux états anticipés $\Phi_k$ et $\Phi_{k+1}$, seront calculés à l'aide d'une structure pipeline. La figure 5.7 montre, pour un degré de parallélisation $\wp = 3$ et une séquence en entrée $(101011\cdots)$, le début d'une branche du treillis correspondant aux états anticipés $\Phi_k$ .

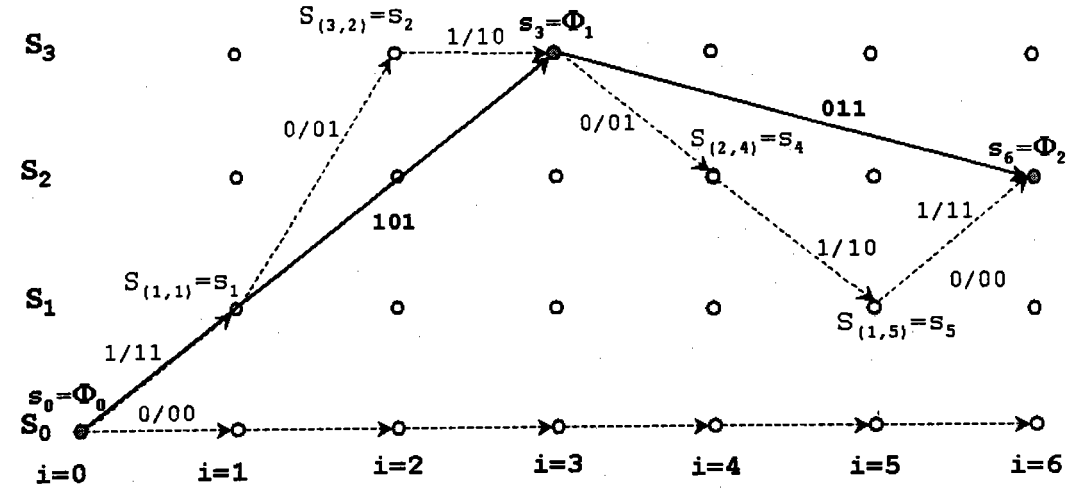



FIG. 5.7 – *Génération des états anticipés pour* $\wp = 3$.

La figure 5.8 schématise le processus de calcul des états anticipés et des états intermédiaires pour un degré de parallélisme $\wp = 3$. Chaque sortie codée est calculée directement dans la structure pipeline à partir d'un couple $(s_i, d_i)$, où $d_i$ représente le bit de donnée correspondant à l'etat $s_i$.

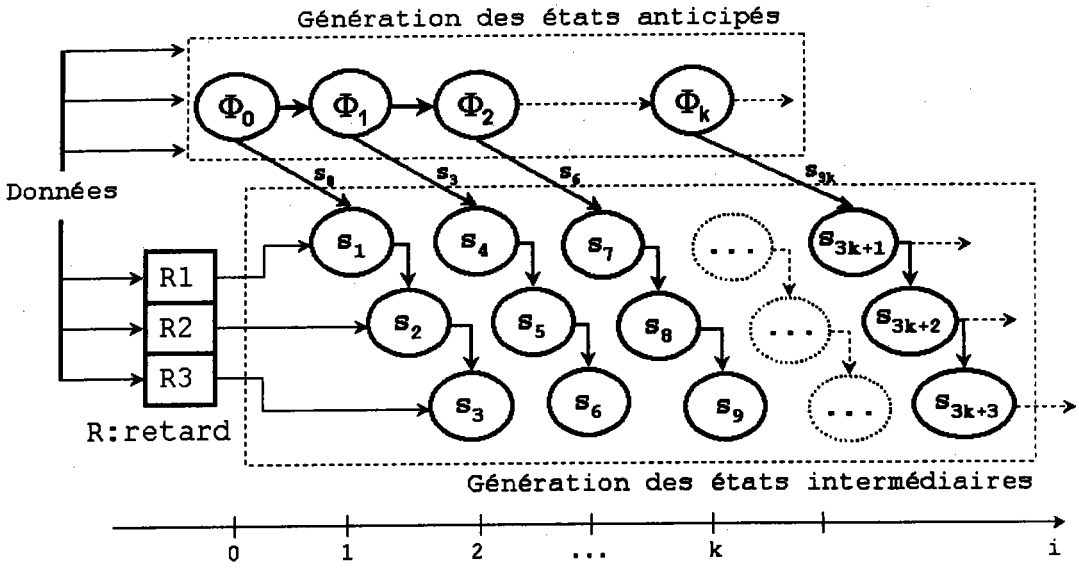



FIG. 5.8 – *Génération des états anticipés et intermédiaires pour* $\wp = 3$.

Pour un degré de parallélisme donné, chaque état anticipé permet de calculer $\wp$ états intermediaires. En prenant par exemple $\wp = 3$, les états intermédiaires $s_i$ sont calculés selon le groupe

d'équations (5.8).

$$\begin{array}{lll} \textbf{EA} & & \textbf{EI} \\ \Phi_0 = s_0 & \Rightarrow & \left\{ \begin{array}{l} s_1 \\ s_2 \\ s_3 \end{array} \right. \\ \Phi_1 = s_3 & \Rightarrow & \left\{ \begin{array}{l} s_4 \\ s_5 \\ s_6 \end{array} \right. \\ \cdots & & \cdots \\ \Phi_k = s_{3k} & \Rightarrow & \left\{ \begin{array}{l} s_{3k+1} \\ s_{3k+2} \\ s_{3k+3} \end{array} \right. \\ \cdots & & \cdots \end{array} \tag{5.8}$$

Dans le cas général, pour un degré de parallélisation donné $\wp$, la génération des EAs et EIs s'effectue suivant l'équation (5.9).

$$\begin{array}{lll} \textbf{EA} & & \textbf{EI} \\ \Phi_k = s_{pk} & \Rightarrow & \left\{ \begin{array}{ll} s_{pk+1} & \\ s_{pk+2} & \\ \cdots & \text{avec } k \in \{0,1,\cdots\} \\ s_{pk+\wp} & \end{array} \right. \end{array} \tag{5.9}$$

Parallèlement au calcul des $s_{pk+1}, \cdots, s_{pk+\wp}$, les bits codés sont générés d'après l'équation (5.10).

$$C_i = F(s_i, d_i) \tag{5.10}$$

## 5.3 Description fonctionnelle du codeur convolutif série

**Notations et terminologies** Avant de commencer la présentation des descriptions fonctionnelles, voici les notations et terminologies qui seront utilisées :

- la notation $(\aleph)_t$ représente le contenu du registre (ou bascule) $\aleph$ à l'instant $t$ ;
- le paramètre $i$ représente le nombre de cycles d'horloge écoulés. L'état $s_i$ par exemple, représente l'état du codeur après $i$ cycles d'horloge ;
- le paramètre $t$ représente le temps. Si l'on note l'instant initial $t_0$ et que si l'on suppose que le contenu du registre $R$ (état du codeur) à l'instant $t_0$ est $s_0$, sachant que $T = 1/f_{clk}$ représente

la période de l'horloge du système, on a alors:

$$\begin{array}{lcll} (R)_{t_0+0\cdot T} & = & (R)_{t_0} & = s_0 \\ (R)_{t_0+1\cdot T} & = & (R)_{t_1} & = s_1 \\ (R)_{t_0+2\cdot T} & = & (R)_{t_2} & = s_2 \\ \ldots & \ldots & \ldots & \ldots \\ (R)_{t_0+i\cdot T} & = & (R)_{t_i} & = s_i \\ (R)_{t_0+(i+1)\cdot T} & = & (R)_{t_{i+1}} & = s_{i+1} \\ \ldots & \ldots & \ldots & \end{array}$$

où $(R)_{t_{i+1}}$ est la représentation simplifiée du contenue du registre $R$ à l'instant $(t+(i+1)\cdot T)$. Le paramètre $T$ sera implicite dans toutes les relations qui suivent. On notera donc à un instant donné $t_i = (t_0 + i \cdot T)$, $(R)_{t_i} = s_i$ et à l'instant suivant $(R)_{t_0+(i+1)\cdot T}$, $(R)_{t_{i+1}} = s_{i+1}$.

Deux types d'équations seront utilisées :

- les équations faisant intervenir des entités telles que les états $s_i$ ou les bits d'information $d_i$, indépendamment d'une architecture particulière seront appelées équations fonctionnelles ;
- les équations faisant intervenir des entités matérielles telles que les registres, décrivant le fonctionnement d'une structure seront appelées équations architecturales.

### 5.3.1 Architecture *Many To One* (MTO) du codeur convolutif

La forme canonique de l'architecture MTO d'un codeur convolutif non systématique avec une sortie de parité est représentée sur la Figure. 5.9. Afin d'assurer un fonctionnement complètement synchrone du système deux registres $u$ et $v$ sont insérés respectivement à l'entrée et à la sortie de ce dernier.

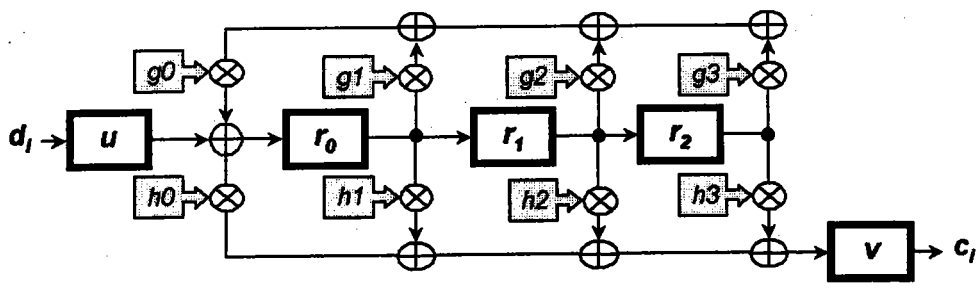



FIG. 5.9 – *Forme canonique du codeur convolutif MTO pour* $m = 3$.

La mémoire du codeur dans l'exemple est $m = 3$. A partir de cette forme canonique, deux configurations du codeur MTO sont possibles.

- **Codeur convolutif non-récursif (CNR)** dans le cas où :

$$(g_0,g_1,g_2,g_3) = \begin{cases} (0,*,*,*) \\ \text{ou} \\ (1,0,0,0) \end{cases} \tag{5.11}$$

  $(*)$ peut prendre la valeur 0 ou 1 où :

- **Codeur convolutif récursif (CR)** dans le cas où :

$$g_0 = 1 \quad \text{et} \quad (g_1,g_2,g_3) \neq (0,0,0) \tag{5.12}$$

  La boucle de retour intervient dans le calcul si $g_0 = 1$ et si au moins un des coefficients restant $g_1$, $g_2$ ou $g_3$ est égal à 1.

Dans la suite, le choix du type de codeur (CR or CNR) sera fait d'après l'équation (5.13).

$$g_0 = 1 \Rightarrow \begin{cases} (g_1,g_2,g_3) = 0, & \text{codeur CNR.} \\ (g_1,g_2,g_3) \neq 0, & \text{codeur CR.} \end{cases} \tag{5.13}$$

A chaque cycle d'horloge, le codeur transite vers un nouvel état en fonction de l'état précédent et du bit d'information présent en entrée. Il fournit simultanément un nouveau bit codé. Cette opération est décrite par les équations architecturales (5.14) et (5.15).

- **Calcul des états**

$$\begin{aligned} (r_0)_{t+1} &= (u)_t + g_0 \cdot [g_1 \cdot (r_0)_t + g_2 \cdot (r_1)_t + g_3 \cdot (r_2)_t] \\ (r_1)_{t+1} &= (r_0)_t \\ (r_2)_{t+1} &= (r_1)_t \end{aligned} \tag{5.14}$$

- **Calcul des sorties**

$$\begin{aligned} (v)_{t+1} &= h_0 \cdot [(u)_t + g_0 \cdot (g_1 \cdot (r_0)_t + g_2 \cdot (r_1)_t + g_3 \cdot (r_2)_t)] \\ &\quad + h_1 \cdot (r_0)_t + h_2 \cdot (r_1)_t + h_3 \cdot (r_2)_t \end{aligned} \tag{5.15}$$

où,

$r_i$ : représente la bascule $i$ du registre R ;

$g_i$ : sont les coefficients du polynôme de la boucle de retour $G(x) = g_0 + g_1x + g_2x^2 + g_3x^3$ ;

$h_i$ : sont les coefficients du polynôme de la branche directe $H(x) = h_0 + h_1x + h_2x^2 + h_3x^3$.

Les équations (5.14) et (5.15) décrivent le fonctionnement du codeur à chaque instant. Pour des codeurs de taille de mémoire $m$ assez grande, cette représentation devient fastidieuse. Par conséquent, on préférera les écrire de façon plus compacte, sous forme matricielle. La forme matricielle des équations architecturales (5.14) et (5.15) est donnée respectivement par les équations (5.16) et (5.17).

$$\begin{aligned}(r_0,r_1,r_2)_{t+1} &= (r_0,r_1,r_2)_t \cdot T_m + ((u)_t,0,0) \\ &= (R)_t \cdot T_m + ((u)_t,0,0) \\ (R)_{t+1} &= (R)_t \cdot T_m + (u)_t \cdot Y_m \end{aligned} \tag{5.16}$$

$$(v)_{t+1} = [((u)_t,0,0,0) + (R)_t \cdot G] \cdot H \tag{5.17}$$

où, $(R)_t = (r_0,r_1,r_2)_t$, $Y_m = (1,0,0)$ et

$$H = \begin{pmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{pmatrix}, \quad G = \begin{pmatrix} g_0g_1 & 1 & 0 & 0 \\ g_0g_2 & 0 & 1 & 0 \\ g_0g_3 & 0 & 0 & 1 \end{pmatrix}, \quad T_m = G \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

Supposons que $s_i$ est l'état du codeur mémorisé dans $R$ à l'instant $t$ ; alors, l'état $s_{i+1}$ mémorisé à l'instant $t+1$ est calculé d'après l'équation fonctionnelle (5.18).

$$s_{i+1} = s_i \cdot T_m + d_i \cdot Y_m \tag{5.18}$$

Pour un codeur MTO de taille de mémoire $m$ quelconque, les équations (5.16) et (5.18) restent valables. Dans ce cas, $(R)_t = (r_0,r_1,\cdots,r_{m-1})_t$, $Y_m = (1,0,\cdots,0)$ et

$$H = \begin{pmatrix} h_0 \\ h_1 \\ \cdot \\ h_{m-1} \\ h_m \end{pmatrix}, \quad G = \begin{pmatrix} g_0g_1 & 1 & 0 & \cdot & 0 \\ g_0g_2 & 0 & 1 & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ g_0g_{m-1} & 0 & 0 & \cdot & 1 \\ g_0g_m & 0 & 0 & \cdot & 1 \end{pmatrix}, \quad T_m = G \cdot \begin{pmatrix} 1 & 0 & \cdot & 0 \\ 0 & 1 & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & \cdot & 1 \\ 0 & 0 & \cdot & 0 \end{pmatrix}$$

En considérant un codeur de taille de mémoire $m$ et de rendement $1/n$, l'équation (5.17) s'écrit :

$$(V)_{t+1} = [((u)_t, 0, \cdots, 0) + (R)_t \cdot G] \cdot H \tag{5.19}$$

avec,

$$\left.\begin{array}{l} (V)_{t+1} = (v_0, v_1, \cdots, v_i)_{t+1}, \\ H = \begin{pmatrix} h_{0,0} & h_{1,0} & \cdot & h_{i,0} \\ h_{0,1} & h_{1,1} & \cdot & h_{i,1} \\ \cdot & \cdot & \cdot & \cdot \\ h_{0,m} & h_{1,m} & \cdot & h_{i,m} \end{pmatrix}. \end{array}\right\} \rightarrow \left\{ \begin{array}{lll} \diamond \text{ Code non systématique} & \rightarrow & i = n-1, \\ \diamond \text{ Code systématique} & \rightarrow & i = n-2. \end{array}\right.$$

### 5.3.2 Architecture *One To Many* (OTM) du codeur convolutif

La forme canonique série d'un codeur convolutif de type OTM, est représentée sur la figure 5.10 pour une taille de mémoire $m = 3$. Comme dans le cas du codeur MTO, afin d'assurer un fonctionnement totalement synchrone du système, deux registres $u$ et $v$ sont insérés respectivement à l'entrée et à la sortie du codeur OTM.

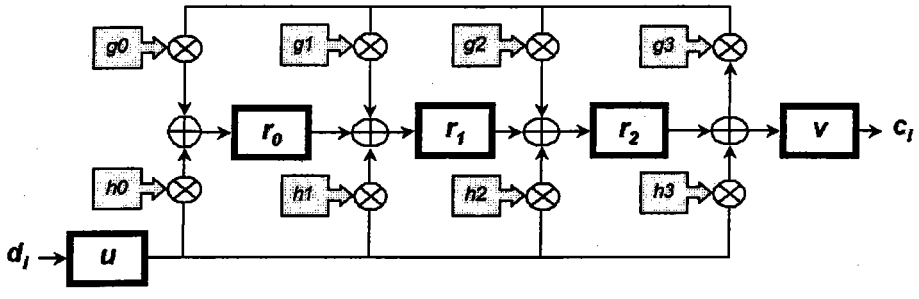



FIG. 5.10 – *Forme canonique du codeur convolutif OTM pour* $m = 3$.

Il existe deux configurations possible du codeur OTM.

- **Codeur convolutif non-récursif (CNR),** dans le cas où :

$$(g_0, g_1, g_2, g_3) = \left\{ \begin{array}{c} (*, *, *, 0) \\ \text{ou} \\ (0,0,0,1) \end{array}\right. \tag{5.20}$$

  $(*)$ peut prendre la valeur 0 ou 1 ;

- **Codeur convolutif récursif (CR),** dans le cas où :

$$g_3 = 1 \quad \text{et} \quad (g_0, g_1, g_2) \neq (0,0,0) \tag{5.21}$$

La boucle de retour intervient dans le calcul uniquement si $g_3 = 1$ et si au moins l'un des coefficients restants $g_0$, $g_1$ ou $g_2$ est égal à 1.

Dans la suite, le choix du type de codeur (CR or CNR) sera faite d'après l'équation (5.22).

$$g_3 = 1 \Rightarrow \begin{cases} (g_0,g_1,g_2) = 0, & \text{codeur CNR ;} \\ (g_0,g_1,g_2) \neq 0, & \text{codeur CR.} \end{cases} \tag{5.22}$$

Les équations architecturales du codeur OTM sont données par (5.23) et (5.24).

– **Calcul des états**

$$\begin{aligned} (r_0)_{t+1} &= h_0 \cdot (u)_t + g_0 g_3[(r_2)_t + h_3 \cdot (u)_t] \\ (r_1)_{t+1} &= h_1 \cdot (u)_t + g_1 g_3[(r_2)_t + h_3 \cdot (u)_t] + (r_0)_t \\ (r_2)_{t+1} &= h_2 \cdot (u)_t + g_2 g_3[(r_2)_t + h_3 \cdot (u)_t] + (r_1)_t \end{aligned} \tag{5.23}$$

– **Calcul des sorties**

$$(v)_{t+1} = (r_2)_t + h_3 \cdot (u)_t \tag{5.24}$$

où,

$r_i$ : repésente la bascule $i$ du registre R ;

$g_i$ : sont les coefficients du polynôme de la boucle de retour $G(x) = g_0 + g_1 x + g_2 x^2 + g_3 x^3$ ;

$h_i$ : sont les coefficients du polynôme de la branche directe $H(x) = h_0 + h_1 x + h_2 x^2 + h_3 x^3$.

L'équation (5.25) représente la forme matricielle de (5.23)

$$(R)_{t+1} = (R)_t \cdot T_o + (u)_t \cdot Y_o \tag{5.25}$$

où[1],

$$Y_o = {}^{tr}\begin{pmatrix} h'_0 \\ h'_1 \\ h'_2 \end{pmatrix} = {}^{tr}\begin{pmatrix} h_0 + g_0 g_3 h_3 \\ h_1 + g_1 g_3 h_3 \\ h_2 + g_2 g_3 h_3 \end{pmatrix}, \quad T_o = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ g_0 g_3 & g_1 g_3 & g_2 g_3 \end{pmatrix}$$

1. La notation ${}^{tr}(\cdots)$ signifie transposé de la matrice $(\cdots)$.

Si $s_i$ représente l'état du codeur à l'instant $t_i$, l'état suivant $s_{i+1}$ à l'instant $t_{i+1}$ est donné par l'équation fonctionnelle (5.26) :

$$s_{i+1} = s_i \cdot T_o + d_i \cdot Y_o \tag{5.26}$$

Pour des codeurs de taille de mémoire $m$, les équations architecturales (5.24), (5.25) et (5.26) restent justes. En généralisant pour $m$, $(R)_t = (r_0, r_1, \cdots, r_{m-1})_t$ et

$$Y_o = {}^{tr}\begin{pmatrix} h'_0 \\ h'_1 \\ \cdot \\ h'_{m-1} \end{pmatrix} = {}^{tr}\begin{pmatrix} h_0 + g_0 g_3 h_3 \\ h_1 + g_1 g_3 h_3 \\ \cdot \\ h_{m-1} + g_{m-1} g_m h_m \end{pmatrix}, \quad T_o = \begin{pmatrix} 0 & 1 & 0 & \cdot & 0 \\ 0 & 0 & 1 & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & \cdot & 1 \\ g_0 g_m & g_1 g_m & g_2 g_m & \cdot & g_{m-1} g_m \end{pmatrix}.$$

## 5.4 Diviseur parallèle

### 5.4.1 Description fonctionnelle du diviseur parallèle

Si l'on ne considère que la partie récursive du codeur convolutif, sa structure est similaire à celle d'un diviseur (cf. paragraphe 2.2.2). Le calcul d'un état $s_{i+1}$ dans la structure série du codeur convolutif s'effectue de la même manière que le calcul du reste de la division. Les équations (5.18) et (5.26), relatives aux codeurs MTO et OTM respectivement, décrivent l'opération effectuée à chaque cycle d'horloge. Les équations fonctionnelles (5.18) et (5.26) ont la forme générale donnée par (5.27).

$$s_{i+1} = F(s_i, d_i) = s_i \cdot T + d_i \cdot Y = F_1(s_i) + F_2(d_i) \tag{5.27}$$

où,

$T$ : matrice de dimension $m \times m$

$Y$ : vecteur de longueur $m$

D'après l'équation (5.27), l'état $s_{i+2}$ est calculé de la façon suivante :

$$s_{i+2} = s_{i+1} \cdot T + d_{i+1} \cdot Y \tag{5.28}$$

En remplaçant $E_{i+1}$ dans l'équation (5.28) par son expression donnée dans (5.27), on obtient l'équation (5.29) qui permet le calcul de l'état $s_{i+2}$ à partir de ($s_i$, $d_i$, $d_{i+1}$) sans passer par $s_{i+1}$.

$$\begin{aligned} s_{i+2} &= (s_i \cdot T + d_i \cdot Y) \cdot T + d_{i+1} \cdot Y \\ &= s_i \cdot T^2 + d_i \cdot Y \cdot T + d_{i+1} \cdot Y \end{aligned} \tag{5.29}$$

En procédant de la même manière, on obtient l'état $s_{i+3}$ en fonction de $(s_i, d_i, d_{i+1}, d_{i+2})$.

$$\begin{aligned} s_{i+3} &= s_{i+2} \cdot T + d_{i+2} \cdot Y, \\ &= (s_{i+1} \cdot T + d_{i+1} \cdot Y) \cdot T + d_{i+2} \cdot Y \\ &= s_{i+1} \cdot T^2 + d_{i+1} \cdot Y \cdot T + d_{i+2} \cdot Y \\ &= (s_i \cdot T + d_i \cdot Y) \cdot T^2 + d_{i+1} \cdot Y \cdot T + d_{i+2} \cdot Y \\ s_{i+3} &= s_i \cdot T^3 + d_i \cdot Y \cdot T^2 + d_{i+1} \cdot Y \cdot T + d_{i+2} \cdot Y \end{aligned} \tag{5.30}$$

En généralisant à un degré de parallélisation $\wp$ quelconque, on obtient l'équation fonctionnelle générale de recurrence (5.31).

$$\begin{aligned} s_{i+\wp} &= s_{i+\wp-1} \cdot T + d_{i+\wp-1} \cdot Y \\ &= s_{i+\wp-2} \cdot T^2 + d_{i+\wp-2} \cdot Y \cdot T + d_{i+\wp-1} \cdot Y \\ &= \cdots \\ s_{i+\wp} &= s_i \cdot T^{\wp} + d_i \cdot Y \cdot T^{\wp-1} + d_{i+1} \cdot Y \cdot T^{\wp-2} + \cdots + d_{i+\wp-1} \cdot Y \end{aligned} \tag{5.31}$$

L'expression (5.31) peut être réarrangée de la manière suivante :

$$\begin{aligned} s_{i+\wp} &= s_i \cdot T^{\wp} \\ &\quad +(d_i, d_{i+1}, \cdots, d_{i+\wp-1}) \cdot Z^0 \cdot L \cdot T^{\wp-1} \\ &\quad +(d_i, d_{i+1}, \cdots, d_{i+\wp-1}) \cdot Z^1 \cdot L \cdot T^{\wp-2} \\ &\quad +\cdots \\ &\quad +(d_i, d_{i+1}, \cdots, d_{i+\wp-1}) \cdot Z^{\wp-1} \cdot L \cdot T^0 \\ s_{i+\wp} &= s_i \cdot T^{\wp} + (d_i, d_{i+1}, \cdots, d_{i+\wp-1}) \cdot \left( \sum_{j=0}^{\wp-1} Z^j \cdot L \cdot T^{\wp-j-1} \right) \\ s_{i+\wp} &= s_i \cdot T^{\wp} + (d_i, d_{i+1}, \cdots, d_{i+\wp-1}) \cdot M. \end{aligned} \tag{5.32, 5.33}$$

où $Z$ représente une matrice de dimension $\wp \times \wp$, $L$ une matrice de dimension $\wp \times m$ et $Y$ un vecteur de dimension $m$, avec :

$$Z = \begin{pmatrix} 0 & 0 & . & 0 & 1 \\ 1 & 0 & . & 0 & 0 \\ 0 & 1 & . & 0 & 0 \\ . & . & . & . & . \\ 0 & 0 & . & 1 & 0 \end{pmatrix}, \quad L = \begin{pmatrix} y_0 & y_1 & . & y_{m-1} \\ 0 & 0 & . & 0 \\ . & . & . & . \\ 0 & 0 & . & 0 \end{pmatrix}, \quad Y = (y_0, y_1, \cdots, y_{m-1}).$$

L'équation (5.33) calcule l'état $s_{i+\wp}$ en fonction de l'état $s_i$ et des $\wp$ bits de données précédents.

On effectue le calcul des états $s_{i+2\cdot\wp}$, $s_{i+3\cdot\wp}$, $\cdots$, $s_{i+k\cdot\wp}$ de la même façon suivant :

$$\begin{aligned}
s_{i+2\cdot\wp} &= s_{i+\wp}\cdot T^{\wp} + (d_{i+\wp}, d_{i+\wp+1}, \cdots, d_{i+2\cdot\wp-1})\cdot M \\
s_{i+3\cdot\wp} &= s_{i+2\cdot\wp}\cdot T^{\wp} + (d_{i+2\cdot\wp}, d_{i+2\cdot\wp+1}, \cdots, d_{i+3\cdot\wp-1})\cdot M \\
\cdots &= \cdots \\
s_{i+k\cdot\wp} &= s_{i+(k-1)\cdot\wp}\cdot T^{\wp} + (d_{i+(k-1)\cdot\wp}, \cdots, d_{i+k\cdot\wp-1})\cdot M
\end{aligned}$$

Ainsi, par recurrence on obtient l'équation fonctionnelle générale de calcul des restes $\Re_k$ de la division de $k\cdot\wp$ bits par le polynôme $G(x)$ (équ. (5.35)).

$$\begin{aligned}
\Re_k = s_{i+k\cdot\wp} &= s_{i+(k-1)\cdot\wp}\cdot T^{\wp} + {}^{\wp}D_{i+(k-1)\cdot\wp}\cdot M, \\
&= F_1{}^*(s_{i+(k-1)\cdot\wp}) + F_2{}^*({}^{\wp}D_{i+(k-1)\cdot\wp}) \qquad (5.34) \\
&= F^*(s_{i+(k-1)\cdot\wp}, {}^{\wp}D_{i+(k-1)\cdot\wp}), \\
\Re_k &= F^*(\Re_{k-1}, {}^{\wp}D_{i+(k-1)\cdot\wp}) \qquad (5.35)
\end{aligned}$$

avec, $k = \{1, 2, 3, \cdots\}$, $M = \left(\sum_{j=0}^{\wp-1} Z^j \cdot L \cdot T^{\wp-j-1}\right)$ et ${}^{\wp}D_{i+(k-1)\cdot\wp} = (d_{i+(k-1)\cdot\wp}, \cdots, d_{i+k\cdot\wp-1})$

En appliquant la formule (5.35) dans le cas où l'état initial est $s_0$ à l'instant $t_0$, les restes sont donnés par (5.36).

| **Instant** | | $\Re$ | | |
|---|---|---|---|---|
| $t+0:$ | $\mapsto$ | $\Re_0$ | $=$ | $s_0$ |
| $t+1:$ | $\mapsto$ | $\Re_1$ | $=$ | $s_{\wp}$ |
| $t+2:$ | $\mapsto$ | $\Re_2$ | $=$ | $s_{2\cdot\wp}$ |
| $\cdots$ | | $\cdots$ | | $\cdots$ |
| $t+k:$ | $\mapsto$ | $\Re_{k+1}$ | $=$ | $s_{k\cdot\wp}$ |

(5.36)

L'équation architecturale déduite de l'équation fonctionnelle (5.37) est donnée par (5.35). La figure 5.11 présente le schéma fonctionnel du diviseur parallèle.

$$(R^A)_{t+1} = (R^A)_t \cdot T^{\wp} + (U)_t \cdot M \qquad (5.37)$$

avec,
- $R^A = (r_0, \cdots, r_{m-1})^A$ : registre d'accumulation du reste taille $m$ ;
- $U = (u_0, \cdots, u_{\wp-1})$ : registre de taille $\wp$.

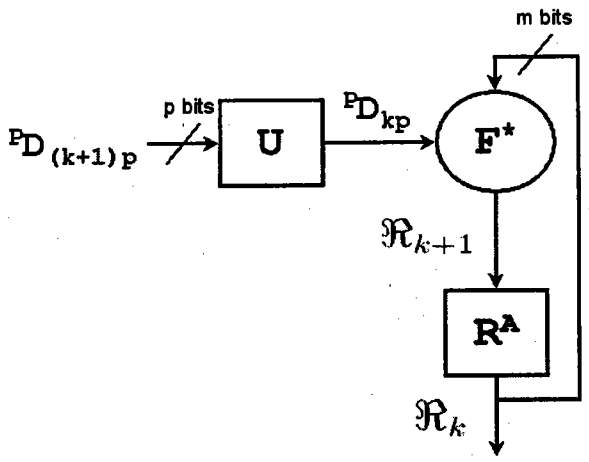



FIG. 5.11 – *Schéma fonctionnel du diviseur parallèle.*

### 5.4.2 Implantation du diviseur parallèle

Comme le montre la figure 5.12, le diviseur parallèle se constitue de deux parties réalisant respectivement les fonctions $F_1{}^*$ et $F_2{}^*$ de l'équation (5.34).

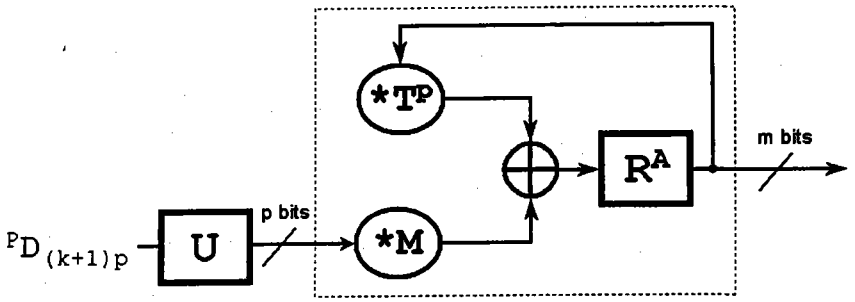



FIG. 5.12 – *Implantation du diviseur parallèle.*

Le bloc qui calcule la fonction $F_1{}^*$ $(\times T^p)$ doit satisfaire aux deux conditions suivantes :

- fournir un résultat à chaque top d'horloge ;
- aucun temps de latence n'est acceptable.

Les deux contraintes, liées au bloc $F_1{}^*$, permettent uniquement une implantation combinatoire de ce dernier. Quant au bloc qui calcule $F_2{}^*$ $(\times M)$, il accepte une latence au début du calcul ce qui permet de l'implanter, soit de façon combinatoire, soit avec une structure pipeline pour réduire son chemin critique. Bien sûr, dans le deuxième cas, un temps de latence sera introduit dans la réponse du diviseur parallèle. La figure 5.13 donne un exemple d'implantation pipeline du bloc $F_2{}^*$ $(\times M)$ pour un degré de parallélisme $\wp = 3$. A noter que cette structure pipeline n'est pas la seule possible.

Dans [17] par exemple, nous avons développé un diviseur parallèle OTM dont le bloc $F_2{}^*$ est implanté en utilisant un pipeline : chaque étage du pipeline est un diviseur série OTM précédé d'un registre à décalage.

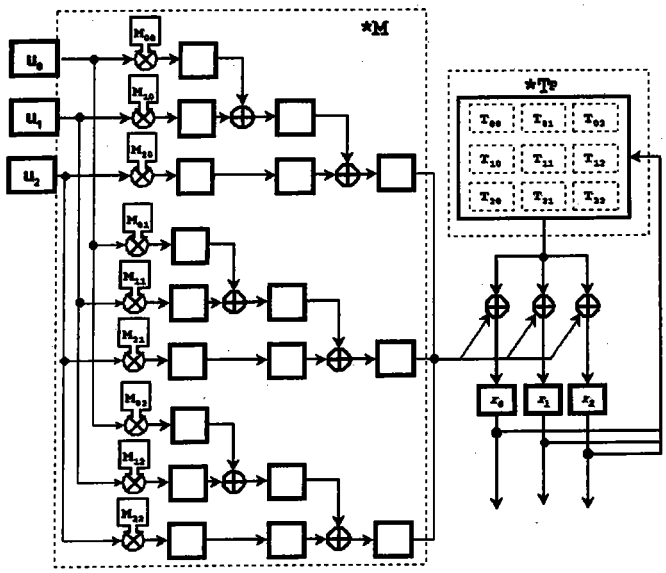


FIG. 5.13 – *Exemple d'une implantation pipeline du bloc $F_2{}^*$ $(\times M)$ pour $\wp = 3$.*

Les deux modèles du diviseur parallèle, correspondant aux structures combinatoire et pipeline de la fonction $F_2{}^*$ ont été implantés sur un FPGA Flex10KE. Chaque modèle a été réalisé pour 3 degrés de parallélisme $\wp$=8, 16 et 32. Les polynômes diviseurs sont donnés dans le tableau 5.2. Ils est à noter que les polynômes $G_6(x)$ et $G_9(x)$ sont utilisés dans les diviseurs de codeurs/décodeurs blocs respectivement du protocole $X25$ pour le premier et de FDDI et ATM/couche AAL5 pour le deuxième.

| Notation | Degré | Polynôme diviseur |
|---|---|---|
| $G_1(x)$ | 2 | $1+x+x^2$ |
| $G_2(x)$ | 3 | $1+x+x^3$ |
| $G_3(x)$ | 4 | $1+x^3+x^4$ |
| $G_4(x)$ | 8 | $1+x+x^2+x^3+x^6+x^7+x^8$ |
| $G_5(x)$ | 16 | $1+x^2+x^{15}+x^{16}$ |
| $G_6(x)$ | 16 | $1+x^5+x^{12}+x^{16}$ |
| $G_7(x)$ | 32 | $1+x+x^4+x^6+x^8+x^{10}+x^{12}+x^{13}$ $+x^{15}+x^{17}+x^{18}+x^{20}+x^{21}+x^{22}+x^{32}$ |
| $G_8(x)$ | 32 | $1+x+x^2+x^{22}+x^{32}$ |
| $G_9(x)$ | 32 | $1+x+x^2+x^4+x^5+x^7+x^8+x^{10}$ $+x^{11}+x^{12}+x^{16}+x^{22}+x^{23}+x^{26}+x^{32}$ |

TAB. 5.2 – *Polynômes diviseurs*

Les figures 5.14 et 5.15 présentent les courbes de compromis débit/surface dans les cas combinatoire et pipeline, respectivement. La version pipeline consomme plus de surface que la version combinatoire. Cependant les débits atteints par la structure pipeline sont plus élevés dans la plupart des cas. Pour les tailles de mémoires $m \leq 8$ et les degrés de parallélisme grand 16 et 32, la version pipeline est plus performante. En effet, le débit atteint pour le diviseur $G_6(x)$ avec la version pipeline est de 4,65 Gbits/s alors qu'il est limité à 2,89 Gbits/s avec la version combinatoire. Pour $\wp = 8$, les deux versions se valent avec moins de surface consommée pour la version combinatoire.

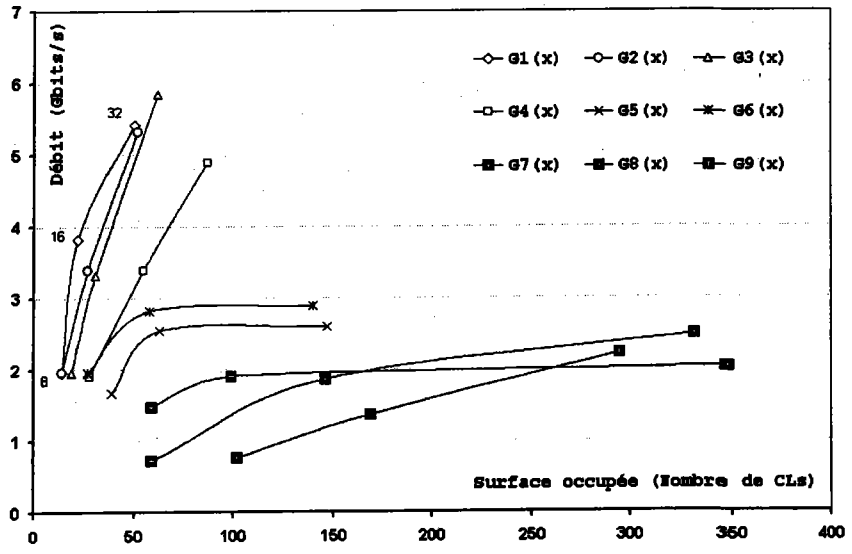



FIG. 5.14 – *Compromis débit/surface du diviseur parallèle* ($F_2$**combinatoire*).

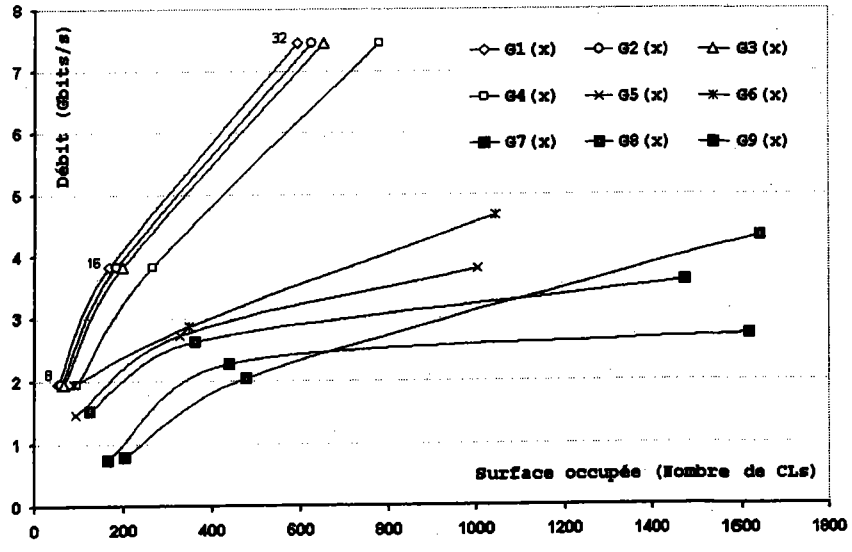



FIG. 5.15 – *Compromis débit/surface du diviseur parallèle* ($F_2$**pipeline*).

La figure 5.16 représente les courbes de simulation d'un diviseur série MTO de mémoire $m = 4$ et polynôme diviseur $G_3(x)$. La sortie `Reste_serie` représente à chaque cycle d'horloge le reste partiel de la division de l'entrée `Din_serie` par $G_3(x)$ . Les valeurs en gris représentent le reste après chaque séquence de 8 cycles d'horloge consécutifs.

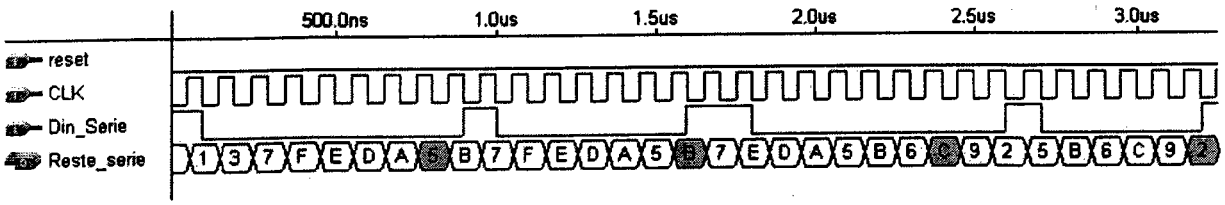



FIG. 5.16 – *Simulation du diviseur série.*

La figure 5.17 présente la simulation du diviseur parallèle ($F_2{}^*$ implantée en combinatoire) pour $\wp = 8$, $m = 4$ et $G_3(x)$. La sortie `Reste` représente le reste de la division parallèle de l'entrée `Din` par le diviseur $G_3(x)$. A chaque cycle d'horloge la structure parallèle fournit un reste partiel correspondant au $(8 \cdot k)^{eme}$ reste partiel avec $k \in \{0,1,\cdots\}$, représenté en gris dans la figure de la structure série (fig. 5.16).

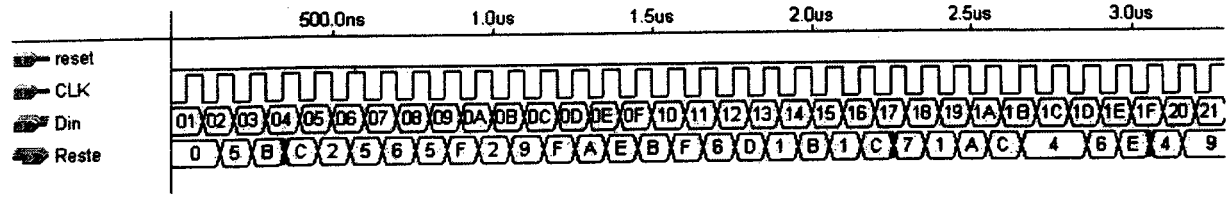



FIG. 5.17 – *Simulation du diviseur parallèle ($F_2{}^*$combinatoire) pour $m = 4$ et $\wp = 8$.*

La figure 5.18 présente la simulation du même codeur pour une implantation pipeline de $F_2{}^*$. Bien sûr, pour les mêmes paramètres ($\wp = 8$, $m = 4$ et $G_3(x)$) et la même séquence d'entrée on obtient les mêmes restes partiels. Cependant, on remarque la présence d'un temps de latence proportionnel au degré de parallélisme $\wp$ et généré par la structure pipeline.

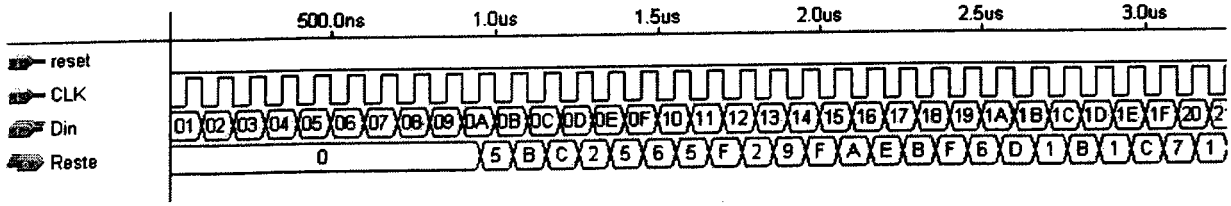



FIG. 5.18 – *Simulation du diviseur parallèle ($F_2{}^*$pipeline) pour $m = 4$ et $\wp = 8$.*

## 5.5 Etude du codeur convolutif haut débit

### 5.5.1 Présentation du codeur convolutif haut débit

Le schéma fonctionnel du codeur haut débit est représenté sur la figure 5.19, il est constitué de trois blocs.

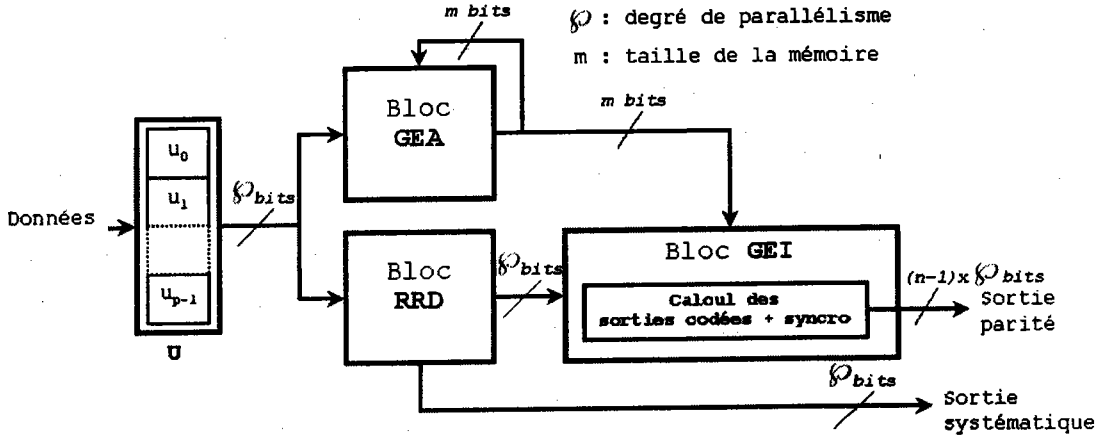



FIG. 5.19 – *Schéma fonctionnel du codeur systématique parallèle.*

- **Bloc de génération des états anticipés (GEA)** Ce bloc fournit à chaque top d'horloge un nouvel état anticipé $\Phi_{k+1}$, calculé à partir de l'état $\Phi_k$ précédent et des $\wp$ bits présents à l'entrée. Le bloc possède deux entrées et une sortie. Les deux entrées correspondent aux états anticipés $\Phi_k$ et bits d'information dont les largeurs respectives sont $m$ et $\wp$ bits. La sortie correspond à l'état anticipé calculé $\Phi_{k+1}$, sa largeur est $m$ bits.

- **Bloc de génération des états intermédiaires (GEI)** Ce bloc présente une structure pipeline, constituée de codeurs série connectés entre eux selon une topologie prédéfinie. Chaque codeur, représentant un étage du pipeline, permet le calcul d'un état intermédiaire $s_i$ et, en fonction du rendement du codeur,la génération d'un ou de plusieurs bits codés. Le bloc possède deux entrées et une sortie. Les deux entrées correspondent aux états anticipés $\Phi_k$, générés par le bloc GEA, et les bits d'information, fournie par le bloc RRD, dont les largeurs respectives sont $m$ et $\wp$ bits. La sortie correspond aux bits codés. Sa largeur est de $(\wp \cdot (n-1))$ bits si le code est systématique ou de $(\wp \cdot (n))$ bits dans le cas contraire.

- **Réseau de Registre à Décalage (RRD)** Ce bloc à deux rôles :
  - il permet de synchroniser chaque bit de donnée $d_i$ avec l'état correspondant $s_i$. Les bits de donnée qui entrent dans le réseau sont décalés, puis prélevés dans le réseau selon un ordre précis avant d'être injectés dans le bloc GEI ;

- dans le cas d'un codeur systématique, il fournit, par simple décalage, les bits systématiques du code. En effet, les données qui entrent sont propagées à travers le réseau et se retrouvent à la sortie après $\wp$ cycles d'horloge.

Ce bloc a un bus de donnée de $\wp$ bits en entrée, et deux ou un bus de $\wp$ bits en sortie selon que le code est systématique ou non.

### 5.5.2 Description fonctionnelle de l'architecture parallèle-pipeline

**Le Bloc GEA - Le diviseur parallèle**

Le bloc GEA est un diviseur parallèle. A chaque cycle d'horloge il fournit l'état anticipé $\Phi_k = \Re_k$ suivant l'équation fontionnelle (5.38).

$$\Phi_k = \Re_k \;=\; F^*(\Phi_{k-1}, {}^{\wp}D_{i+(k-1)\cdot\wp}) \tag{5.38}$$

avec, $k = \{1, 2, 3, \cdots\}$ et ${}^{\wp}D_{i+(k-1)\cdot\wp} = (d_{i+(k-1)\cdot\wp}, \cdots, d_{i+k\cdot\wp-1})$.

Les états anticipés générés sont donnés par (5.39), en appliquant l'équation fonctionnelle (5.35) dans le cas d'un état initial $s_0$ à l'instant $t$.

$$\begin{array}{lcll} \texttt{Instant} & & \Phi & \\ t+0: & \mapsto & \Phi_0 = & s_0 \\ t+1: & \mapsto & \Phi_1 = & s_{\wp} \\ t+2: & \mapsto & \Phi_2 = & s_{2\cdot\wp} \\ \cdots & & \cdots & \cdots \\ t+k: & \mapsto & \Phi_k = & s_{k\cdot\wp} \end{array} \tag{5.39}$$

**Le Bloc RRD**

Comme mentionné précédemment, les $\wp$ bits de donnée qui intègrent le codeur sont décalés à travers le réseau de registres à décalage (équ. (5.40)). Pour, d'une part, synchroniser chaque bit avec l'état intermédiaire correspondant dans le bloc GEI, et d'autre part générer la sortie systématique dans le cas de codes systématiques.

$$\begin{cases} (U'_0)_{t+1} = (u'_{0,0}, \cdots, u'_{0,\wp-1})_{t+1} = (U)_t \\ (U'_k)_{t+1} = (u'_{k,0}, \cdots, u'_{k,\wp-1})_{t+1} = (U'_{k-1})_t \\ \\ \text{avec } k = \{1, 2, \cdots, (\wp-1)\}. \end{cases} \tag{5.40}$$

Comme le montre la figure 5.20, le bloc RRD est composé de $\wp \times \wp$ cellules de retard (bascules).

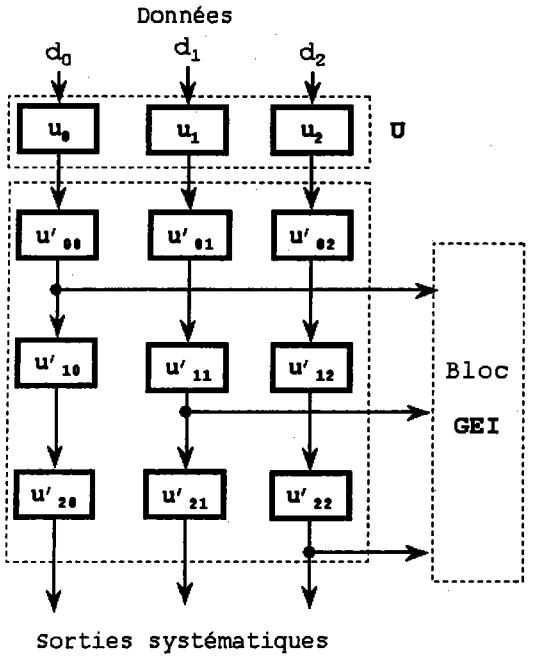



FIG. 5.20 – *Architecture interne du bloc RRD pour* $\wp = 3$.

Dans l'exemple, $\wp = 3$.

**Le bloc GEI**

Le bloc GEI est un pipeline à $\wp$ étages. Chaque état intermédiaire ($EI$) inclus entre deux états anticipés $\Phi_k = s_{k\cdot\wp}$ et $\Phi_{k+1} = s_{k\cdot\wp+\wp}$ est calculé par un des étages de pipeline. Ainsi, l'étage $k$ calcule l'état $EI_k$ à partir de l'état $EI_{k-1}$ et du bit de donnée correspondant d'après l'equation fonctionnelle (5.27). Le tableau 5.3 montre comment les états intermédiaires sont calculés dans les différents étages.

| Cycle | | 0 | 1 | 2 | 3 | ··· | i |
|---|---|---|---|---|---|---|---|
| Bloc GEA | | $\Phi_0 = s_0$ | $\Phi_1 = s_3$ | $\Phi_2 = s_6$ | $\Phi_3 = s_9$ | ··· | $\Phi_i = s_{i\cdot\wp}$ |
| Bloc GEI | Étage 1 | 0 | $s_1$ | $s_4$ | $s_7$ | ··· | $s_{i\cdot\wp-1\cdot(\wp-1)}$ |
| | Étage 2 | 0 | 0 | $s_2$ | $s_5$ | ··· | $s_{i\cdot\wp-2\cdot(\wp-1)}$ |
| | Étage 3 | 0 | 0 | 0 | $s_3$ | ··· | $s_{i\cdot\wp-3\cdot(\wp-1)}$ |
| | ··· | ··· | ··· | ··· | ··· | ··· | ··· |
| | Étage k | 0 | 0 | 0 | 0 | ··· | $s_{i\cdot\wp-k\cdot(\wp-1)}$ |
| | ··· | ··· | ··· | ··· | ··· | ··· | ··· |
| | Étage $\wp$ | 0 | 0 | 0 | 0 | ··· | $s_{i\cdot\wp-\wp\cdot(\wp-1)}$ |

TAB. 5.3 – *Flot de propagation des EIs dans le pipeline GEI.*

avec $i \in \{0,1,\cdots\}$, $k \in \{1,\cdots,\wp-1\}$ et si $(i\cdot\wp - k\cdot(\wp-1)) < 0 \Rightarrow s_{i\cdot\wp-k\cdot(\wp-1)} = 0$.

Chaque étage du pipeline est relié à l'étage précédent et au bloc RRD suivant l'équation architecturale (5.41). Les bits codés sortent directement de chaque étage.

$$\begin{cases} (R^I{}_0)_{t+1} = (R^A)_t \cdot T + (u'_{0,0})_t \cdot Y = F((R^A)_t,(u'_{0,0})_t) \\ (R^I{}_k)_{t+1} = (R^I{}_{k-1})_t \cdot T + (u'_{k,k})_t \cdot Y = F((R^I{}_{k-1})_t,(u'_{k,k})_t) \\ \\ \text{avec } k \in \{1,\cdots,\wp-1\}. \end{cases} \tag{5.41}$$

Le bloc GEI de la figure 5.21 représente un pipeline à 3 étages. Dans le cas général, sa taille est de $\wp$ étages.

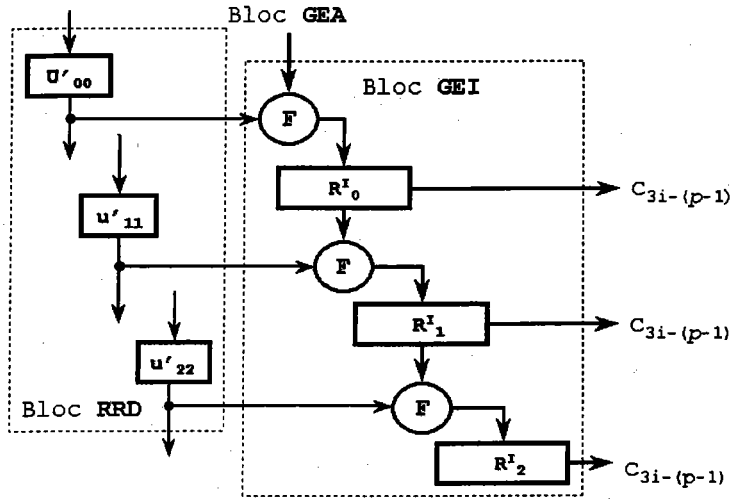



FIG. 5.21 – *Architecture interne du bloc GEI pour* $\wp = 3$.

## 5.6 Implantation du codeur convolutif parallèle

### 5.6.1 Implantation du codeur parallèle type MTO ($CP_{mto}$)

Par analogie avec l'équation fonctionnelle générale (5.27), l'équation (5.18) permettant le calcul des états du codeur MTO série à chaque top d'horloge, est obtenue en remplaçant juste les valeurs des matrices $T$ et $Y$ par celles correspondant à la version MTO selon l'équation (5.42).

$$\Rightarrow \begin{cases} T = T_m \\ Y = Y_m \end{cases} \tag{5.42}$$

Par conséquent, l'équation fonctionnelle (5.43) qui calcule les états anticipés dans un codeur parallèle MTO se déduit directement de l'équation (5.35).

$$\begin{aligned}\Phi_k = s_{i+k\cdot\wp} &= s_{i+(k-1)\cdot\wp}\cdot {T_m}^{\wp} + {}^{\wp}D_{i+(k-1)\cdot\wp}\cdot M_m \\ &= F^*(s_{i+(k-1)\cdot\wp},\ {}^{\wp}D_{i+(k-1)\cdot\wp})\end{aligned} \tag{5.43}$$

avec, $M_m = \left(\sum_{j=0}^{\wp-1} Z^j \cdot L_m \cdot T_m^{\wp-j-1}\right)$, $L_m = \begin{pmatrix} 1 & 0 & . & 0 \\ 0 & 0 & . & 0 \\ . & . & . & . \\ 0 & 0 & . & 0 \end{pmatrix}$ et ${}^{\wp}D_i = (d_i, \cdots, d_{i+\wp-1})$.

L'équation architecturale, déduite de l'équation fonctionnelle (5.43), décrivant le fonctionnement du bloc GEA est donnée par l'équation (5.44).

$$(R^A)_{t+1} = (R^A)_t \cdot T_m^{\wp} + (U)_t \cdot M_m \tag{5.44}$$

En ce qui concerne le bloc GEI, son fonctionnement est décrit par l'équation architecturale (5.45).

$$\begin{cases} (R^I{}_0)_{t+1} = (R^A)_t \cdot T_m + (u'_{0,0})_t \cdot Y_m = F((R^A)_t, (u'_{0,0})_t) \\ (R^I{}_k)_{t+1} = (R^I{}_{k-1})_t \cdot T_m + (u'_{k,k})_t \cdot Y_m = F((R^I{}_{k-1})_t, (u'_{k,k})_t) \\ \\ \text{avec } k \in \{1, \cdots, \wp - 1\}. \end{cases} \tag{5.45}$$

Dans le cas d'un codeur de rendement $1/n$, les $\wp$ sorties parallèles sont calculées directement à partir de chaque étage du pipeline selon :

$$(V_k)_{t+1} = [((u'_{k,k})_t, 0, \cdots, 0) + (R^I)_t \cdot G] \cdot H \tag{5.46}$$

avec $(V_k)_{t+1} = (v_{k,0}, v_{k,1}, \cdots, v_{k,i})_{t+1}$,

$$H = \begin{pmatrix} h_{0,0} & h_{1,0} & . & h_{i,0} \\ h_{0,1} & h_{1,1} & . & h_{i,1} \\ . & . & . & . \\ h_{0,m} & h_{1,m} & . & h_{i,m} \end{pmatrix} \text{ et } G = \begin{pmatrix} g_0 g_1 & 1 & 0 & . & 0 \\ g_0 g_2 & 0 & 1 & . & 0 \\ . & . & . & . & . \\ g_0 g_{m-1} & 0 & 0 & . & 1 \\ g_0 g_m & 0 & 0 & . & 1 \end{pmatrix}.$$

$$\begin{cases} \diamond \text{ Code non systématique} \rightarrow i = n-1, \\ \diamond \text{ Code systématique} \rightarrow i = n-2. \end{cases}$$

Le fonctionnement du bloc RRD est, quant à lui, décrit par l'équation architecturale (5.40).

### 5.6.2 Implantation du codeur parallèle type OTM ($CP_{otm}$)

Le codeur parallèle OTM se construit aisément, en utilisant la même approche que celle du paragraphe 5.6.1. En effet, en remplaçant, comme cela est décrit par l'équation (5.47), les matrices $T$ et $Y$ dans l'équation (5.27) par les valeurs qui correspondent à une structure OTM, nous obtenons l'équation correspondant au calcul des états du codeur OTM série, c'est à dire l'équation (5.26).

$$\Rightarrow \begin{cases} T &= T_o \\ Y &= Y_o \end{cases} \tag{5.47}$$

Par analogie, le calcul des états anticipés dans le cas OTM se déduit directement de l'équation fonctionnelle (5.35) et conduit à l'équation (5.48).

$$\begin{aligned} \Phi_k = s_{i+k\cdot\wp} &= s_{i+(k-1)\cdot\wp} \cdot T_o^{\wp} + {}^{\wp}D_{i+(k-1)\cdot\wp} \cdot M_o \\ &= F^*(s_{i+(k-1)\cdot\wp}, {}^{\wp}D_{i+(k-1)\cdot\wp}) \end{aligned} \tag{5.48}$$

avec $M_o = \left(\sum_{j=0}^{\wp-1} Z^j \cdot L_o \cdot T_o^{\wp-j-1}\right)$, $L_o = \begin{pmatrix} h'_0 & h'_1 & . & h'_{m-1} \\ 0 & 0 & . & 0 \\ . & . & . & . \\ 0 & 0 & . & 0 \end{pmatrix}$ et ${}^{\wp}D_i = (d_i, \cdots, d_{i+\wp-1})$.

L'équation architecturale du bloc GEA dans le cas OTM, déduite de l'équation fonctionnelle (5.48), est donnée par (5.49).

$$(R^A)_{t+1} = (R^A)_t \cdot T_o^{\wp} + (U)_t \cdot M_o \tag{5.49}$$

L'équation architecturale du bloc GEI est donnée par (5.50).

$$\begin{cases} (R^I{}_0)_{t+1} &= (R^A)_t \cdot T_o + (u'_{0,0})_t \cdot Y_o &= F((R^A)_t, (u'_{0,0})_t) \\ (R^I{}_k)_{t+1} &= (R^I{}_{k-1})_t \cdot T_o + (u'_{k,k})_t \cdot Y_o &= F((R^I{}_{k-1})_t, (u'_{k,k})_t) \\ \\ \text{avec } k \in \{1, \cdots, \wp - 1\}\,. \end{cases} \tag{5.50}$$

Comme pour le codeur $CP_{mto}$, les bits codées dans le cas $CP_{otm}$ sortent directement de chaque étage (équ. (5.51)).

$$(v_k)_{t+1} = (r_{k,m-1})_t + h_m \cdot (u'_{k,k})_t \tag{5.51}$$

où $m$ est la taille de la mémoire du codeur série et $h_m$ représente le coefficient correspondant au poids fort du polynôme générateur de la branche directe. Le bloc RRD reste inchangé.

**Remarque** dans le cas $CP_{otm}$, on considère uniquement les codeurs systématiques de rendement $1/2$.

## 5.7 Résultats expérimentaux

### 5.7.1 Synthèse de l'architecture parallèle

Le modèle du codeur parallèle a été décrit en VHDL au niveau RTL. Il a été implanté sur des FPGA de type Flex10KE d'Altera. La fréquence maximale de fonctionnement ($f_{max}$) de cette famille de FPGA est limité à 250 MHz. Par conséquent, tous les résultats obtenus en terme de fréquence de fonctionnement seront comparés à cette limite $f_{max}$.

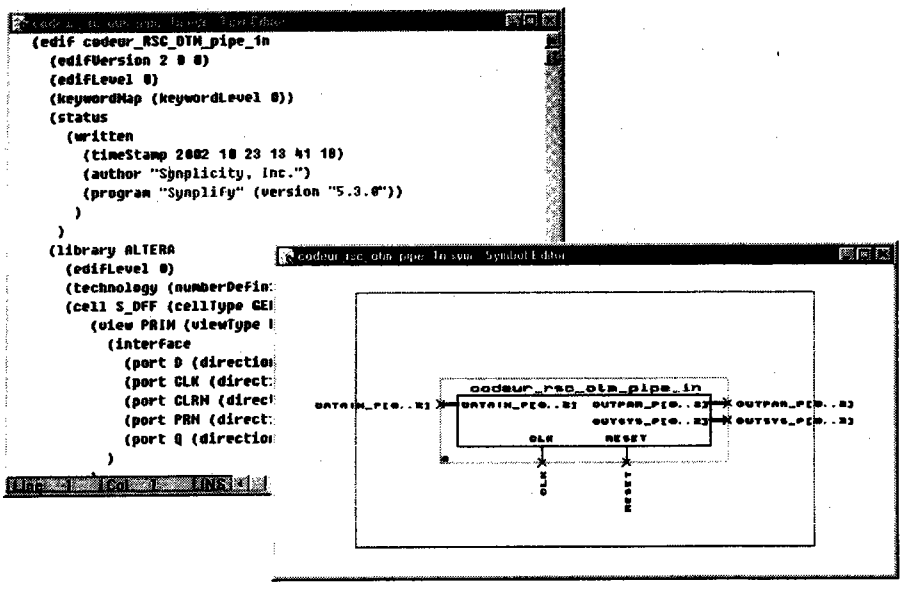



FIG. 5.22 – *Netlist générée après synthèse et symbole du codeur correspondant.*

La synthèse du modèle a été effectuée avec Synplify (version 5.3.1) de Synplicity. Le placement/routage de la netlist issue de Synplify et les simulations on été réalisés avec MAX+plus II (version 10.0) d'Altera. La figure 5.22 représente le symbole d'un codeur généré par MAX+plus II à partir de la netlist fournie par Synplify.

Les modèles de codeur $CP_{mto}$ et $CP_{otm}$ ont été décrits de façon générique, indépendamment d'une technologie cible. La phase d'initialisation des paramètres génériques (codeur systématique ou non, taille $m$ de la mémoire, valeur des polynômes générateurs, rendement $R$ et degré de parallélisme $\wp$) s'effectue avant la synthèse par l'intermédiaire d'un paquetage généré par un programme de pré-calcul écrit en langage C (fig. 5.23).

Le programme de pré-calcul a pour rôle de :

- calculer, à partir des paramètres de configuration, les matrices de transitions ($T_m$, $M_m$) dans le cas $CP_{mto}$ et ($T_o$, $M_o$) dans le cas $CP_{otm}$ ;
- générer le paquetage de configuration VHDL associé au modèle générique du codeur parallèle.

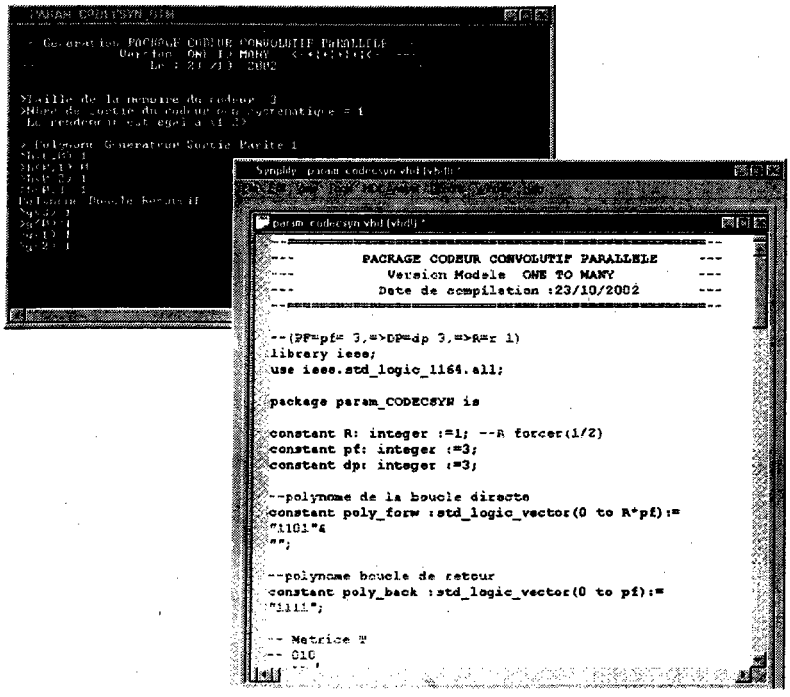



FIG. 5.23 – *Génération du paquetage de configuration du codeur parallèle.*

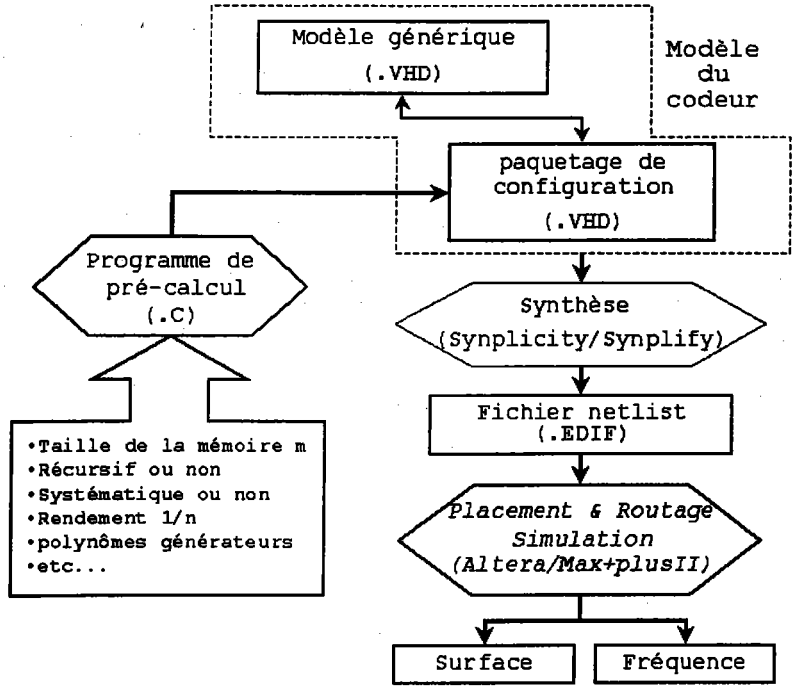



FIG. 5.24 – *Flot de conception.*

La figure 5.24 résume le flot de conception du codeur parallèle.

Les architectures globales des codeurs parallèles RSC $CP_{mto}$ et $CP_{otm}$ pour $m = 3$ et $\wp = 3$ sont représentées sur les figures 5.25 et 5.26, respectivement.

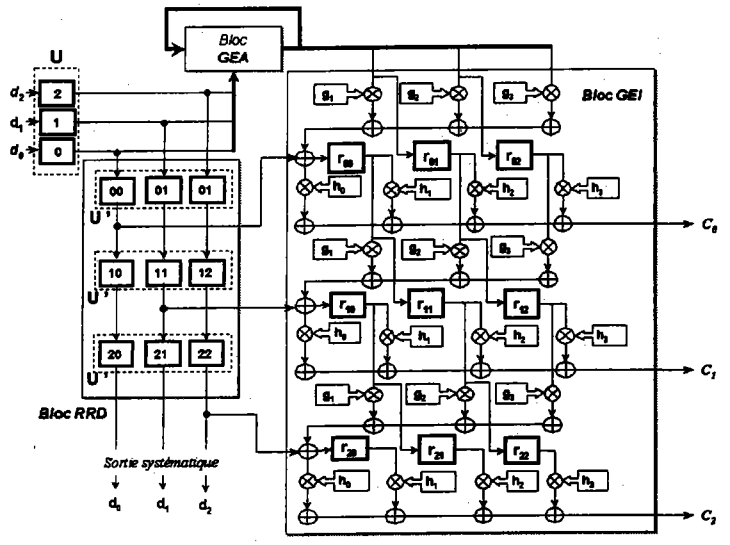



FIG. 5.25 – *Codeur parallèle MTO pour R=1/2, $\wp = 3$ et m=3.*

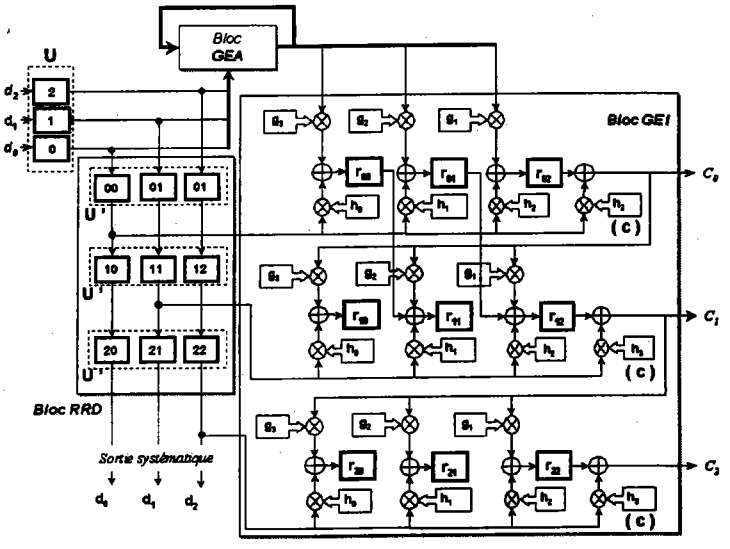



FIG. 5.26 – *Codeur parallèle OTM pour R=1/2, $\wp = 3$ et m=3.*

Pour vérifier les performances de chaque version, le bloc GEA du codeur a été implanté sous forme combinatoire pure et en version mixte « combinatoire/pipeline ». Ce qui fait, deux versions pour chacun des codeurs $CP_{mto}$ et $CP_{otm}$.

$$\texttt{Version combinatoire } (CP1_x) : \begin{cases} {F_1}^* & \rightarrow \ combinatoire \\ {F_2}^* & \rightarrow \ combinatoire \end{cases}$$

$$\texttt{Version mixte } (CP2_x) : \begin{cases} {F_1}^* & \rightarrow \ combinatoire \\ {F_2}^* & \rightarrow \ pipeline \end{cases}$$

La notation $(CP)_x$ signifie $(CP)_{mto}$ ou $(CP)_{otm}$.

Les figures 5.27 et 5.28 représentent les vues RTL des codeurs $CP1_{mto}$ et $CP2_{mto}$, respectivement. Les figures 5.29 et 5.30 représentent les vues RTL des codeurs $CP1_{otm}$ et $CP2_{otm}$, respectivement.

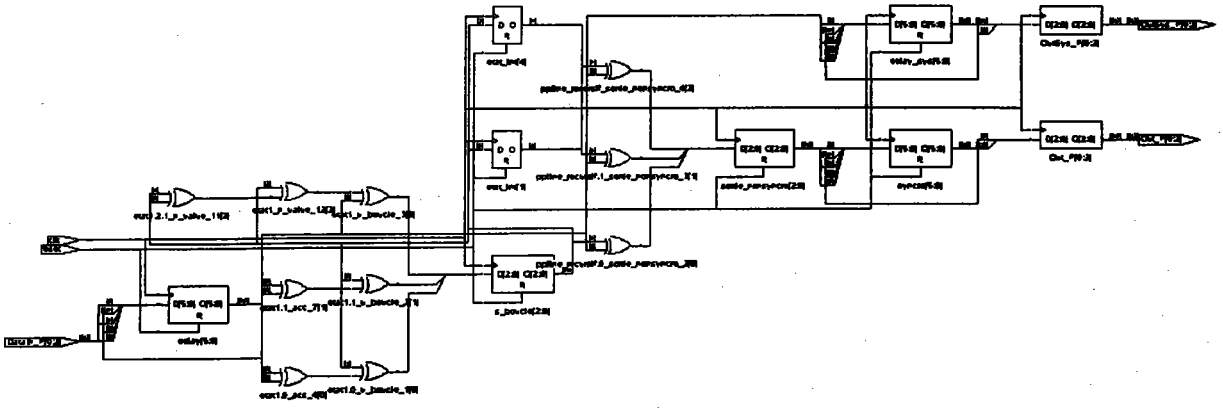


FIG. 5.27 – *Vue RTL du codeur parallèle* $CP1_{mto}$ *avec* $\wp = 3$.

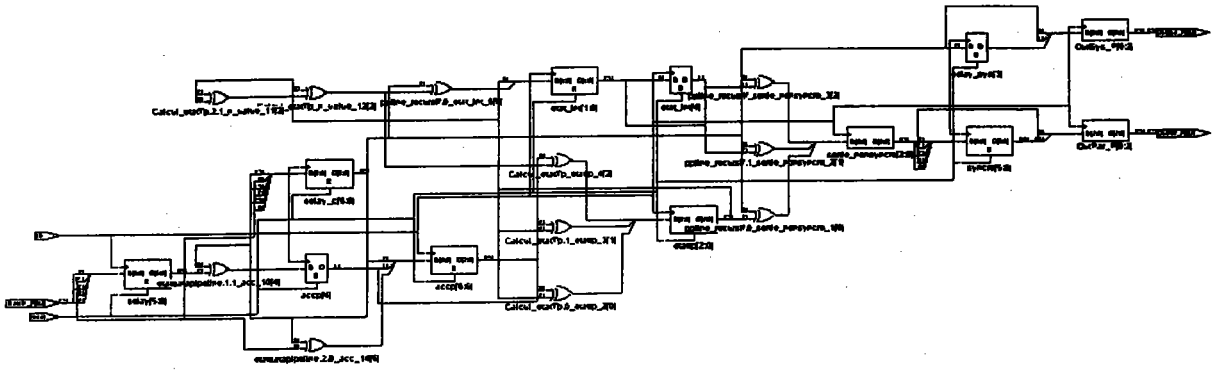


FIG. 5.28 – *Vue RTL du codeur parallèle* $CP2_{mto}$ *avec* $\wp = 3$.

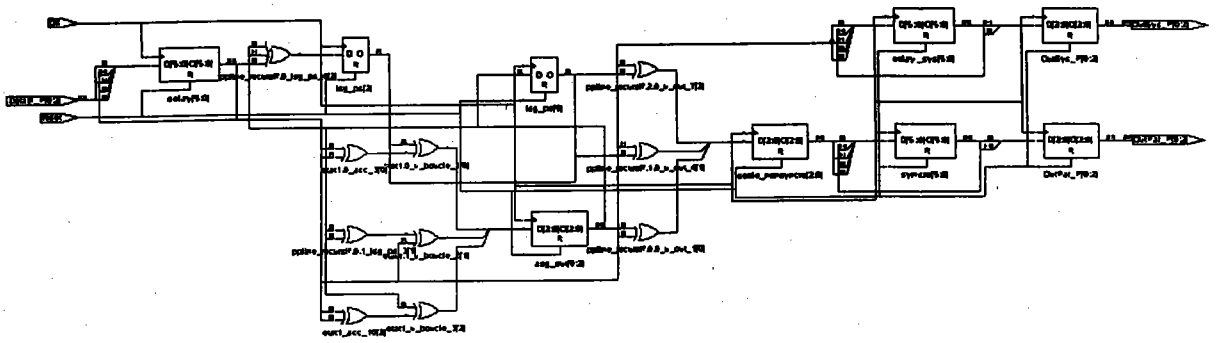


FIG. 5.29 – *Vue RTL du codeur parallèle* $CP1_{otm}$ *avec* $\wp = 3$.

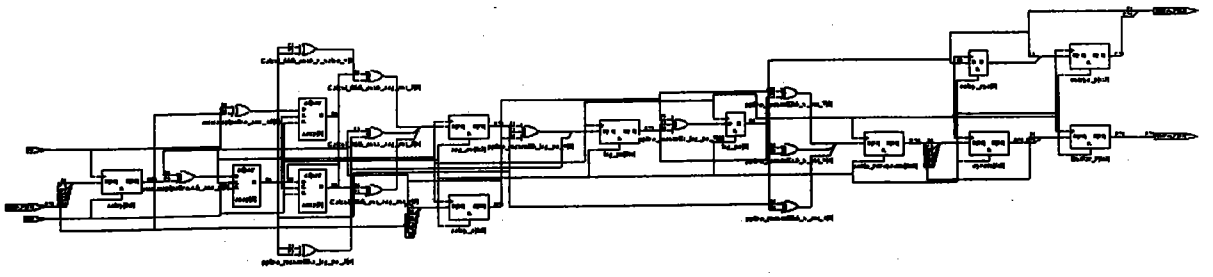


FIG. 5.30 – *Vue RTL du codeur parallèle* $CP2_{otm}$ *avec* $\wp = 3$.

### 5.7.2 Performances du codeur parallèle $CP_{mto}$

Les deux architectures, $CP1_{mto}$ et $CP2_{mto}$ du codeur parallèle MTO, ont été testées pour des codeurs de rendement différents, $1/2$, $1/3$ et $1/4$ sur trois niveaux de parallélisme, 8, 16 et 32 [127]. Les polynômes générateurs des codes sont donnés dans le tableau 5.4. Leur choix est justifié par le fait qu'ils présentent les meilleures distances libres pour une taille de mémoire donnée. Les détails et démonstrations relatifs à leurs choix sont donnés dans [69].

**Notations** Chaque codeur est définit par la notation $(k,n,m)(G,H_1,\cdots,H_i)_8$ où :

- $k$ représente le nombre d'entrées du codeur ;
- $n$ représente le nombre de sorties ;
- $m$ est la taille de la mémoire du codeur ;
- $G$ représente les coefficients du polynôme de la boucle récursive ;
- $H_1,\cdots,H_i$ sont les $i$ polynômes correspondant aux $i$ sorties codées ;
- $()_8$ signifie que les polynômes générateurs sont donnés en octal.

| R=1/2 | R=1/3 | R=1/4 |
|---|---|---|
| $(1,2,2)(7,5)_8$ | $(1,3,3)(7,5,3)_8$ | $(1,4,3)(13,17,15,11)_8$ |
| $(1,2,3)(13,17)_8$ | $(1,3,3)(13,17,15)_8$ | $(1,4,4)(23,35,27,37)_8$ |
| $(1,2,4)(23,33)_8$ | $(1,3,4)(23,25,37)_8$ | $(1,4,4)(23,33,37,25)_8$ |
| $(1,2,4)(23,25)_8$ | - | - |

TAB. 5.4 – *Polynômes générateurs de codes.*

**Courbes de compromis débit/surface**

Les courbes de compromis débit/surface des deux versions $CP1_{mto}$ et $CP2_{mto}$ du codeur parallèle MTO sont reportées sur les figures 5.31, 5.32 et 5.33.

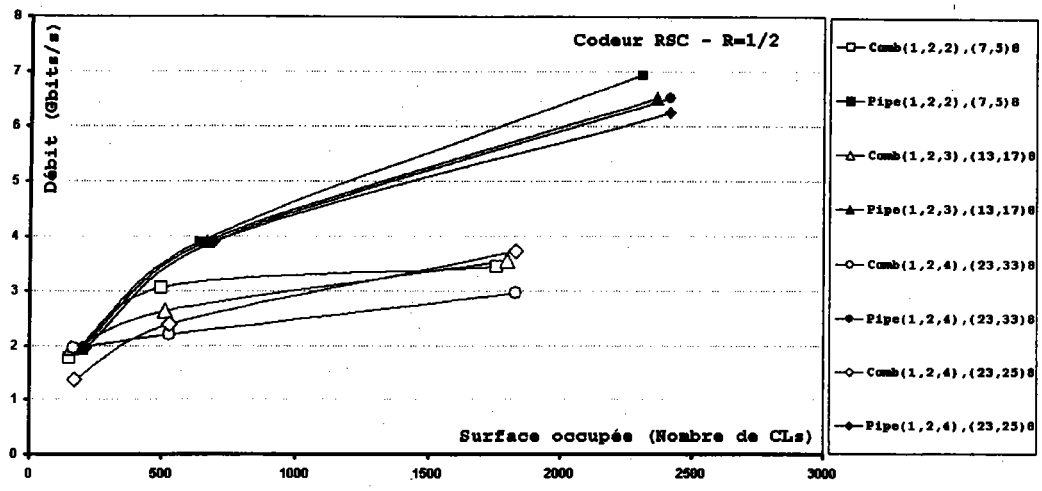



FIG. 5.31 – *Courbes de compromis vitesse/surface pour des codeurs RSC (cas R=1/2).*

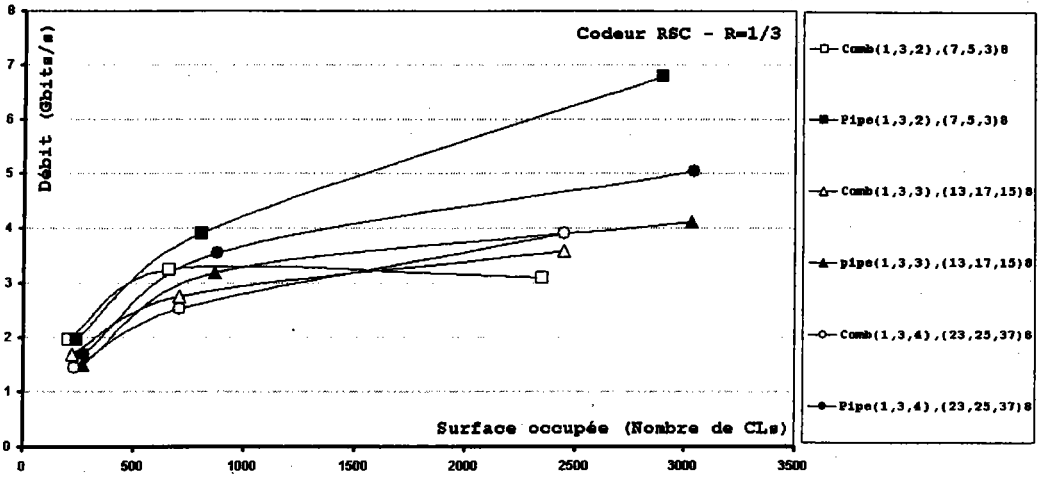



FIG. 5.32 – *Courbes de compromis vitesse/surface pour des codeurs RSC (cas R=1/3).*

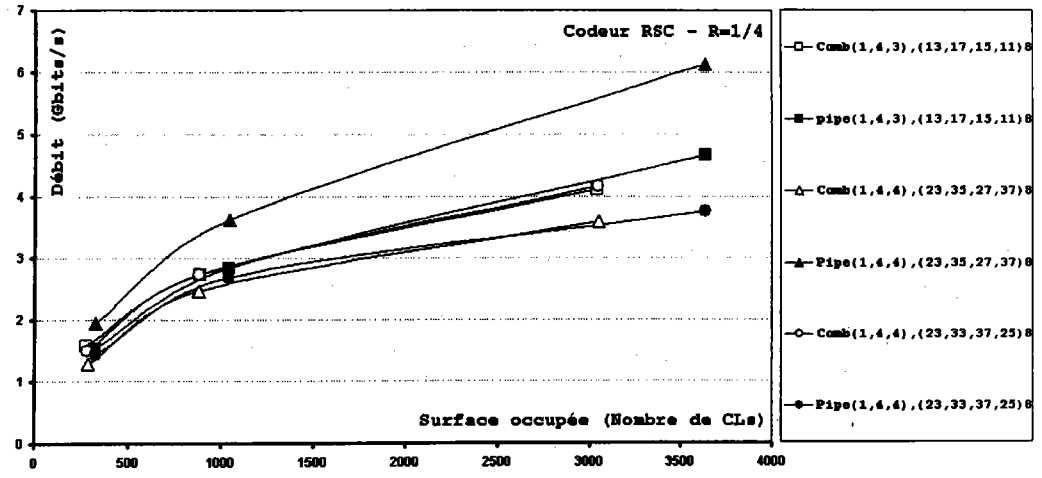



FIG. 5.33 – *Courbes du compromis vitesse/surface pour des codeurs RSC (cas R=1/4).*

Sur ces courbes :

- la surface occupée ou taux d'occupation $S_o$ dans le FPGA est donnée en nombre de cellules logiques $CL$ utilisées.
- le débit $D_\wp$ (nombre de bits traités par seconde) est donné en $Gbits/s$. Il est calculé à partir de l'équation (5.4).

**Analyse du modèle $CP_{mto}$**

Les résultats expérimentaux montrent que les courbes de compromis débit/surface sont favorables à la version $CP2_{mto}$ du codeur parallèle dont le bloc qui réalise la fonction ($F_2$*) a une structure pipeline. Ceci paraît logique puisque le chemin critique dans le version mixte est réduit à celui d'un bloc fonctionnel ($F_1$*), tandis que dans la version combinatoire $CP1_{mto}$, il dépend de tout le bloc GEA.

De l'observation des courbes on peut constater que pour un niveau de parallélisation donné, la surface $S_o$ occupée dans le FPGA est plus importante dans le cas mixte $CP2_{mto}$. Ceci est une conséquence directe du pipeline. En effet, des registres supplémentaires sont utilisés dans la structure pipeline du bloc réalisant ($F_2$*). Cependant, il est important de constater qu'à surface égale, la version mixte $CP2_{mto}$ est plus rapide que la version combinatoire $CP1_{mto}$.

En général, pour un degré de parallélisation fixe, les fréquences de fonctionnement atteintes sont plus élevées dans le cas du codeur mixte. La contrepartie est l'introduction d'une latence supplémentaire qui dépend fortement du degré de parallélisation $\wp$.

Dans le cas d'une implantation combinatoire du bloc GEA, le chemin critique dépend des valeurs à la fois des matrices $M$ et $T^\wp$. Dans le cas de la version mixte il dépend uniquement de la matrice $M$ et plus précisément du nombre maximal de 1 présents dans chaque colonne.

Les fonctions ($F_2$*), liée à l'implantation de la matrice $M$, et ($F_1$*), liée à l'implantation de la matrice $T^{\wp}$, sont les mêmes pour les versions $CP1_{mto}$ et $CP2_{mto}$. Comme dans le cas mixte, le chemin critique dépend uniquement du bloc ($F_1$*), alors le chemin critique du bloc GEA, dans la version $CP1_{mto}$, reste inférieur ou égal à celui du même bloc dans le cas $CP2_{mto}$.

### 5.7.3 Comparaison des performances des codeurs OTM et MTO

Comme le montre l'étude du modèle $CP_{mto}$ dans le paragraphe 5.7.2, la version mixte $CP2_{mto}$ présente les meilleurs performances en terme de compromis débit/surface. Dans cette partie deux versions mixtes $CP2_{mto}$ et $CP2_{otm}$ seront analysées. Elles correspondent à des codeurs RSC de rendement 1/2 d'architectures MTO et OTM, respectivement.

Quatres configurations ont été testées pour chaque modèle pour trois niveaux de parallélisation 8, 16 et 32. Les polynômes générateurs choisis sont donnés dans le tableau 5.5.

| Notation | Degré | Pôlynomes de la boucle récursif |
|---|---|---|
| $G_5(x)$ | 16 | $1+x^2+x^{15}+x^{16}$ |
| $G_6(x)$ | 16 | $1+x^5+x^{12}+x^{16}$ |
| $G_7(x)$ | 32 | $1+x+x^4+x^6+x^8+x^{10}+x^{12}+x^{13}$ $+x^{15}+x^{17}+x^{18}+x^{20}+x^{21}+x^{22}+x^{32}$ |
| $G_8(x)$ | 32 | $1+x+x^2+x^{22}+x^{32}$ |
| **Notation** | **Degré** | **Pôlynomes de la branche directe** |
| $H_1(x)$ | 16 | $1+x+x^2+x^4+x^7+x^9+x^{10}+x^{14}+x^{16}$ |
| $H_2(x)$ | 32 | $1+x+x^2+x^3+x^5+x^{10}+x^{11}+x^{12}+x^{13}+x^{15}+x^{19}+x^{20}$ $+x^{21}+x^{25}+x^{27}+x^{31}+x^{32}$ |

TAB. 5.5 – *Polynômes générateurs.*

Les codeurs parallèles testés correspondent aux fonctions de transfert suivantes :

| Codeur | Fonction de transfert |
|---|---|
| (1,2,16)($G_5(x)$, $H_1(x)$) | $H_1(x)/G_5(x)$ |
| (1,2,16)($G_6(x)$, $H_1(x)$) | $H_1(x)/G_6(x)$ |
| (1,2,32)($G_7(x)$, $H_2(x)$) | $H_2(x)/G_7(x)$ |
| (1,2,32)($G_8(x)$, $H_2(x)$) | $H_2(x)/G_8(x)$ |

TAB. 5.6 – *Codeurs testés.*

Les courbes de la figure 5.34 représentent les fréquences de fonctionnement $f_{mto(\wp)}$ et les surfaces occupées $S_{o,mto}$ en fonction du degré de parallélisme $\wp$ pour une implantation du codeur $CP2_{mto}$.

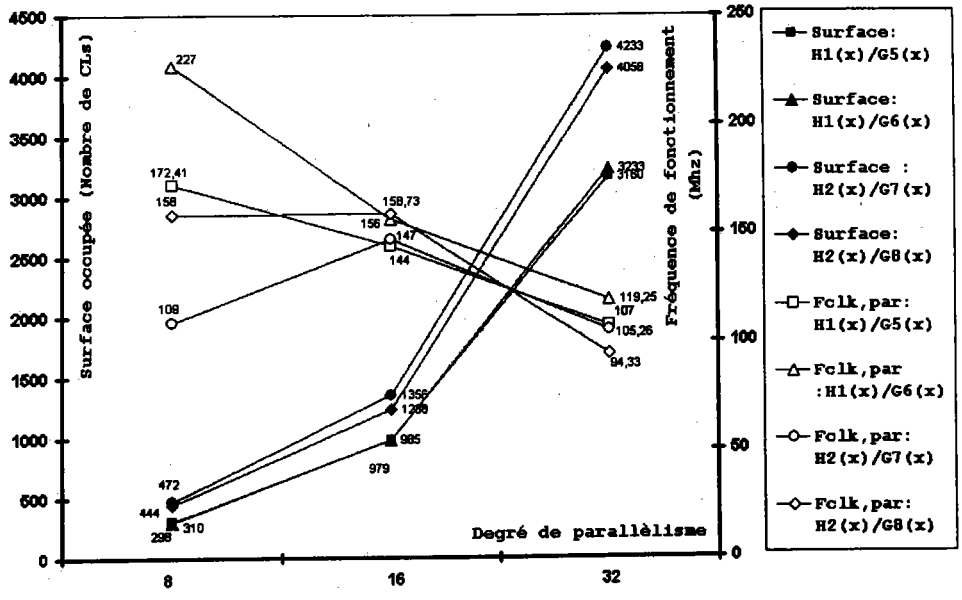



FIG. 5.34 – *(surface, $f_{clk,par}$)/degré de parallélisme pour le codeur parallèle MTO.*

De manière générale, la fréquence de fonctionnement décroît en fonction du degré de parallélisme. Le tableau 5.7 met en évidence la dégradation de la fréquence en pourcentage en fonction du degré de parallélisme, dans le cas du codeur MTO.

| Codeur | $\Delta f_{mto\ (8)}\%$ | $\Delta f_{mto\ (16)}\%$ | $\Delta f_{mto\ (32)}\%$ |
|---|---|---|---|
| $H_1(x)/G_5(x)$ | 31 % | 42 % | 57% |
| $H_1(x)/G_6(x)$ | 9% | 37% | 52% |
| $H_2(x)/G_7(x)$ | 56 % | 41% | 57% |
| $H_2(x)/G_8(x)$ | 36% | 36% | 62% |
| **Moyenne** | $\overline{(\Delta f)}_{mto\ (8)}\%$ | $\overline{(\Delta f)}_{mto\ (16)}\%$ | $\overline{(\Delta f)}_{mto\ (32)}\%$ |
| | 33% | 39% | 57% |

TAB. 5.7 – *Dégradation de $f_{mto\ (\wp)}$ en fonction du degré de parallélisme $\wp$.*

où $\Delta f_{x\ (\wp)}\%$[2] représente le taux de dégradation en pourcentage, pour un degré de parallélisme $\wp$ et une architecture $x$, de la fréquence $f_{x\ (\wp)}$ par rapport à la fréquence maximale de fonctionnement $f_{max}$. Dans le cas présent $f_{max} = 250$ MHz, correspondant à la fréquence de fonctionnement maximale des FPGA de type Altera/Flex10KE. Le taux de dégradation est calculé d'après l'équation (5.52).

$$\Delta f_{x\ (\wp)}\% = \frac{f_{max} - f_{x\ (\wp)}}{f_{max}} \times 100 \tag{5.52}$$

La dégradation moyenne de la fréquence de fonctionnement $\overline{(\Delta f)}_{x\ (\wp)}\%$, pour un degré de parallélisme $\wp$ et une architecture $x$, par rapport à la valeur maximale $f_{max}$ est calculée selon :

2. La notation $\Delta f_{x\ (\wp)}$ signifie $\Delta f_{mto\ (\wp)}$ ou $\Delta f_{otm\ (\wp)}$.

$$\overline{(\Delta f)}_{x\ (\wp)}\% = \frac{\sum_i^n \Delta f_{x\ (\wp)}^i\%}{n} \tag{5.53}$$

où $\Delta f_{x\ (\wp)}^i$ est le taux de dégradation $\Delta f_{x\ (\wp)}$ pour le codeur $i$ et $n$ représente le nombre de codeurs évalués.

On déduit des résultats du tableau 5.7 que la dégradation moyenne de la fréquence $\overline{(\Delta f)}_{mto\ (\wp)}\%$ dans le cas d'un codeur MTO est de :

- 33% pour un gain de 8 cycles d'horloge ;
- 39% pour un gain de 16 cycles d'horloge ;
- 57% pour un gain de 32 cycles d'horloge.

En effectuant le bilan pour chaque degré de parallélisme $\wp$, on obtient l'augmentation effective $A_{eff\ (\wp)}$ du nombre de bits traités par rapport à un traitement série de fréquence $f_{max} = 250$ MHz, où 250 Mbits sont traités chaque seconde.

L'augmentation effective $A_{eff\ (\wp)}$ est calculée d'après l'équation (5.54)[3].

$$A_{eff\ (\wp)} = (1 - \overline{(\Delta f)}_{x\ (\wp)}) \times \wp \tag{5.54}$$

Dans le cas du codeur parallèle MTO, on obtient pour chaque degré de parallélisme les valeurs suivantes :

$$\begin{aligned}
\widetilde{\Delta f}_{mto\ (8)} &\approx 0{,}33 \Rightarrow A_{eff\ (8)} = 0.67 \times 8 = 5{,}36 \\
\widetilde{\Delta f}_{mto\ (16)} &\approx 0{,}39 \Rightarrow A_{eff\ (16)} = 0.61 \times 16 = 9{,}76 \\
\widetilde{\Delta f}_{mto\ (32)} &\approx 0{,}57 \Rightarrow A_{eff\ (32)} = 0.43 \times 32 = 13{,}73
\end{aligned} \tag{5.55}$$

Les courbes de la figure 5.5 représentent, pour les versions des codeurs du tableau 5.7, les fréquences de fonctionnement $f_{otm\ (\wp)}$ et les surfaces occupées $S_{o,otm}$ en fonction du degré de parallélisme $\wp$.

Comme pour la version $CP2_{mto}$, les pourcentages de dégradation de la fréquence de fonctionnement des différents codeurs de type $CP2_{otm}$ sont calculés suivant (5.52) et présentés dans le tableau 5.8.

3. La notation $()_x$ signifie $()_{mto}$ ou $()_{otm}$

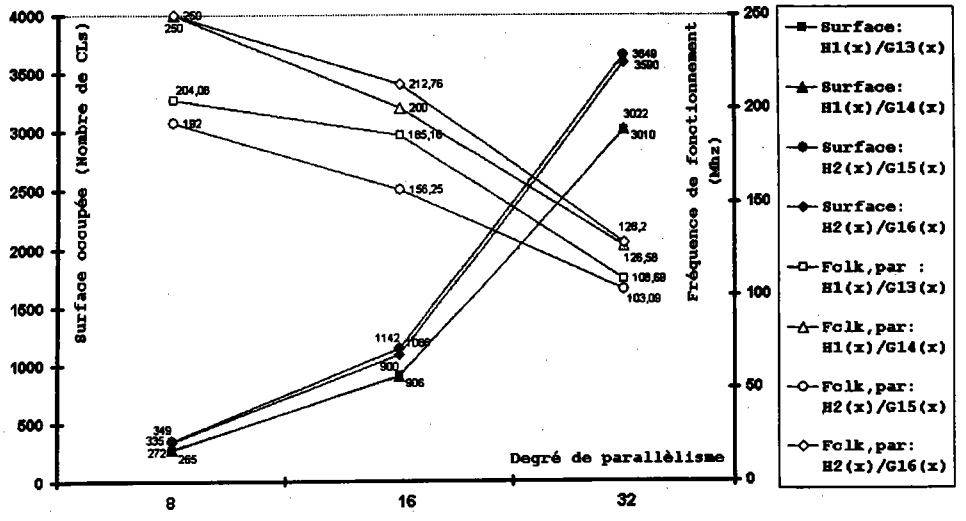



FIG. 5.35 – *(surface, $f_{clk,par}$)/degré de parallélisme pour le codeur parallèle OTM.*

| Codeur | $\Delta f_{otm\ (8)}\%$ | $\Delta f_{otm\ (16)}\%$ | $\Delta f_{otm\ (32)}\%$ |
|---|---|---|---|
| $H_1(x)/G_5(x)$ | 18 % | 25% | 56% |
| $H_1(x)/G_6(x)$ | 0% | 20% | 49% |
| $H_2(x)/G_7(x)$ | 23 % | 37% | 58% |
| $H_2(x)/G_8(x)$ | 0% | 14% | 48% |
| **Moyenne** | $\widetilde{\Delta f}_{otm\ (8)}\%$ | $\widetilde{\Delta f}_{otm\ (16)}\%$ | $\widetilde{\Delta f}_{otm\ (32)}\%$ |
| | 10% | 25% | 53% |

TAB. 5.8 – *Dégradation de $f_{(otm,\ \wp)}$ en fonction du degré de parallélisme $\wp$.*

On déduit des résultats du tableau 5.8 que la dégradation moyenne de la fréquence $\overline{(\Delta f)}_{otm\ (\wp)}\%$ dans le cas d'un codeur OTM est de :

- 10% pour un gain de 8 cycles d'horloge ;
- 25% pour un gain de 16 cycles d'horloge ;
- 53% pour un gain de 32 cycles d'horloge.

En calculant le bilan de la même manière que pour le codeur MTO, on obtient, pour chaque degré de parallélisme, l'augmentation effective du nombre de bits traités (équ. (5.56)) dans le cas du codeur OTM.

$$\begin{aligned}
\widetilde{\Delta f}_{otm\ (8)} &\approx 0{,}10 \Rightarrow A_{eff\ (8)} = 0.9 \times 8 = 7{,}2 \\
\widetilde{\Delta f}_{otm\ (16)} &\approx 0{,}25 \Rightarrow A_{eff\ (16)} = 0.75 \times 16 = 12 \\
\widetilde{\Delta f}_{otm\ (32)} &\approx 0{,}53 \Rightarrow A_{eff\ (32)} = 0.47 \times 32 = 15{,}04
\end{aligned} \tag{5.56}$$

En comparant les valeurs obtenues avec les équations (5.55) et (5.56) des versions $CP2_{mto}$ et $CP2_{otm}$, respectivement, on voit bien que la version $CP2_{otm}$ est plus performante pour les degrés de parallélisme choisis 8, 16, et 32 (équ. (5.57)).

$$A_{eff\ (8,16,32)}(CP2_{mto}) < A_{eff\ (8,16,32)}(CP2_{otm}) \tag{5.57}$$

**Courbes de compromis débit/surface**

Les courbes de compromis débit/surface des deux codeurs sont reportées sur la figure 5.36.

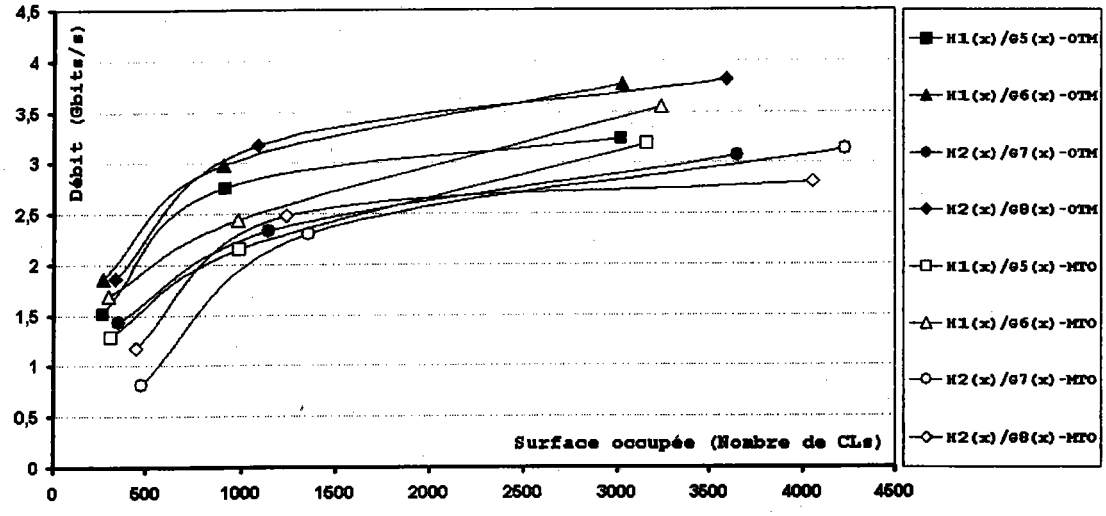



FIG. 5.36 – *Comparatif compromis (vitesse,surface) codeur MTO & OTM.*

Comme on peut le constater, pour un débit donné, la version $CP2_{otm}$ consomme moins de surface. Inversement, pour une surface fixe, le codeur $CP2_{otm}$ fonctionne à des débits plus élevés. Soit, par exemple, la courbe du codeur $(1,2,16)(H_1(x),G_6(x))$ de la figure 5.36. En relevant les valeurs des débits de chaque codeur pour une surface occupée de 2000 cellules logiques, on constate que le débit atteint est plus grand dans le cas OTM (équ. (5.58)). Maintenant, en relevant les valeurs des surfaces occupées de chaque codeur pour un débit de 3,5 Gbits/s on remarque que la version OTM consomme moins de surface (équ. (5.59)).

$$S_{o\ (CP2_{mto})} = S_{o\ (CP2_{otm})} \simeq 2000\ LCs \ \mapsto \ \begin{cases} D_{\wp\ (CP2_{mto})} \simeq 3\ Gbits/s \\ D_{\wp'\ (CP2_{otm})} \simeq 3{,}5\ Gbits/s \end{cases} \tag{5.58}$$

$$D_{\wp\ (CP2_{mto})} = D_{\wp'\ (CP2_{otm})} \simeq 3{,}5\ Gbits/s \ \mapsto \ \begin{cases} S_{o\ (CP2_{mto})} \simeq 3250\ LCs \\ S_{o\ (CP2_{otm})} \simeq 2000\ LCs \end{cases} \tag{5.59}$$

**Analyse du modèle $CP_{otm}$**

Dans la version $CP_{otm}$ du codeur, le bloc GEI ne possède aucun chemin critique[4]. Le chemin critique se situe uniquement au niveau du bloc GEA. La complexité de ce dernier, qui execute la fonction ($F_1$*), dépend de la matrice $T^{\wp}{}_o$. Par conséquent, dans la version $CP2_{otm}$ du codeur, le chemin critique dépend uniquement de la matrice $T^{\wp}{}_o$ et du degré de parallélisme $\wp$.

Dans le cas $CP_{mto}$, les deux blocs GEA et GEI peuvent contenir le chemin critique selon la taille $m$ de la mémoire du codeur. En effet, pour de faibles valeurs de $m$, le chemin critique est localisé dans le bloc GEA réalisant la fonction ($F_1$*) et dépend de l'implantation de la matrice $T^{\wp}{}_m$. Pour de grandes valeurs de $m$ la complexité du bloc GEI augmente. Dans ce cas le chemin critique dépend de l'implantation des matrices $T^{\wp}{}_m$ de la fonction ($F_1$*) et $T_m$.

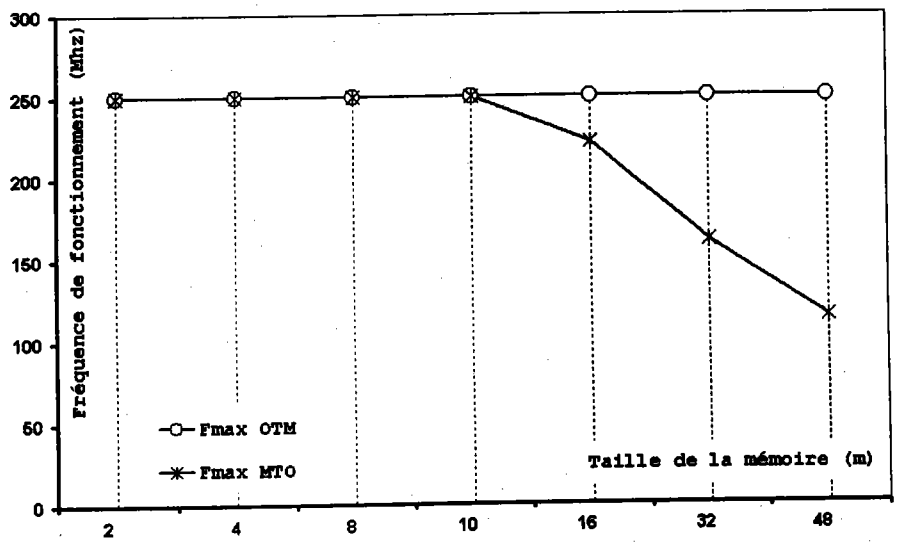



FIG. 5.37 – *Fréquence de fonctionnement du bloc GEI en fonction du degré de parallélisme $\wp$.*

La figure 5.37 représente la fréquence de fonctionnement du bloc GEI, des codeurs $CP2_{mto}$ et $CP2_{otm}$ équivalents, en fonction du degré de parallélisme $\wp$. On voit clairement, dans le cas MTO, que plus $m$ augmente, plus la fréquence de fonctionnement $f_{mto}$ se dégrade. Alors que dans le cas OTM, la fréquence $f_{otm}$ est indépendante du degré de parallélisme $\wp$.

4. Le chemin critique dans un codeur OTM se limite à une porte XOR à 3 entrées (fig. 5.10 et 5.26).

Le tableau 5.9 donne la localisation des chemins critiques dans les codeurs parallèles MTO et OTM. Le tableau 5.10 récapitule les temps de latence de chaque type de codeur[5].

| | $CP1_{mto}$ | $CP2_{mto}$ | $CP1_{otm}$ | $CP2_{otm}$ |
|---|---|---|---|---|
| **Bloc GEA** - $T^{\wp}$ | + | + | + | + |
| **Bloc GEA** - $M$ | + | - | + | - |
| **Bloc GEI** - $T$ | + | + | - | - |
| **Bloc RRD** | - | - | - | - |

\+ : Chemin critique possible
\- : Aucun chemin critique[a]

TAB. 5.9 – *Localisation des chemins critiques dans les codeurs parallèles MTO et OTM.*

GEA : Génération des états anticipés.
GEI : Génération des états intermédiaires.
RRD : Réseau de registre à décalage.

[a] Dans ce cas, le chemin critique de ce bloc n'est pas le plus long.

| Codeur | Latence | $\wp \nearrow$ |
|---|---|---|
| $CP1_x$ | $(\wp+2) = \wp \cdot (1+\frac{2}{\wp})$ | $\simeq \wp$ |
| $CP2_x$ | $\wp + (\wp+2) = \wp + \wp \cdot (1+\frac{2}{\wp})$ | $\simeq 2 \cdot \wp$ |

TAB. 5.10 – *Temps de latence des codeurs parallèles MTO et OTM.*

## 5.8 Simulation des codeurs parallèles $CP_{mto}$ et $CP_{otm}$

Les simulations des deux codeurs parallèles $CP_{mto}$ et $CP_{otm}$ ont été effectuées avec la version 10.0 de MAX+plus II d'Altera. Les Chronogrammes de simulation des quatre versions $CP1_{mto}$, $CP2_{mto}$, $CP1_{otm}$ et $CP2_{otm}$ permettent de mieux voir l'influence du pipeline sur le temps de réponse des codeurs (dont le bloc exécutant la fonction ($F_1{}^*$) a une structure pipeline). Les chronogrammes des codeurs série MTO et OTM équivalents permettent de comparer les valeurs de leur sortie avec celles des codeurs parallèles correspondants.

**Remarque** Pour une taille de mémoire donnée $m$, les codeurs séries MTO et OTM sont dits équivalents s'ils génèrent les même sorties codées pour des entrées similaires. Pour cela leurs polynômes générateurs respectifs (récursif et directe) $P_{otm}(x)$ and $P_{mto}(x)$ doivent satisfaire à l'équation (2.8) [128], ce qui donne :

5. La notation $()_x$ dans le tableau 5.10 signifie $()_{mto}$ ou $()_{otm}$.

$$\left.\begin{array}{rcl} P_{mto}(x) & = & a_0 + a_1x + a_2x^2 + \cdots + a_mx^m \\ P_{otm}(x) & = & b_0 + b_1x + b_2x^2 + \cdots + b_mx^m \end{array}\right\} \rightarrow b_i = a_{m-i}, \forall i \in \{0,1,\cdots,m\}. \quad (5.60)$$

Les codeurs simulés dans cette partie sont[6] :

$$(1,2,3)(17,15)_8 \mapsto \left\{\begin{array}{l} CP1_{mto} \\ CP2_{mto} \\ CS_{mto} \end{array}\right.$$

$$(1,2,3)(17,13)_8 \mapsto \left\{\begin{array}{l} CP1_{otm} \\ CP2_{otm} \\ CS_{otm} \end{array}\right. \quad (5.61)$$

Les courbes correspondant aux architectures séries et parallèles des codeurs MTO et OTM sont référencées comme indiqué dans le tableau 5.11.

| | | |
|---|---|---|
| CLK | : | signal d'horloge. |
| | | **Codeur Série OTM** |
| Input_OTM | : | entrée des données du codeur série OTM, $CS_{otm}$. |
| Out_OTM | : | sortie non systématique du codeur série OTM, $CS_{otm}$. |
| OutSys_OTM | : | sortie systématique du codeur série OTM, $CS_{otm}$. |
| | | **Codeur Série MTO** |
| Input_MTO | : | entrée des données du codeur série MTO, $CS_{mto}$. |
| Out_MTO | : | sortie non systématique du codeur série MTO, $CS_{mto}$. |
| OutSys_MTO | : | sortie systématique du codeur série MTO, $CS_{mto}$. |
| | | **Codeur parallèle MTO & OTM** |
| reset | : | signal de remise à zéro |
| DataIn_P | : | bus d'entrée des données de taille $\wp$. |
| OutSys_P | : | bus de la sortie systématique de taille $\wp$. |
| OutPar_P | : | bus de la sortie non systématique de taille $(n-1)\cdot\wp$. |

TAB. 5.11 – *Légende des courbes de simulations.*

Les codeurs parallèles ont été simulés pour un degré de parallélisme $\wp = 8$. La trame d'information test choisie pour tester le codeur est la séquence binaire $T_{inf}$.

$E_s$ : entrée série.

$E_\wp$ : entrée parallèle.

$()_{16}$ : représentation hexadécimale.

6. Pour les notations se référer à 5.7.2.

$$
\begin{array}{lcccccccc}
E_s = T_{inf} & = & \underbrace{1000\,0000} & \underbrace{0100\,0000} & \underbrace{1100\,0000} & \underbrace{0010\,0000} & \underbrace{1010\,0000} & \underbrace{0110\,0000} & \cdots \\
 & = & \overbrace{8\,0} & \overbrace{4\,0} & \overbrace{c\,0} & \overbrace{2\,0} & \overbrace{a\,0} & \overbrace{6\,0} & \cdots \\
 & & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \cdots \\
\text{Découpage} & \Rightarrow & [7..0] & [7..0] & [7..0] & [7..0] & [7..0] & [7..0] & \cdots \\
 & & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \cdots \\
E_\wp = T_{inf} & = & \overbrace{(8\,0)_{16}} & \overbrace{(4\,0)_{16}} & \overbrace{(c\,0)_{16}} & \overbrace{(2\,0)_{16}} & \overbrace{(a\,0)_{16}} & \overbrace{(6\,0)_{16}} & \cdots
\end{array}
$$

### 5.8.1 Simulation du codeur *Many To One*

La figure 5.38 présente la simulation du codeur série MTO.

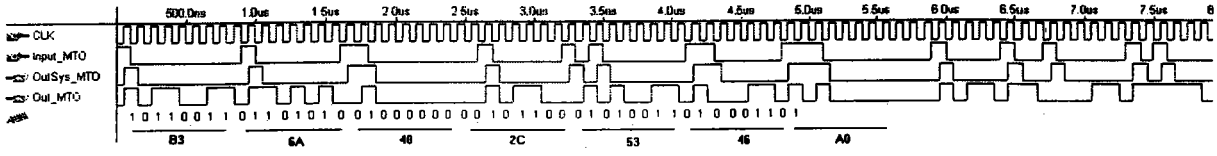



FIG. 5.38 – *Simulation du codeur RSC série* Many To One ($CS_{mto}$).

Les figures 5.39 et 5.40 présentent les courbes de simulations obtenues pour respectivement, les codeurs parallèles $CP1_{mto}$ et $CP2_{mto}$.

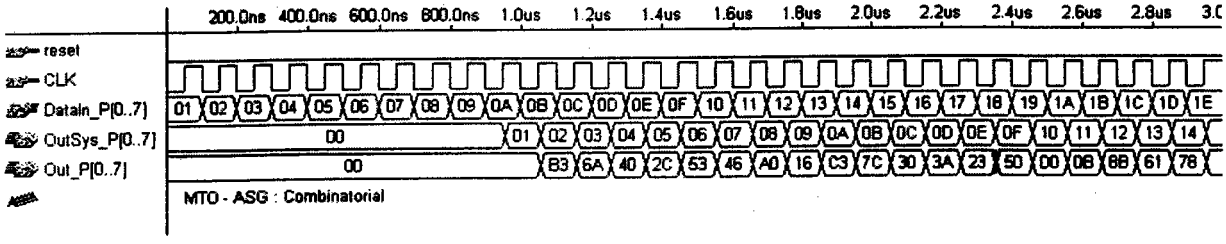



FIG. 5.39 – *Simulation du codeur RSC parallèle MTO* ($CP1_{mto}$) - (*cas*) *GEA combinatoire.*

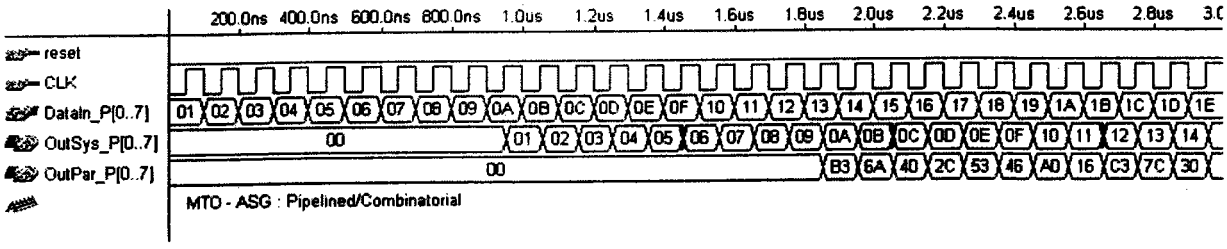



FIG. 5.40 – *Simulation du codeur RSC parallèle MTO* ($CP2_{mto}$) - (*cas*) *GEA mixte.*

En plus du temps de latence induit par la parallélisation du codeur, qui est de $p + 2$ cycles d'horloge[7], on remarque, dans le cas $CP2_{mto}$, l'apparition d'un temps de latence supplémentaire généré

7. Les 2 cycles additionnels sont dus à l'introduction de registres de synchronisation lors de la description VHDL.

par le pipeline de la fonction ($F_2^*$) dans le bloc GEA. Sa valeur est ici de 8 cycles d'horloge. Dans le cas général cette valeur de latence additionnelle est égale au degré de parallélisme $\wp$.

### 5.8.2 Simulation du codeur *One To Many*

Les courbes de simulation du codeur OTM série sont présentées par la figure 5.41. Il est évident que l'on obtient les mêmes sorties codées que dans le cas MTO puisque le codeur OTM est équivalent (équ. (5.60)).

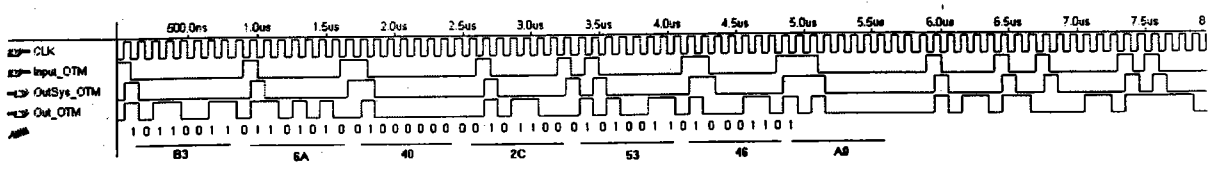



FIG. 5.41 – *Simulation du codeur RSC série* One To Many ($CS_{otm}$).

Les figures 5.42 et 5.43 représentent respectivement les courbes de simulations obtenues pour les codeurs parallèles $CP1_{otm}$ et $CP2_{otm}$.

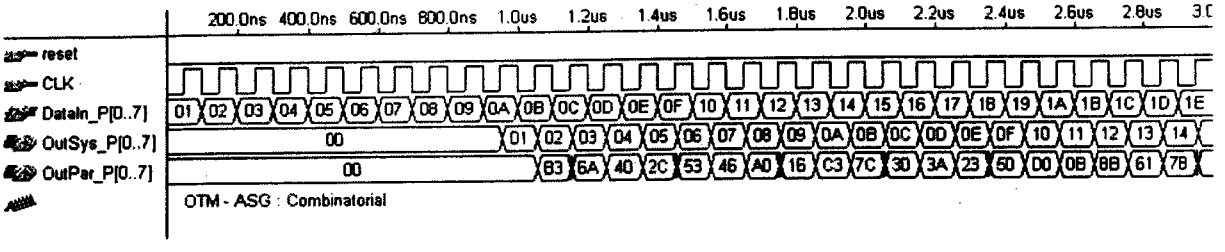



FIG. 5.42 – *Simulation du codeur RSC parallèle OTM* ($CP1_{otm}$) - (*cas*) *GEA combinatoire.*

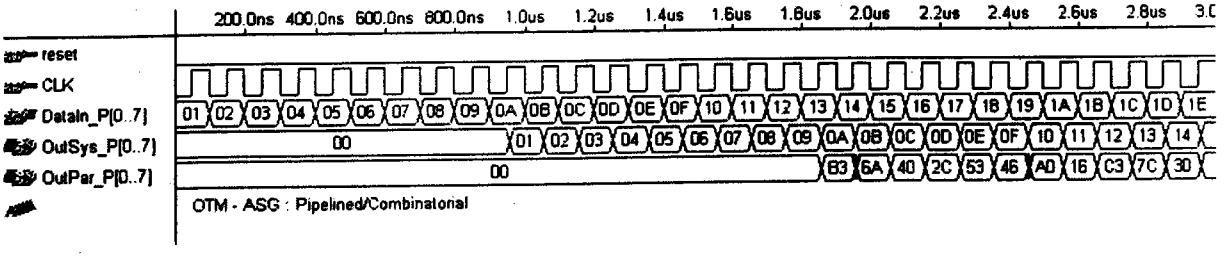



FIG. 5.43 – *Simulation du codeur RSC parallèle OTM* ($CP2_{otm}$) - (*cas*) *GEA mixte.*

Les commentaires du cas $CP_{mto}$ restent valides (cf. paragraphe 5.8.1).

## 5.9 Application aux turbo-codes

Avant de pouvoir être utilisés comme codes constituants, les codeurs RSC parallèles, ont été adaptés pour gérer la phase de terminaison du treillis. Du codeur de la figure 5.44, on déduit les équations architecturales qui régissent le fonctionnement normal (codage) et la phase de terminaison du treillis.

$$(R)_{t+1} = \begin{cases} (R)_t \cdot T + ((u)_t, 0, \cdots, 0) & \leftarrow \quad tt = 0 \\ (R)_t \cdot J & \leftarrow \quad tt = 1 \end{cases} \quad \text{avec } J = \begin{pmatrix} 0 & 1 & \cdot & 0 \\ 0 & 0 & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & \cdot & 1 \\ 0 & 0 & \cdot & 0 \end{pmatrix}. \tag{5.62}$$

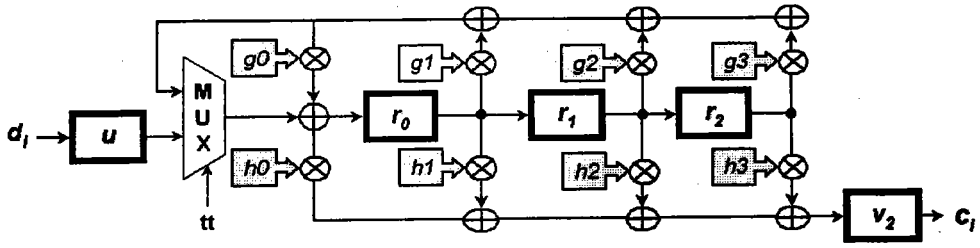



FIG. 5.44 – *Terminaison du treillis.*

En suivant la même stratégie de parallélisation précédente que pour l'architecture série d'un codeur MTO, on aboutit aux équations architecturales qui décrivent le fonctionnement du bloc GEA durant la phase normale (équ. (5.63) : $tt = 0$) et la phase de terminaison du treillis (équ. (5.63) : $tt = 1$).

$$(R)_{t+1} = \begin{cases} (R)_t \cdot T^{\wp} + (U)_t \cdot M & \leftarrow \quad tt = 0 \\ (R)_t \cdot J^{\wp} & \leftarrow \quad tt = 1 \end{cases} \tag{5.63}$$

Le fonctionnement du bloc GEI est décrit pour les phases de codage et de terminaison du treillis les équations ((5.64) : $(w_k)_t = 0$) et ((5.64) : $(w_k)_t = 1$), respectivement.

$$(R_k)_{t+1} = \begin{cases} (R_{k-1})_t \cdot T + ((u'_{k,k})_t, 0, \cdots, 0) & \leftarrow \quad (w_k)_t = 0 \\ (R_{k-1})_t \cdot J & \leftarrow \quad (w_k)_t = 1 \end{cases} \tag{5.64}$$

La figure 5.45 représente le bloc GEA avec terminaison du treillis.

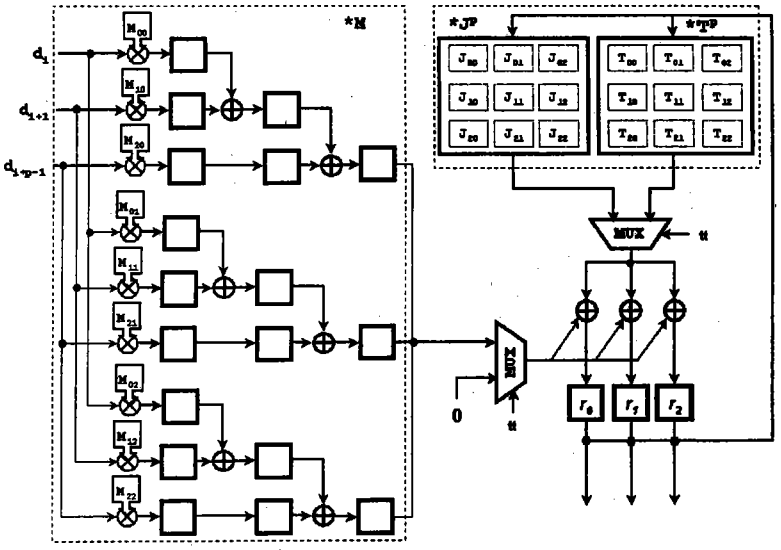



FIG. 5.45 – *Bloc GEA avec terminaison du treillis.*

La figure 5.46 représente les blocs RRD et GEI avec terminaison du treillis.

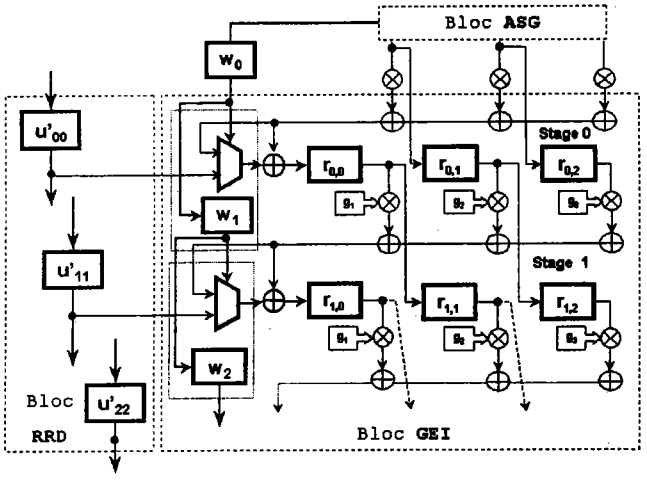



FIG. 5.46 – *Bloc GEI et RRD avec terminaison du treillis.*

Le turbo-code implanté est un turbo-code parallèle à deux niveaux avec un rendement[8] $R = 1/3$. Il est représenté sur la figure 5.47 dans la cas d'un degré de parallélisme $\wp = 8$. il est constitué de deux codes identiques de type RSC de rendement $1/2$. L'entrelaceur utilisé est un entrelaceur circulaire, sa description est donnée dans le paragraphe 2.8.1.

8. En pratique dans le cas d'une transmission sur un canal AWGN, la sortie systématique du deuxième code constituant n'est pas transmise. Cependant, elle peut être transmise pour doubler la diversité en présence d'évanouissements dans le cas d'un canal de Rayleigh [3].

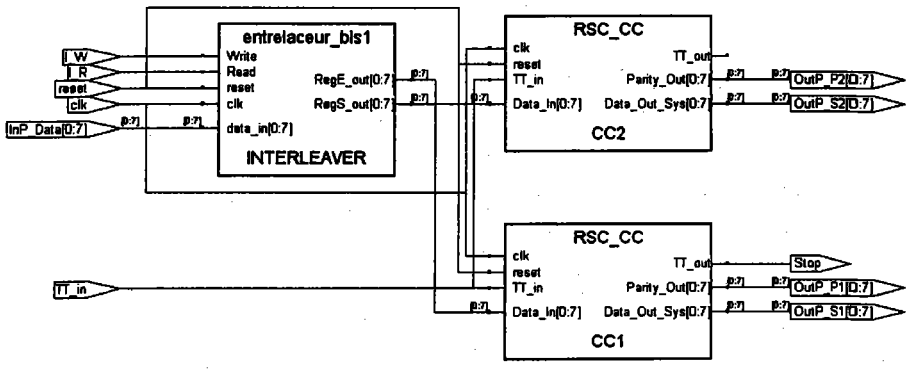



FIG. 5.47 – *Vue RTL du turbo-codeur pour $\wp = 8$ et R=1/4.*

Une architecture a été implantée pour chaque modèle de turbo-codeur, pour chacun des polynômes générateurs donnés dans le tableau 5.12.

| | $(k,n,m)(G(x),H(x))_8$ |
|---|---|
| TC1 | $(1,2,2)(7,5)_8$ |
| TC2 | $(1,2,3)(13,17)_8$ |
| TC3 | $(1,2,4)(23,33)_8$ |

TAB. 5.12 – *Polynômes générateurs.*

Les courbes de compromis débit/surface correspondant à chaque codeur sont données pour trois niveaux de parallélisme ($\wp$=8, 16 et 32) et trois tailles de matrice d'entrelacement ($K$=256, 512 et 1024).

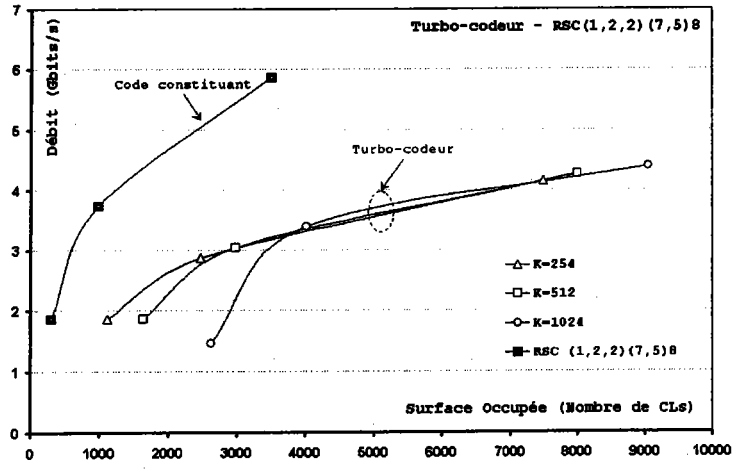



FIG. 5.48 – *Courbes de compromis débit/surface pour TC1.*

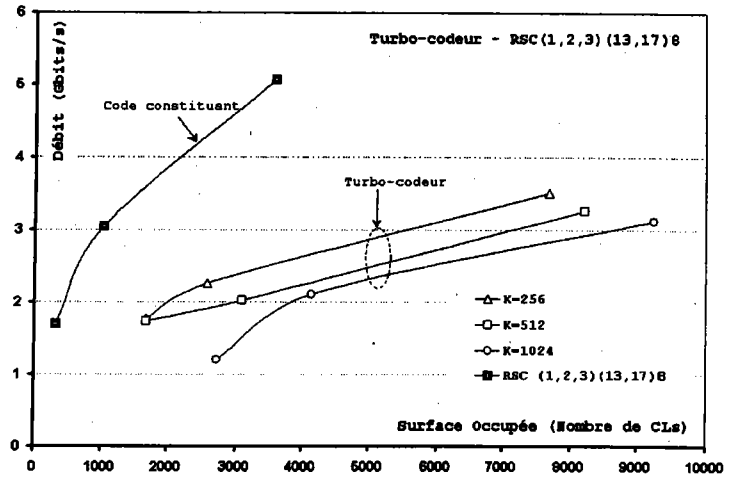



FIG. 5.49 – *Courbes de compromis débit/surface pour TC2.*

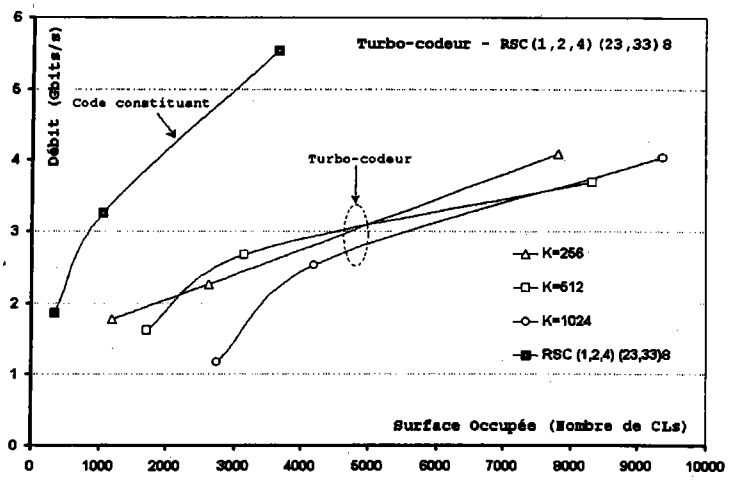



FIG. 5.50 – *Courbes de compromis débit/surface pour TC3.*

## 5.10 Conclusion

Ce chapitre présente une architecture rapide pour les codeurs convolutifs (récursifs et non récursifs) et son application à un turbo-code. La structure globale des codeurs est basée sur des blocs fonctionnant de manière parallèle. Les blocs sont implantés selon une structure combinatoire ou pipeline quand cela est possible. Quatre architectures du codeur convolutif ont été testées pour différents degrés de parallélisme, polynômes générateurs et rendements. Tous les modèles RTL ont été décrits en VHDL. Ils sont indépendants d'une technologie cible et configurables avant la synthèse. Le module de configuration est généré automatiquement par un programme écrit en C à partir des paramètres ($\wp$, $m$, $R$, $G(x)$, $H_i(x)$, etc.). Les débits obtenus pour des implantations sur FPGA de type Flex10KE

sont compris, toutes versions confondues, entre 1 Gbits/s et 7.45 Gbits/s pour des degrés de parallélisme $\wp = 8$ et $\wp = 32$, respectivement (cf. annexe D). Les versions $CP1_{otm}$ et $CP1_{mto}$ sont moins performantes que les $CP2_{otm}$ et $CP2_{mto}$. Par contre, elles consomment moins de surface. La version $CP2_{otm}$ du codeur parallèle présente les meilleurs débits pour des codeurs de grande taille de mémoire . Enfin, plusieurs configurations d'un turbo-codeur parallèle à deux niveaux (constitué de deux codes RSC de rendement $R = 1/2$) avec un rendement $R = 1/3$ ont été testées. Les résultats obtenus[9] après implantation sur FPGA sont compris entre 1 et 2 Gbits/s pour un degré de parallélisme $\wp = 8$ et entre 3 et 5 Gbits/s pour $\wp = 32$ (cf. annexe E).

9. En considérant les polynômes générateurs donnés dans le tableau 5.12.

# Chapitre 6

# Décodeur de syndrome haut débit

Comme cela a été défini dans le paragraphe 1.8.2, un décodeur de syndrome est constitué d'un codeur, d'un registre de syndrome à partir duquel une décision est prise sur la validité du bit testé et de quelques additionneurs modulo-2. Le principal avantage de ce type de décodeur réside dans sa mise en œuvre simple et dans sa rapidité de décodage. Ceci le rend mieux adapté aux applications hauts débits que les décodeurs de type Viterbi ou séquentiels [12]. Dans ce chapitre nous proposons une architecture parallèle du décodeur de syndrome. La parallélisation de la structure du décodeur de syndrome repose sur la parallélisation de son codeur et de son registre de syndrome. Le bloc codeur étant traité dans le chapitre 5 il ne reste plus qu'à étudier le bloc registre de syndrome.

Ce chapitre est donc consacré à l'étude de la parallélisation du registre de syndrome. Après une étude de l'architecture série d'un décodeur de syndrome, la structure parallèle générale du décodeur est présentée. La dernière partie présente les résultats expérimentaux de l'implantation sur des FPGA de type Flex10KE d'Altera.

**Mots clés** décodeurs à logique majoritaire, codes convolutifs, architectures parallèles.

## 6.1 Introduction

Si l'on considère un code systématique de rendement $R = 1/2$, les équations de codage d'une trame d'information $D(x) = d_0 + d_1x + d_2x^2 + \cdots + d_ix^i$ sont données par les équations (6.1) et (6.2) associées aux sorties systématique et parité, respectivement.

$$D(x) \cdot H_0(x) = D(x) = d_0 + d_1x + d_2x^2 + \cdots + d_ix^i \quad (6.1)$$

$$D(x) \cdot H_1(x) = C(x) = c_0 + c_1x + c_2x^2 + \cdots + c_ix^i \quad (6.2)$$

où $H_0(x) = 1$ représente le polynôme générateur de la sortie systématique et $H_1(x) = h_{0,1} +$

$h_{1,1}x + h_{1,2}x^2 + \cdots + h_{1,m}x^m$ le polynôme générateur de la sortie de parité.

Après multiplexage, la séquence émise sur le canal de transmission est $T(x)$.

$$D(x),C(x) \rightarrow \text{Fmux}[D(x),C(x)] = T(x) = D(x^2) + x \cdot C(x^2) \tag{6.3}$$

avec, $T(x) = d_0 + c_0x + d_1x^2 + c_1x^3 + d_2x^4 + \cdots + d_ix^{2i} + c_ix^{2i+1}$.

Au niveau du récepteur la séquence reçue est $\widetilde{T}(x)$.

$$\text{Fdemux}[\widetilde{T}(x)] = \text{Fdemux}[\widetilde{D}(x^2) + x \cdot \widetilde{C}(x^2)] \rightarrow \widetilde{D}(x),\widetilde{C}(x) \tag{6.4}$$

où Fmux et Fdemux sont les fonctions de multiplexage et démultiplexage, respectivement, et

$$\begin{aligned}
\widetilde{T}(x) &= \widetilde{d}_0 + \widetilde{c}_0x + \widetilde{d}_1x^2 + \widetilde{c}_1x^3 + \widetilde{d}_2x^4 + \cdots + \widetilde{d}_ix^{2i} + \widetilde{c}_ix^{2i+1} \\
\widetilde{D}(x) &= \widetilde{d}_0 + \widetilde{d}_1x + \widetilde{d}_2x^2 + \cdots + \widetilde{d}_ix^i \\
\widetilde{C}(x) &= \widetilde{c}_0 + \widetilde{c}_1x + \widetilde{c}_2x^2 + \cdots + \widetilde{c}_ix^i
\end{aligned}$$

Les notations $\widetilde{D}(x)$ et $\widetilde{C}(x)$ représentent respectivement les trames systématique et de parité entachées d'erreurs $E^d(x)$ et $E^c(x)$, (équ. (6.5) et (6.6)).

$$\widetilde{D}(x) = D(x) + E^d(x) \tag{6.5}$$

$$\widetilde{C}(x) = C(x) + E^c(x) \tag{6.6}$$

avec

$$\begin{aligned}
E^d(x) &= e_0^d + e_1^dx + e_2^dx^2 + \cdots + e_i^dx^i. \\
E^c(x) &= e_0^c + e_1^cx + e_2^dx^2 + \cdots + e_i^cx^i.
\end{aligned}$$

En remplaçant $D(x)$ dans l'équation (6.5) par son expression donnée dans l'équation (6.1) et $C(x)$ dans (6.6) par son expression donnée dans l'équation (6.2), on obtient :

$$\widetilde{D}(x) = D(x) \cdot H_0(x) + E^d(x) \tag{6.7}$$

$$\widetilde{C}(x) = D(x) \cdot H_1(x) + E^c(x) \tag{6.8}$$

La séquence de syndrome est calculée d'après l'équation (6.9).

$$\begin{aligned}
\Psi(x) &= \widetilde{D}(x) \cdot H_1(x) + \widetilde{C}(x) \\
&= \widehat{C}(x) + \widetilde{C}(x)
\end{aligned} \tag{6.9}$$

En remplaçant $\widehat{C}(x)$ $\widetilde{C}(x)$ dans l'équation (6.9) par leurs expressions données dans les équations (6.7) et (6.8) repectivement, on obtient :

$$\begin{aligned} \Psi(x) &= [D(x) \cdot H_0(x) + E^d(x)] \cdot H_1(x) + D(x) \cdot H_1(x) + E^c(x), && (6.10) \\ &= E^d(x) \cdot H_1(x) + E^c(x) && (6.11) \end{aligned}$$

On déduit de l'équation (6.11) que la séquence de syndrome $\Psi(x)$ calculée ne dépend pas des bits transmis mais uniquement des erreurs introduites dans le canal de transmission. Par conséquent, le syndrome donne une information sur la présence ou non d'erreurs dans le canal.

Le décodage de syndrome est basé sur le calcul des syndromes « $\Psi_i$ » qui correspondent au bit évalué. Le syndrome $\Psi_i$ permet de tester la validité d'un bit reçu $\widetilde{d}_i$ selon l'équation (6.12).

$$\begin{cases} \Psi_i &= \{\psi_{i+m}, \psi_{i+m-1}, \cdots, \psi_i\} \rightarrow bit\ \widetilde{d}_i \\ i &\in \{0,1,\cdots\} \end{cases} \tag{6.12}$$

La détection[1] de la présence d'un bit erroné se fait grâce à une fonction d'évaluation $F_d$ appliquée à une combinaison de bits $\psi_i$ du syndrome $\Psi_i$, équ. (6.13). Cette combinaison $\Psi^*{}_i$ est construite selon une règle prédéfinie.

$$\widehat{e}_i = F_d(\Psi^*{}_i) \tag{6.13}$$

La fonction de décision $F_d$ fournit une valeur approximative $\widehat{e}_i$ de l'erreur. Dans le corps de Galois CG(2), la sortie $\widehat{e}_i$ de $F_d$ est un 1 lors de la detection d'une erreur et 0 dans le cas contraire. Finalement, la valeur $\widehat{d}_i$ corrigée[2] du bit détecté erroné s'obtient par simple addition du bit reçu $\widetilde{d}_i$ avec l'erreur évaluée $\widehat{e}_i$, équ. (6.14).

$$\begin{aligned} \text{erreur détectée} &\rightarrow \widehat{d}_i = \widetilde{d}_i + (\widehat{e}_i = 1) = \overline{\widetilde{d}_i} \\ \text{erreur non détectée} &\rightarrow \widehat{d}_i = \widetilde{d}_i + (\widehat{e}_i = 0) = \widetilde{d}_i \end{aligned} \tag{6.14}$$

Le décodeur série calcule un nouveau syndrome $\Psi_i$ par cycle d'horloge. Simultanément, un bit reçu est évalué. L'approche proposée consiste à calculer $\wp$ syndromes à chaque cycle d'horloge puis à les réorganiser à travers un réseau de registres construit sous forme de pipeline. De cette manière $\wp$ syndromes ($\Psi_i$, $\Psi_{i+1}$, $\cdots$, $\Psi_{i+\wp-1}$) seront disponibles à chaque cycles, ce qui permet de corriger simultanément $\wp$ bits reçus.

Dans le cas du décodeur considéré ici, l'évaluation des bits reçus se fait par une fonction à logique majoritaire, d'où le nom de décodeur à logique majoritaire. Son principe de fonctionnement est basé sur le calcul des sommes de contrôle de parité orthogonales (*orthogonal parity-chek sums*).

1. Dans la limite fixée par la distance minimale du code (cf. paragraphe 2.2.1).
2. A condition que le code ne dépasse pas sa capacité de correction (cf. paragraphe 2.2.1).

## 6.2 Principe du décodage à logique majoritaire

### 6.2.1 Somme des contrôles de parité

Comme le montre l'équation (6.11), chaque bit $\psi_i$ d'un syndrome représente une erreur potentielle $e_i$ introduite par le canal de transmission. La somme de deux ou plusieurs bits du syndrome $\Psi_i$ est appelée somme de contrôle de parité (*parity-chek sum*). Dans le cas où la séquence reçue est un mot de code, tous les bits de syndrome et toutes les sommes de contrôle de parité sont nuls. Dans le cas contraire si le mot reçu n'appartient pas au code alors il existe des bits de syndrome et des sommes de contrôle de parité qui ne sont pas nuls. Ces sommes non nulles contiennent forcément le bit d'erreur $e_i$. On dit alors, qu'un bit d'erreur quelconque est contrôlé par une somme de contrôles de parité si ce dernier est inclus dans la somme.

#### Sommes des contrôles de parité orthogonales

Du point de vue algébrique la définition de deux vecteurs orthogonaux est la suivante :

**Définition** Soit un espace vectoriel à n-dimension sur CG(2). Deux éléments $A = (a_0, a_1, \cdots, a_{n-1})$ et $B = (b_0, b_1, \cdots, b_{n-1})$ de cet espace sont orthogonaux ($\perp$) si le produit scalaire ($\cdot$) des deux est nul.

$$(A \perp B) \Rightarrow (A \cdot B) = (a_0 \cdot b_0 + a_1 \cdot b_1 + \cdots + a_{n-1} \cdot b_{n-1}) = 0 \tag{6.15}$$

Dans le cas d'un ensemble de $\varrho$ sommes de contrôle de parité, on dit que les $\varrho$ sommes sont orthogonales au bit d'erreur $e_i$ si chaque somme contrôle le bit $e_i$ et si pas plus d'une somme contrôle un bit d'erreur différent de $e_i$.

### 6.2.2 Règle à logique majoritaire pour le décodage

Ayant défini un ensemble de $\varrho$ sommes de contrôle de parité orthogonales au bit d'erreur $e_i$, celui-ci est considéré erroné ($e_i$=1) si au moins ($\delta + 1$) des $\varrho$ sommes sont égales à 1. Dans le cas contraire $e_i$=0. Le paramètre $\delta$ est appelé « capacité de correction d'erreur par logique majoritaire » ou seuil de décodage, sa valeur est donnée par l'équation (6.16).

$$\delta = \frac{\varrho}{2} \tag{6.16}$$

## 6.3 Architecture série du décodeur de syndrome

La figure 6.1 représente l'architecture série d'un décodeur de syndrome.

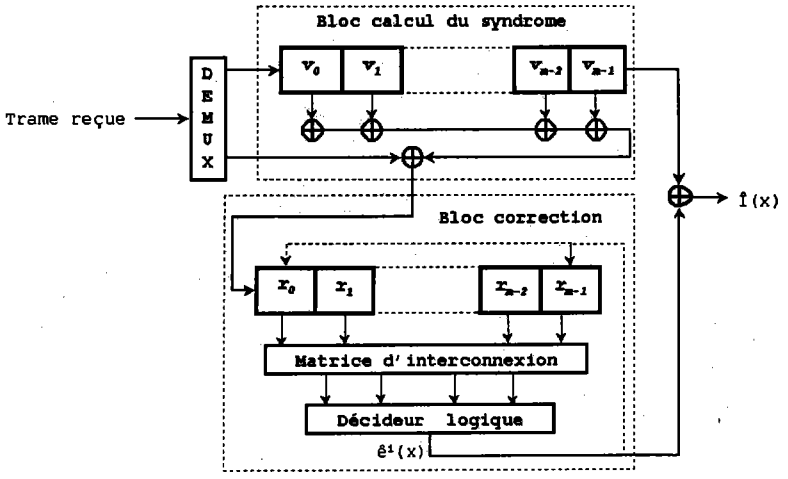



FIG. 6.1 – *Architecture série du décodeur de syndrome.*

Elle est divisée en deux blocs :

- **Bloc de calcul des syndrome** Ce bloc contient une réplique exacte du codeur qui calcule les bits de parités $\widehat{c}_i$ à partir des bits systématiques reçus $\widetilde{d}_i$. Les bits de syndrome sont ensuite calculés suivant la relation (6.9).
- **Bloc de correction** Ce bloc contient un registre de syndrome à partir duquel sont formées les sommes de parité orthogonales. La fonction de décision fournit une évaluation du bit testé d'après l'équation (6.13). Le bit est corrigé ensuite suivant la relation (6.14).

La figure 6.2 décrit succinctement les opérations réalisées dans un décodeur à logique majoritaire.

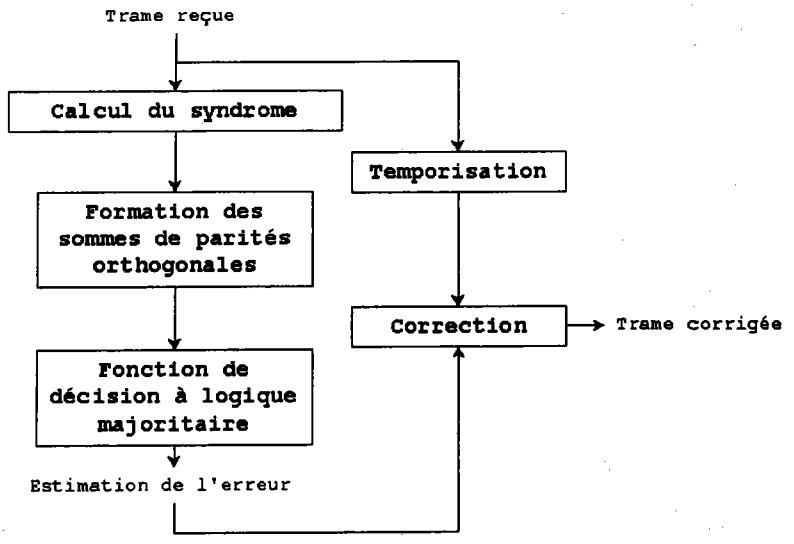



FIG. 6.2 – *Algorithme de fonctionnement du décodeur à logique majoritaire.*

## 6.4 Architecture parallèle proposée

Dans le cas d'un codeur systématique de rendement $1/n$ et de degré de parallélisme $\wp$, le décodeur parallèle doit être capable de traiter les $n \cdot \wp$ bits reçus à chaque cycle d'horloge. Les $\wp$ syndromes sont calculés à partir des $\wp$ bits systématiques et des $(n-1) \cdot \wp$ bits de parité. Cette tâche est réalisée par une réplique du codeur. La partie correction des erreurs doit évaluer les $\wp$ bits reçus à partir des syndromes correspondants. Le point critique réside dans le fait que $(m+1)$ bits de syndrome $(\psi_{i+m}, \psi_{i+m-1}, \cdots, \psi_i)$ sont nécessaires pour l'évaluation d'un bit $d_i$ tandis qu'à chaque cycle d'horloge, $\wp$ bits de syndrome sont disponibles. Trois cas se présentent $\wp < m$, $\wp = m$ et $\wp > m$. Une solution différente est proposée pour chaque cas. L'étude architecturale du codeur parallèle portera d'abord sur le cas plus simple d'un rendement $1/2$ avant d'être généralisée au cas $1/n$.

La figure 6.3 représente le schéma fonctionnel du décodeur de syndrome parallèle. Il est composé de 4 blocs :

- **Bloc Codeur Parallèle (CP)** Ce bloc traite les $\wp$ bits systématiques reçus.
- **Bloc Réseau de Correction (CR)** Ce bloc corrige les $\wp$ bits testés.
- **Réseau de Registres à Décalage (RRD)** Ils sont au nombre de deux et permettent la synchronisation des flots de données circulant dans le décodeur.

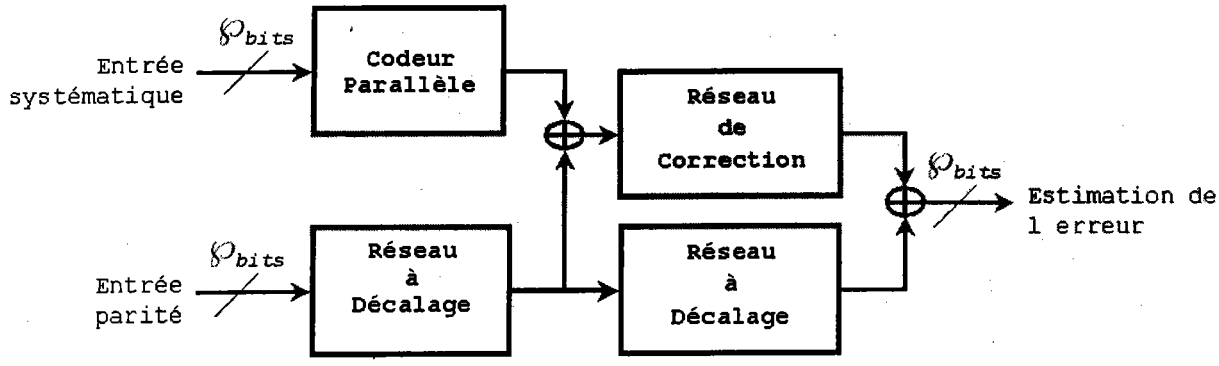



FIG. 6.3 – *Schéma fonctionnel du décodeur parallèle.*

### 6.4.1 Codeur parallèle

Le codeur parallèle calcule $\wp$ bits de parité $(\widehat{c}_i, \cdots, \widehat{c}_{i+\wp-1})$ à partir des $\wp$ bits systématiques reçus $(\widetilde{d}_i, \cdots, \widetilde{d}_{i+\wp-1})$. Les $\wp$ syndromes sont ensuite calculés d'après l'équation (6.9). L'architecture du codeur parallèle est détaillée dans le chapitre 5.

### 6.4.2 Réseau de correction

La figure 6.4 représente le bloc de correction série pour un rendement $1/2$ et une taille de mémoire $m = 3$. La boucle de rétroaction n'est pas prise en compte.

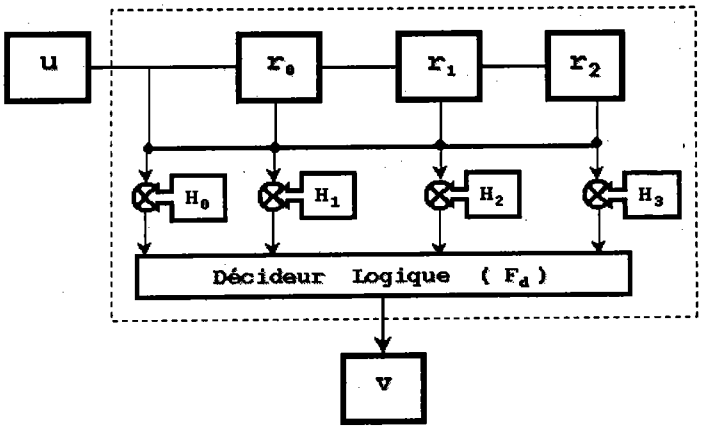



FIG. 6.4 – *Registre de syndrome série pour* $m = 3$.

Le fonctionnement du registre de syndrome est décrit par l'équation 6.17. Pour des raisons de synchronisation, deux registres $u$ et $v$ sont insérés respectivement à l'entrée du registre de syndrome et à la sortie de la fonction de décision.

$$\begin{aligned} (r_0)_{t+1} &= (u)_t \\ (r_1)_{t+1} &= (r_0)_t \\ (r_2)_{t+1} &= (r_1)_t \end{aligned} \tag{6.17}$$

La forme matricielle de l'équation (6.17) est donnée par l'équation (6.18)

$$(R)_{t+1} = (R)_t \cdot T_s + ((u)_t, 0, 0) \tag{6.18}$$

où $R = (r_0,r_1,r_2)$ et $T_s = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$.

La correction du bit évalué se fait selon l'équation (6.19).

$$\begin{aligned} (v)_{t+1} &= F_d[H_0 \cdot (R^*)_t, H_1 \cdot (R^*)_t, H_2 \cdot (R^*)_t, H_3 \cdot (R^*)_t] \\ (v)_{t+1} &= F_d[H \cdot (R^*)_t] \end{aligned} \tag{6.19}$$

avec $(R^*)_t = (u, r_0, r_1, r_2)_t$, $H = \begin{pmatrix} h_{0,0} & h_{0,1} & h_{0,2} & h_{0,3} \\ h_{1,0} & h_{1,1} & h_{1,2} & h_{1,3} \\ h_{2,0} & h_{2,1} & h_{2,2} & h_{2,3} \\ h_{3,0} & h_{3,1} & h_{3,2} & h_{3,3} \end{pmatrix}$ et $H_i = \begin{pmatrix} h_{0,i} \\ h_{1,i} \\ h_{2,i} \\ h_{3,i} \end{pmatrix}$.

Dans le cas général d'un codeur de mémoire $m$, les équations (6.18) et (6.19) restent valables, avec

$$R = (r_0, r_1, r_2, \cdots, r_{m-1}), T_s = \begin{pmatrix} 0 & 1 & 0 & . & 0 \\ 0 & 0 & 1 & . & 0 \\ . & . & . & . & . \\ 0 & 0 & 0 & . & 1 \\ 0 & 0 & 0 & . & 0 \end{pmatrix} \text{ et } H = \begin{pmatrix} h_{0,0} & h_{0,1} & h_{0,2} & . & h_{0,m} \\ h_{1,0} & h_{1,1} & h_{1,2} & . & h_{1,m} \\ h_{2,0} & h_{2,1} & h_{2,2} & . & h_{2,m} \\ . & . & . & . & . \\ h_{m,0} & h_{m,1} & h_{m,2} & . & h_{m,m} \end{pmatrix}.$$

Si par exemple, à l'instant $t$, le registre $R$ contient le syndrome $\Psi_{i-1} = (\psi_{i+m-2}, \psi_{i+m-1}, \cdots, \psi_{i-1})$ alors aux instants respectifs $t+1$ et $t+2$, les syndromes seront calculés d'après (6.20) et (6.21).

$$\text{à l'instant t+1} \rightarrow \begin{cases} (R)_{t+1} &= (R)_t \cdot T_s + (u)_t \\ &= \Psi_i = (\psi_{i+m-1}, \psi_{i+m-2}, \cdots, \psi_i) \\ (v)_{t+2} &= F_d[H \cdot (R^*)_{t+1}] \\ &= \widehat{e}_i = F_d[H \cdot (\psi_{i+m}, \psi_{i+m-1}, \psi_{i+1}, \cdots, \psi_i)] \end{cases} \tag{6.20}$$

$$\text{à l'instant t+2} \rightarrow \begin{cases} (R)_{t+2} &= (R)_{t+1} \cdot T_s + (u)_{t+1}, \\ &= \Psi_{i+1} = (\psi_{i+m}, \psi_{i+m-1}, \cdots, \psi_{i+1}). \\ (v)_{t+3} &= F_d[H \cdot (R^*)_{t+2}], \\ &= \widehat{e}_{i+1} = F_d[H \cdot (\psi_{i+m+1}, \psi_{i+m}, \psi_{i+m-1}, \cdots, \psi_{i+1})]. \end{cases} \tag{6.21}$$

Le tableau 6.1 donne un exemple de formation et propagation des syndromes et montre pour chaque syndrome le bit testé. On remarque l'existence d'un temps de latence de $m$ cycles d'horloge qui correspondent au temps de remplissage du registre de syndrome. Dans le cas général, pour une taille de mémoire quelconque $m$, au $k^{ime}$ cycle d'horloge le registre de syndrome contient le syndrome $\Psi_{k-m-1} = (\psi_{k-1}, \cdots, \psi_{k-m-2}, \psi_{k-m-1})$ et corrige le bit reçu $\widetilde{d}_{k-m-1}$.

| cycle | $R$ | $(r_0, r_1, r_2)$ | $\widetilde{d}_i$ |
|---|---|---|---|
| t+1 | x | $\psi_0$,0,0 | - |
| t+2 | x | $\psi_1, \psi_0, 0$ | - |
| t+3 | $\Psi_0$ | $\psi_2, \psi_1, \psi_0$ | $\widetilde{d}_0$ |
| t+4 | $\Psi_1$ | $\psi_3, \psi_2, \psi_1$ | $\widetilde{d}_1$ |
| t+5 | $\Psi_2$ | $\psi_4, \psi_3, \psi_2$ | $\widetilde{d}_2$ |
| ... | ... | ... | ... |
| t+k | $\Psi_{k-3}$ | $\psi_{k-1}, \psi_{k-2}, \psi_{k-3}$ | $\widetilde{d}_{k-3}$ |

TAB. 6.1 – *Formation et propagation des syndromes.*

Le réseau de registres de la figure 6.5, appelé dans la suite « réseau de propagation », permet d'assurer la propagation des syndromes comme indiqué dans le tableau 6.1. Si par exemple le codeur parallèle génère trois syndromes ($\wp = 3$) à chaque cycle d'horloge, le réseau de propagation devra fournir les trois syndromes en même temps. La figure 6.5 montre la propagation des bits pendant les trois cycles suivant l'instant $t$ où les bits de syndrome ($\psi_0, \psi_1, \psi_2$) sont présentés à l'entrée du réseau.

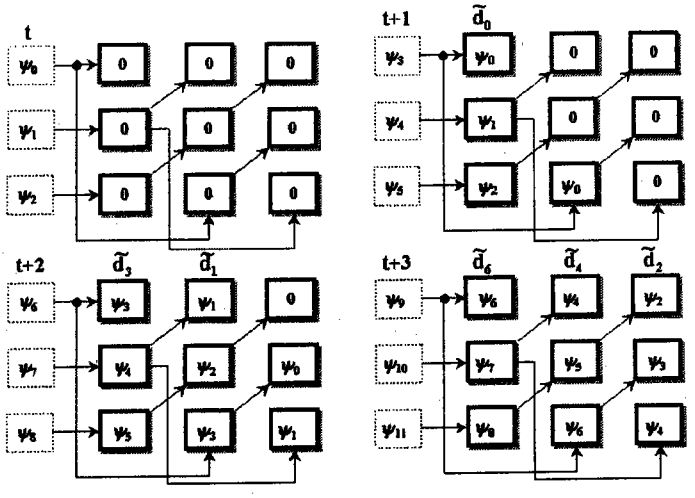



FIG. 6.5 – *Exemple du réseau de propagation pour* $\wp = 3$.

Après un temps de latence, à chaque cycles trois syndromes ($\Psi_{im}, \Psi_{im-2}, \Psi_{im-4}$) sont disponibles dans le réseau.

## 6.5 Architecture parallèle du bloc de correction

En généralisant le réseau de la figure 6.5 à un degré de parallélisme $\wp$, on constate que sa taille doit être de ($\wp \times \wp$) cellules pour qu'il puisse prendre en charge les $\wp$ bits de syndrome générés. Concrètement, l'évaluation d'un bit se fait à partir des $(m + 1)$ syndromes calculés au préalable, ce qui correspond à la longueur de contrainte du code. Par conséquent, trois cas de figure sont possibles :

- $\wp = m$ ;
- $\wp < m$ ;
- $\wp > m$.

où $\wp$ est le degré de parallélisme et $m$ la taille de la mémoire du codeur.

### 6.5.1 Cas $\wp = m$

Dans ce cas idéal, les $m$ bits de syndrome nécessaires à l'évaluation sont disponibles à chaque cycle. De plus, la taille du réseau de ($m \times m$) cellules permet bien d'évaluer les $\wp$ bits reçus. La figure 6.6 représente le réseau de correction dans le cas $m = \wp = 4$.

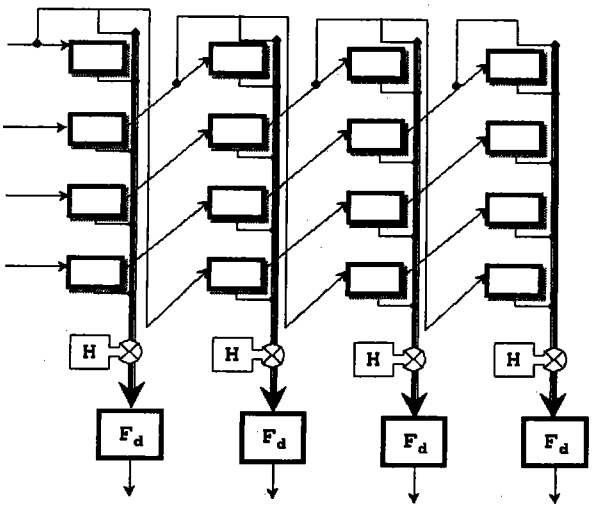



FIG. 6.6 – *Réseau de correction pour* $m = \wp = 4$.

### 6.5.2 Cas $\wp < m$

Dans ce cas, les $m$ bits de syndrome nécessaires à l'évaluation ne sont pas disponibles à chaque cycle d'horloge. Un réseau additionnel de registres à décalage est utilisé afin de former les syndromes requis. Son rôle consiste à retarder les $\wp$ premier bits, d'un nombre de cycles qui dépend de $m$.

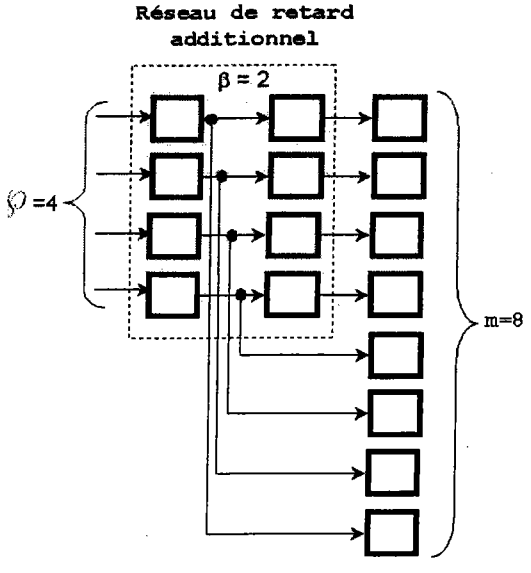



FIG. 6.7 – *Réseau de retard pour* $(m, \wp, \beta) = (8, 4, 2)$.

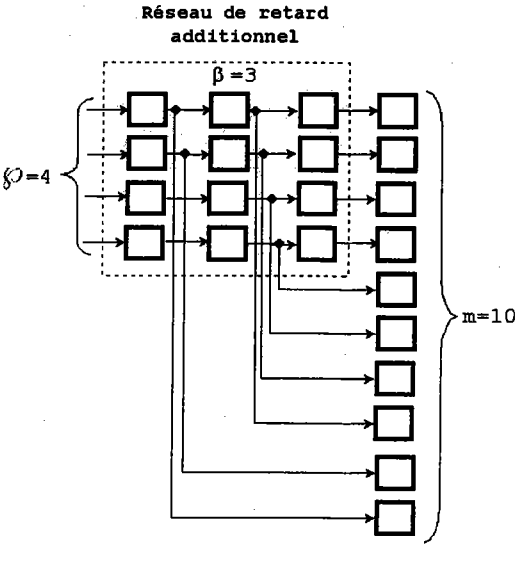



FIG. 6.8 – *Réseau de retard pour* $(m, \wp, \beta) = (10, 4, 3)$.

Le réseau de retard est calculé d'après les relations (6.22) et (6.23).

$$\text{si} \quad m \ \texttt{modulo}\ [\wp] = 0 \Rightarrow \quad \beta = \frac{m}{\wp} \tag{6.22}$$

$$\text{si} \quad m \ \texttt{modulo}\ [\wp] \neq 0 \Rightarrow \quad \beta = \frac{m}{\wp} + 1 \tag{6.23}$$

Le paramètre $\beta$ représente le nombre de cycles de retard appliqués aux premiers $\wp$ bits du syndrome. Dans la pratique, il représente le nombre de registres à décalage de taille $\wp$ insérés à l'entrée du réseau de correction. Les figures 6.7 et 6.8 représentent le réseau de retard dans les cas $(m, \wp) = (8, 4)$ et $(m, \wp) = (10, 4)$, respectivement.

### 6.5.3 Cas $\wp > m$

Dans ce cas, il y a un excédent de bits de syndrome car seuls $m$ bits sont nécessaires. Deux solutions sont envisageables, la première permet de retarder les bits en surplus via un réseau de décalage. Cette solution simple, présente l'inconvenient d'exiger une taille de réseau de retard qui augmente considérablement avec le degré de parallélisme. La deuxième solution consiste à modifier le réseau de correction de sorte à optimiser la propagation des bits de syndromes à travers lui, ce qui conduit à un réseau de retard de moindre surface.

**- Structure 1**

Un premier type de structure pour le bloc de correction (structure 1), correspond au cas $(m, \wp) = (4, 8)$, est présenté sur la figure 6.10. Le réseau de retard $T_{RR1}$ y est représenté en gris. Sa taille augmente suivant la relation (6.24).

$$T_{RR1} = \frac{\wp \cdot (\wp + 1)}{2} \tag{6.24}$$

**- Structure 2**

Le réseau de retard est calculé selon les équations (6.25) et (6.26).

$$\text{si} \quad \wp \ \text{mod}\ [m] = 0 \Rightarrow \quad \beta = 1 \tag{6.25}$$

$$\text{si} \quad \wp \ \text{mod}\ [m] \neq 0 \Rightarrow \quad \beta = 2 \tag{6.26}$$

La taille du réseau de retard additionnel est $\beta \times \wp$ cellules ($\beta$ registres à décalage de $\wp$ cellules). Le réseau de correction est modifié d'après les équations (6.27) et (6.28).

$$\text{si} \quad \wp \bmod [m] = 0 \Rightarrow \quad \alpha = \frac{\wp}{m} \tag{6.27}$$

$$\text{si} \quad \wp \bmod [m] \neq 0 \Rightarrow \quad \alpha = \frac{\wp}{m} + 1 \tag{6.28}$$

La taille du réseau de correction modifié est de $\alpha \times m \times m$ cellules. La taille du réseau de retard additionnel $T_{RR2}$ est calculée selon la relation (6.29).

$$T_{RR2} \leq \alpha \cdot m^2 - \wp \cdot m + \beta \cdot \wp \tag{6.29}$$

La figure 6.9 représente la comparaison, pour différentes tailles de mémoire, des deux structures en fonction du degré de parallélisme.

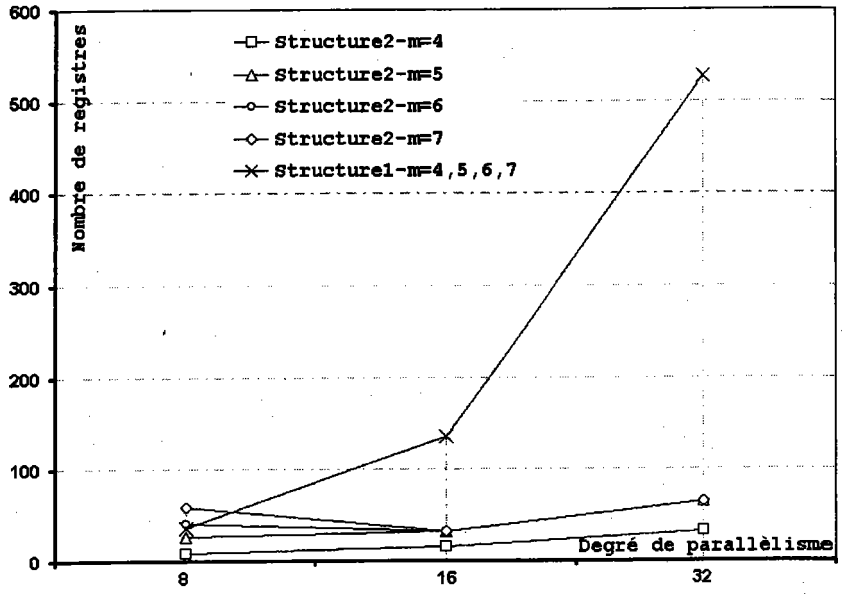



FIG. 6.9 – *Tailles $T_{RR1}$ et $T_{RR2}$ des 2 structures.*

Les figures 6.10 et 6.11 représentent, pour $(m, \wp) = (4, 8)$, les blocs de correction dans le cas de la structure 1 et de la structure 2, respectivement.

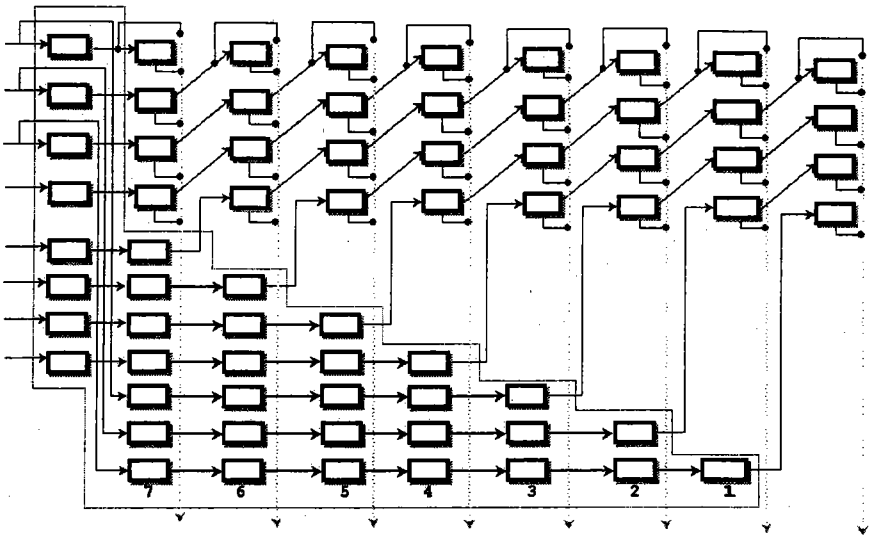



FIG. 6.10 – *Structure 1 du réseau de correction pour* $(m, \wp) = (4, 8)$.

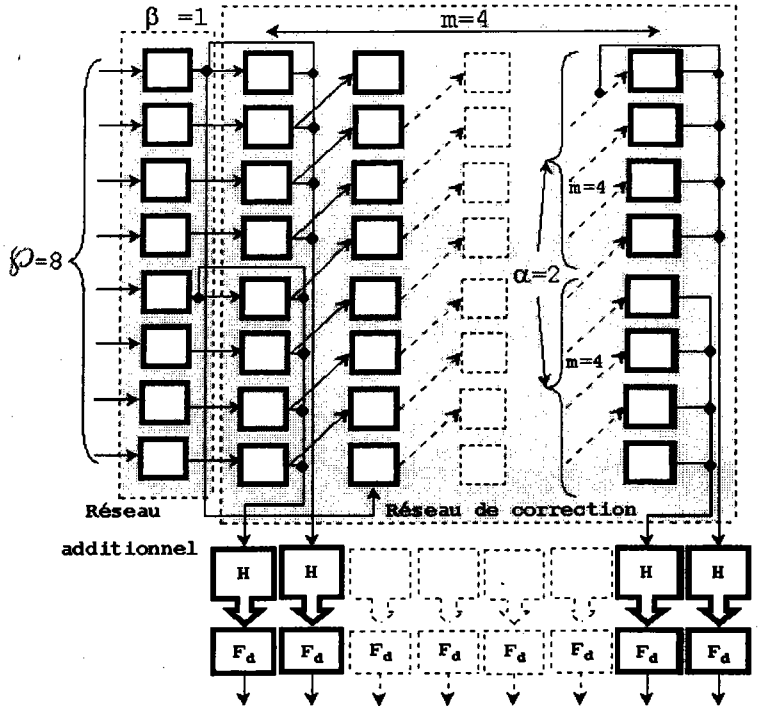



FIG. 6.11 – *Structure 2 du réseau de correction pour* $(m, \wp) = (4, 8)$.

## 6.6 Réseau de correction dans le cas de rendement $1/n$

Dans le cas d'un décodeur de rendement $1/n$, le nombre de réseaux de correction est proportionnel au nombre d'entrées de parité. Dans le cas par exemple de la figure 6.12, le nombre de réseaux de correction est de 3. Les sommes de contrôle de parité sont formées à partir de la matrice d'interconnexion.

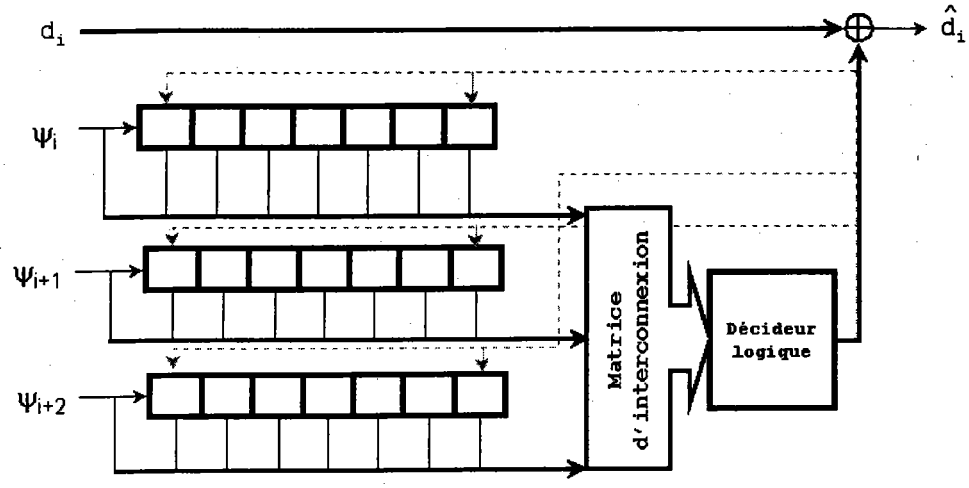



FIG. 6.12 – *Bloc de correction pour* $R$=1/4.

Chaque registre de syndrome du cas série est remplacé par le réseau correspondant en suivant les règles définies dans la section 6.5 et en fonction des valeurs de $m$ et de $\wp$. Pour la formation des sommes de parité orthogonales, chaque registre de syndrome de $m$ bits de largeur est connecté à son voisin selon les relations suivantes :

- cas $\wp \leq m$,

$$\begin{cases} k = 1 \rightarrow (n-1) \\ i = 1 \rightarrow m \\ R_g(k,i) \text{ est connecté avec } R_g(k+1,i) \end{cases} \tag{6.30}$$

  où $(n-1)$ représente le nombre de sorties non systématiques et $m$ les tailles du codeur et du registre de syndrome $R_g$ ;

- cas $\wp > m$,

$$\begin{cases} k = 1 \rightarrow (n-1) \\ i = 1 \rightarrow \alpha \\ j = 1 \rightarrow m \\ R_g(k,i,j) \text{ est connecté avec } R_g(k+1,i,j) \end{cases} \tag{6.31}$$

  $\alpha$ est calculé d'après les équations (6.27) et (6.28).

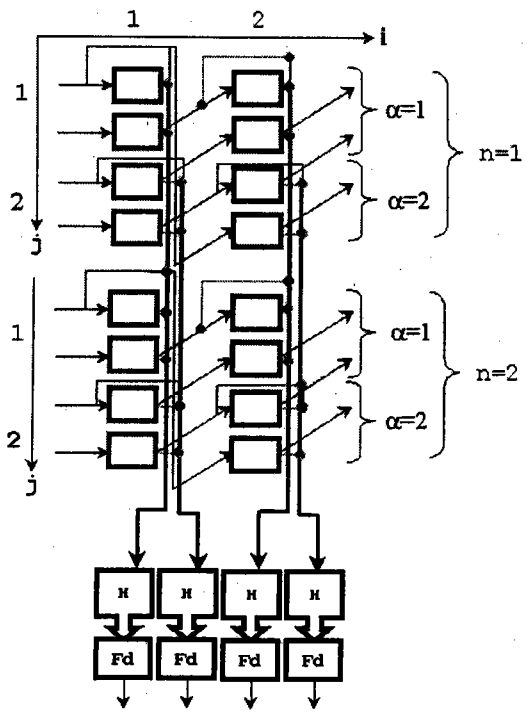



FIG. 6.13 – *Connexion de deux réseaux de correction.*

## 6.7 Matrice d'interconnexion

Les sommes de parité orthogonales sont formées en additionnant les bits de syndrome. Le choix des bits additionnés se fait par le biais de la matrice d'interconnexion. L'addition entre deux bits est représentée par un « 1 » et la non-addition par un « 0 ». La figure 6.14 représente le réseau d'interconnexion pour la matrice $M$.

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \Rightarrow \begin{cases} c_0 = 0 \\ c_1 = r_6 \\ c_2 = r_2 + r_5 \\ c_3 = r_4 \\ c_4 = r_3 + r_6 \\ c_5 = r_4 + r_7 \\ c_6 = r_1 \\ c_7 = r_0 \end{cases}$$

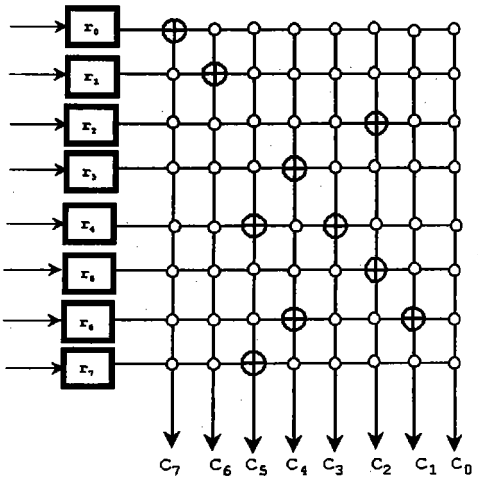



FIG. 6.14 – *Construction des sommes de parité orthogonales pour la matrice $M$.*

## 6.8 Résultats expérimentaux

L'architecture du décodeur de syndrome a été implantée sur des FPGA de type Flex10KE d'Altera. La synthèse a été effectuée avec Synplify et pour les phases de placement, routage et simulation avec MAX+plus II. Comme pour le codeur, les tests du réseau de correction ont été réalisés pour différents niveaux de parallélisation, valeurs de rendement et tailles de mémoire.

Les Figures 6.15, 6.16 et 6.17 donnent les compromis débit/surface des réseaux de correction pour des rendements $1/2$, $1/3$ et $1/4$ et différentes tailles de mémoire.

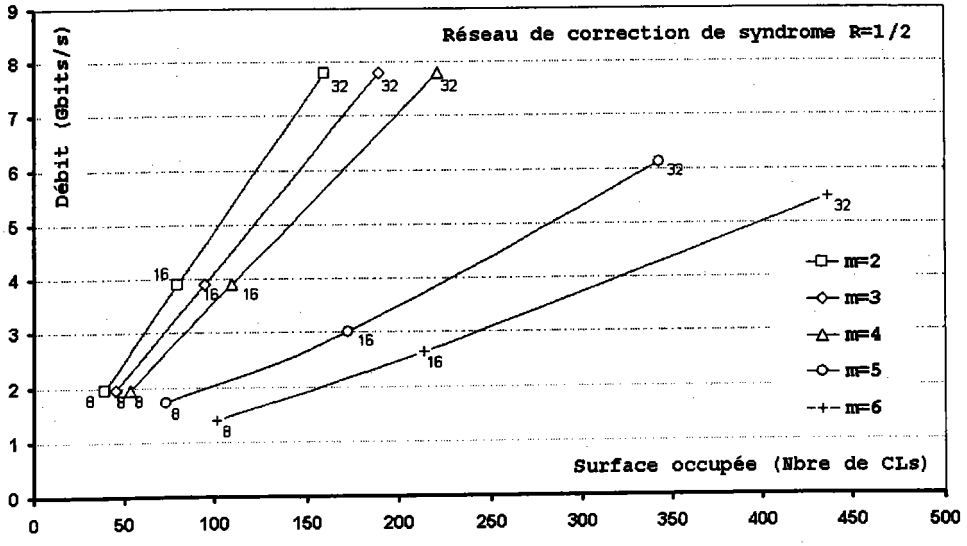



FIG. 6.15 – *Compromis débit/surface en fonction de $\wp$ pour $R = 1/2$.*

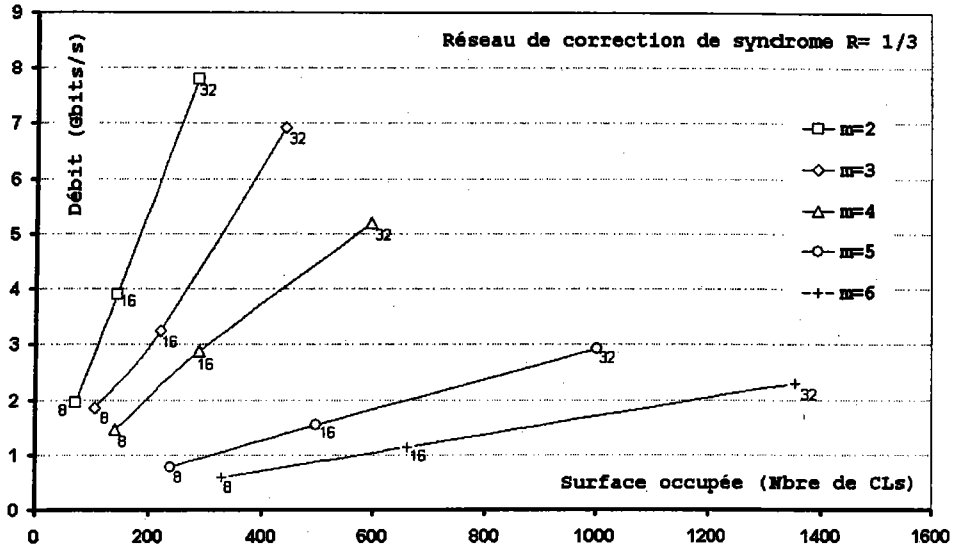



FIG. 6.16 – *Compromis vitesse/surface en fonction de $\wp$ pour $R = 1/3$.*

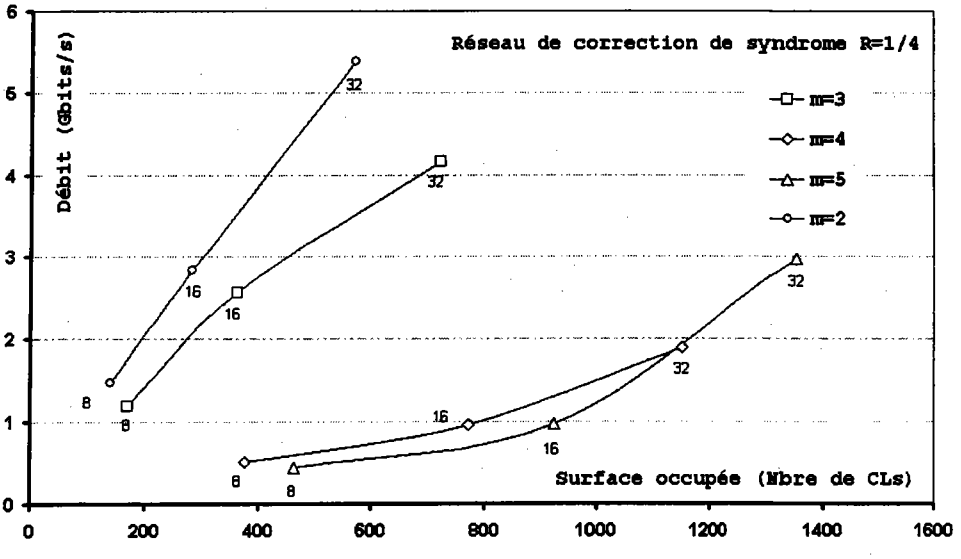



FIG. 6.17 – *Compromis débit/surface en fonction de $\wp$ pour $R = 1/4$.*

Les figures 6.18 et 6.19 représentent les résultats de simulation du réseau de correction parallèle et du bloc de correction série, respectivement. Les ciruits ont été simulés pour un degré de parallélisation $\wp = 4$, une taille de mémoire $m = 4$ et un seuil de décodage $\delta = 2$. On suppose que chaque bit de syndrome est calculé en amont par le bloc codeur. Dans les figures, `Sin` et `ErrS` sont respectivement, les entrées des bits de syndrome et les sortie des erreurs estimées dans le cas parallèle. `S_serie` et `Err_serie` correspondent au cas série.

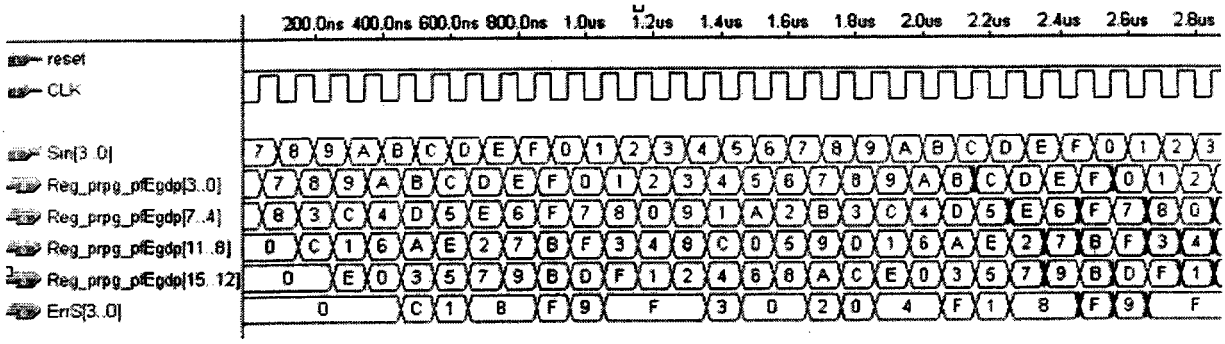



FIG. 6.18 – *Simulation du réseau de correction parallèle.*

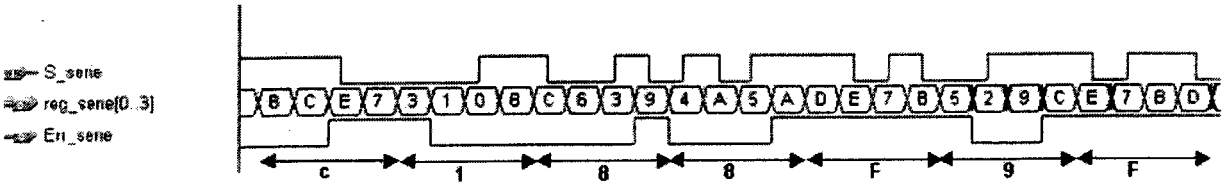



FIG. 6.19 – *Simulation du bloc de correction série.*

### 6.8.1 Analyse du réseau de correction

Le chemin critique dans le réseau de correction est généralement dû à la matrice d'interconnexion. Celle-ci définie le nombre d'additions utilisées pour former les sommes de parité orthogonales. Plus précisément, le nombre maximum de ou-exclusifs (XOR) utilisés pour réaliser une somme détermine le chemin critique. Les figures 6.20 et 6.21 donnent, en fonction du degré de parallélisation, les fréquences maximales de fonctionnement et les surfaces occupées dans le FPGA pour deux matrices d'interconnexions différentes $M1$ et $M2$.

$$M1 = \begin{pmatrix} 10101010 \\ 01010101 \\ 10101010 \\ 01010101 \\ 10101010 \\ 01010101 \\ 10101010 \\ 01010101 \end{pmatrix} \text{ et } M2 = \begin{pmatrix} 10000000 \\ 01000000 \\ 00100000 \\ 00010000 \\ 00001000 \\ 00000100 \\ 00000010 \\ 00000001 \end{pmatrix}$$

En fonction du degré de parallélisation pour une matrice d'interconnexion donnée, les surfaces consommées augmentent alors que les fréquences maximales de fonctionnement restent relativement stables. En effet, l'augmentation du degré de parallélisme n'affecte pratiquement pas le chemin critique du circuit. A l'inverse pour des paramètres donnés ($\wp$,m,r,$F_d$) fixes, les fréquences maximales de fonctionnement dépendent des matrices d'interconnexion correspondantes.

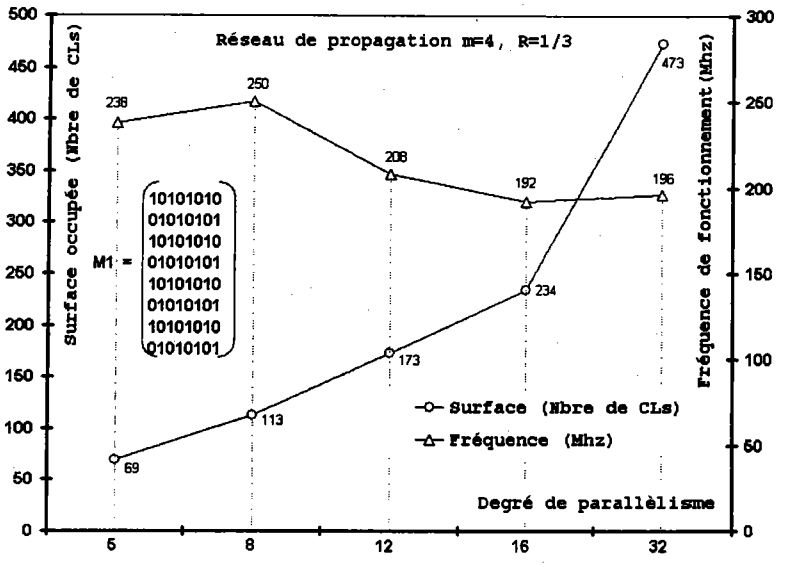



FIG. 6.20 – *(surface, fréquence)/degré de parallélisme pour la matrice M1.*

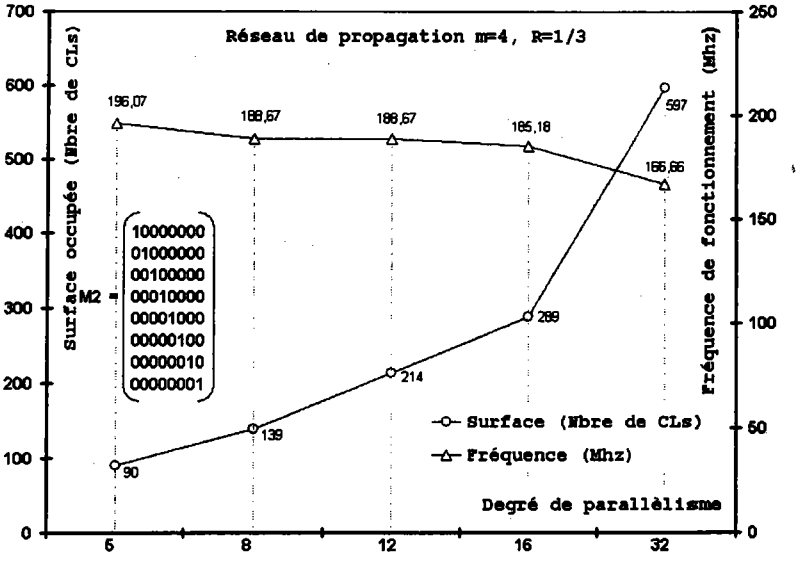



FIG. 6.21 – *(surface, fréquence)/degré de parallélisme pour la matrice M2.*

### 6.8.2 Implantation du décodeur

Le décodeur complet a été implanté pour 3 configurations différentes (voir tableau 6.2) pour un rendement de $1/2$. Les codeurs utilisés sont de type $CP2_{mto}$ (cf. paragraphe 5.7.1). Les courbes de compromis débit/surface sont données dans la figure 6.22.

| R=1/2 | $(k,n,m)(G(x),H(x))_8$ | Matrice |
|---|---|---|
| Décodeur 1 | $(1,2,2)(7,5)_8$ | $M1$ |
| Décodeur 2 | $(1,2,3)(13,17)_8$ | $M1$ |
| Décodeur 3 | $(1,2,4)(23,33)_8$ | $M1$ |

TAB. 6.2 – *Polynômes générateurs de codes.*

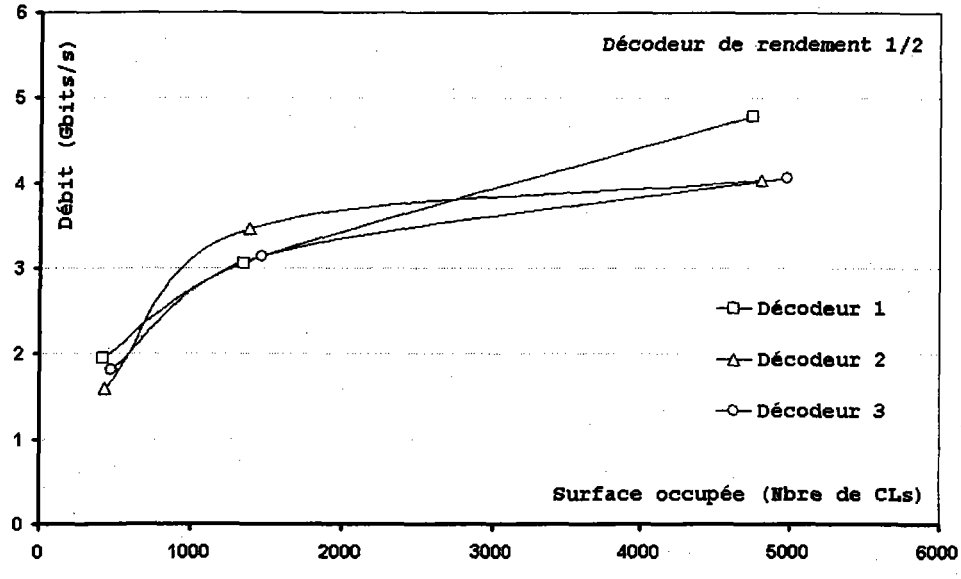



FIG. 6.22 – *Compromis débit/surface pour un décodeur de rendement* $1/2$.

## 6.9 Conclusion

Ce chapitre présente une architecture rapide d'un décodeur de syndrome pour les codes convolutifs. L'architecture proposée repose sur la parallélisation du codeur et du registre de syndrome constituant du décodeur. les modèles implantés sont indépendants d'une technologie cible et configurables avant la synthèse.

Le chemin critique dépend du bloc codeur d'une part, et du réseau de correction d'autre part. L'analyse du bloc codeur est donnée dans les paragraphes 5.7.2 et 5.7.3. Concernant le bloc de correction, le chemin critique dépend, pour une fonction de décision donnée, des valeurs des matrices d'interconnexion. Il est pratiquement indépendant du degré de parallélisme.

Les débits obtenus[3] pour des implantations sur FPGA de type Flex10KE sont compris, pour la matrice d'interconnexion $M1$, entre 1.59 Gbits/s et 4.78 Gbits/s pour des degrés de parallélisme $\wp = 8$ et $\wp = 32$, respectivement (cf. annexe F).

3. En considérant les polynômes générateurs donnés dans le tableau 6.2.

# CONCLUSION GÉNÉRALE

# Conclusion générale

Le travail de cette thèse fait partie d'un projet global mené dans le laboratoire LICM concernant la conception architecturale d'une chaîne de transmission de faible coût et à haut débit. Il se situe au niveau d'un maillon de la chaîne de transmission, à savoir, le codage canal. L'objectif du travail est de développer une architecture de codeur canal qui satisfasse aux quatres conditions suivantes :

- protection optimale des données ;
- traitement rapide (haut débit) ;
- architecture configurable ;
- utilisation de technologies de faible coût.

Pour cela, une étude approfondie à travers le chapitre 1 est effectuée sur les stratégies de codage canal dans le domaine des télécommunications. Il en ressort que les stratégies FEC (codes correcteurs et détecteurs d'erreurs) constituent l'approche offrant le meilleur compromis débit/protection. En poursuivant l'étude dans la famille FEC, il apparaît que les codes convolutifs sont parmi les codes les plus utilisés en télécommunication. De plus, la possibilité de les utiliser, soit comme entité indépendante, soit dans une structure concaténée avec un autre code, soit comme élément constituant d'un turbo-code les rend très attractifs. D'autant plus que, dans les stratégies FEC, les turbo-codes offrent actuellement les meilleures performances de codage. Le choix s'est donc naturellement porté sur les codes convolutifs et leur application aux turbo-codes.

Le chapitre 2 est consacré à l'étude des aspects fonctionnels et architecturaux des codes convolutifs. Dans ce travail, sont abordés différentes architectures de codeurs convolutifs et les critères permettant leur selection dans les turbo-codes. Les paramètres déterminant les performances d'un turbo-codeur et les conditions de bon fonctionnement sont analysés.

Le chapitre 3 traite des différents algorithmes de décodage itératif des turbo-codes. La complexité architecturale des algorithmes les plus utilisés dans un turbo-décodeur (SOVA, Log MAP et Max-Log MAP) y est comparée. Il est à noter que toutes ces approches sont basées sur des versions modifiées de l'algorithme de Viterbi dont l'étude montre que le débit est principalement limité par le bloc ACS (*Add-Compare-Select*). Différentes structures peuvent être utilisées pour augmenter le débit global d'un décodeur élémentaire du turbo-décodeur. Les plus connues sont les structures Radix-$k$. Une implantation sur FPGA de type Flex10KE a permis d'observer que les structures Radix-4 ont le

meilleur compromis débit/surface.

L'étude expérimentale des architectures systoliques dédiées à la convolution (implantations sur FPGA de type Altera/Flex10KE et simulations) fait l'objet du chapitre 4. Elle met en évidence la simplicité de leur mise en œuvre, due à leurs structures régulières. La possibilité de rendre ces architectures résistantes aux pannes, de les reconfigurer en ligne et d'effectuer des traitements en flot continu, les rendent particulièrement intéressantes. Cependant, le traitement sériel des données limite le débit à une valeur égale au mieux, à la fréquence maximale de fonctionnement du composant cible. D'où la nécessité de développer une architecture parallèle permettant un traitement plus rapide.

Le chapitre 5 est dédié à l'étude d'une architecture rapide de codeurs convolutifs (récursifs et non récursifs) et à son application à des turbo-codes. La méthodologie de conception adoptée consiste en la formulation mathématiques des algorithmes de traitement suivie de la recherche des solutions offrant la meilleure adéquation algorithme/architecture. Les architectures retenues sont ensuite décrites au niveau RTL, synthétisées sur FPGA, simulées puis évaluées en termes de compromis débit/surface ou de variations de (fréquence, surface) en fonction du degré de parallélisme.

L'application de la méthodologie de conception sur les codes convolutifs a permis le développement et l'implantation, sur FPGA de type Altera/Flex10KE, de quatre modèles RTL selon différentes stratégies d'implantation. Les modèles décrits en VHDL sont génériques, indépendants de toute technologie cible et configurables avant la synthèse. Un programme écrit en C a été développé pour permettre la génération automatique des paquetages de configuration à partir de certains paramètres ($\wp$, $m$, $R$, $G(x)$, $H_i(x)$, etc.). Chaque architecture a été testée pour différents degrés de parallélisme, choix de polynômes générateurs et rendements. Pour les codeurs convolutifs, les débits obtenus sont compris entre 1 Gbits/s et 7.45 Gbits/s pour des degrés de parallélisme 8 et 32, respectivement. Un turbo-code parallèle à deux niveaux, de rendement $1/4$ basé sur des codes convolutifs récursifs et systématiques, a permis d'atteindre des débits compris entre 1 et 2 Gbits/s et entre 3 et 5 Gbits/s pour les degrés de parallélisme 8 et 32, respectivement.

Dans le chapitre 6, l'architecture rapide du codeur convolutif est exploitée pour la conception d'un décodeur de syndrome parallèle. Ce type de décodeur est constitué principalement d'une réplique du codeur (déjà traitée) et un registre de syndrome. Par conséquent, l'étude porte uniquement sur la parallélisation de ce registre de syndrome. L'implantation du décodeur sur un Altera/Flex10KE a permis d'atteindre des débits allant jusqu'a 4.78 Gbits/s pour un rendement de 1/2.

La suite logique, à court terme, du travail présenté dans ce mémoire de thèse est de développer une architecture rapide de turbo-décodeur. Les algorithmes et architectures de décodage itératif des turbo-codes du chapitre 3 peuvent être exploités dans le développement d'un décodeur rapide. La stratégie qui consiste à augmenter le débit de chaque code constituant d'un turbo-code, reste valide dans le cas du turbo-décodeur. En effet, comme vu dans le chapitre 3, le circuit de décodage d'un turbo-code convolutif est constitué de décodeurs élémentaires fonctionnant de manière itérative.

Une autre perspective de ce travail, à plus long terme, est d'implanter le turbo-(codeur/décodeur)

sous forme d'un module ADM (*Auxiliary Dedicated Module*) dans un processeur modulaire et hautement parallèle développé au LICM [1]. Les ADM de ce processeur, basé sur un cœur de DSP, sont dédiés à l'exécution de fonctions critiques. Dans ce cas présent, la fonction critique est le codage canal.

# Bibliographie

[1] S. Philip, « Etude et développement d'un nouvelle architecture de processeur DSP dédiée aux applications modem en télécommunication sur câble T.V », manuscrit de thèse, LICM, Univérsité de Metz, (Metz, France), Déc. 1999.

[2] C. Berrou, A. Glavieux, and P. Thitimajshima, « Near Shannon limit error correcting coding and decoding : Turbo-codes », *IEEE Int. Conf. on Commun.*, (Geneva, Switzerland), pp.1064-1070, May 1993.

[3] J. J. Boutros, « Les turbo codes parallèles et séries, décodage SISO et performance ML », rapport, Oct. 1998.

[4] C. Berrou, A. Glavieux, R. Pyndiah, « Turbo-Codes: Principes et Applications », *GRETSI'95*, (Juan-les-pins, France), Sept. 1995.

[5] Matthew C. Valenti, « Iterative detection and decoding for wireless communication », manuscrit de thèse PhD, Virginia Polytechnic Institute and State University, (Blacksburg, Virginia), July 1999.
Disponible sur http://scholar.lib.vt.edu/theses/available/etd-071399-133447, consulté en 2001.

[6] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, « Serial concatenation of interleaved codes: Performance analysis, design, and iterative decoding », *IEEE Trans. on Inf. Theory*, vol. 44, no. 3, pp. 909-926, May 1998.

[7] D. Divsalar, S. Dolinar, F. Pollara, and R.J. McEliece. « Transfer function bounds on the performance of turbo codes ». *JPL TDA Progress report 42-122*, Aug. 1995.

[8] A. Poli et LI. Huguet, *Codes Correcteurs, Théorie et Applications*, Ed. Masson, 1989.

[9] W. Weaver et C. E. Shannon, *Théorie Mathématique de la Communication*, Ed. Les Classiques des Sciences Humaines, 1975.

[10] A. Spataru, *Fondements de la Théorie de la Transmission de L'Information*, Ed. Presses Polytechniques Romandes, 1987.

[11] G. Battail, *Sciences et Avenir*, hors-série, pp.28-29, Déc. 1999.

[12] Shu Lin, Daniel J. Costello, Jr. *Error Control Coding : Fundamentals and Applications*, Prentice-Hall Serie in Computer Applications in Electrical Enginnering, 1983.

[13] J. Oswald, *Théorie de l'Information ou Analyse Diacritique des systèmes*, Ed. Masson, 1986.

[14] A. Dandache, « Contribution à la conception de circuits et de systèmes pour les applications télécom », Travaux Scientifiques en Vue d'une Habilitation à Diriger des Recherches, Univ. Metz, Jan. 2000.

[15] Specification of the bluetooth system, vol. 1.0B, Dec. 1999.

[16] H. Yamamoto, K. Itoh, « Viterbi decoding algorithm for convolutional codes with repeat request », *IEEE Trans. on Inf. Theory*, vol. IT-26, Num.5, Sept. 1980.

[17] F. Monteiro, A. Dandache, A. M'sir and B. Lepley, « A fast CRC implementation on FPGA using a pipelined architecture for the polynomial division », *8th ICECS*, (St Julian, Malta), Sept. 2001.

[18] ETS, Digital Broadcasting for Television, Sound and Data Service, « Framing structure, channel coding and modulation for cable systems », PrETS 300 429, Dec. 1994.

[19] T.K Matsushima, T. Matsushima and S. Hirasawa, « Parallel encoder and decoder architectures for cyclic codes », *IEIEC Trans. on Fundamentals*, vol. E79-A, pp.1313-1323, Sept. 1996.

[20] T. Vallino,S. J. Piestrak, A. Dandache, F. Monteiro, B. Lepley, « Study of a new parallel architecture dedicated to the family of the difference set cyclic codes », *Proc. 5th IEEE Int. On-Line Testing Workshop*, (Rhodes, Greece), pp.237-239, July 1999.

[21] M. Braum et al., « Parrallel CRC computation in FPGA's », *Proc. 6th Int. Workshop on Field-Programmable Logic: Smart Applications, New Paradigms and Compilers*, (Spring Verlag, Darmstatdt, Germany), pp.156-165, Sept. 1996.

[22] T. B. Pei and C. Zukowski, « High speed parallel CRC circuits in VLSI », *IEEE Trans. Commun.*, vol. 40, pp. 653-657, April. 1992.

[23] B. Castagnolo and M. Rizzi, « High speed error correction circuit based on iterative cells »,*Int. J. Electronics*, vol. 14-4, pp.529-540, 1993.

[24] Kobayashi et al., « 50 MHz CMOS pipelined majority logic decoder for (1057,813) difference-set cyclic code », *IEICE Trans.*, vol. E79-A, pp. 1060-1067, July 1996.

[25] F. Monteiro, A. Dandache and B. Lepley, « Fast configurable polynomial division for error control coding applications », *in Proc., IOLTW 2001*, (Taormina, Italy), pp. 158-161, July 2001.

[26] Pierre Csillag, *Introduction aux codes correcteurs*, Edition Marketing, 1990.

[27] V. Cappellini, *Data Compression and Error Control Techniques with Applications*, Ed. Academic Press, 1985.

[28] A. J. Viterbi. « Error bounds for convolutional codes and an asymptotically optimum decoding algorithm ». *IEEE Trans. on Inf. Theory*, IT-13, 260-269, April. 1967.

[29] E. R. Berlekamp, R. E. Peile and S. P. Pope, « The application of error control to communication », *IEEE Commun. Mag.*, vol. 25, pp. 44-57, Avril 1987.

[30] J. P. Odenwalder and A. J. Viterbi, « Overview of existing and projected uses of coding in military satellite communications », *NTC Conf.-Rec.*, (Los Angeles, California), Dec. 1977.

[31] M. K. Simon and J. G. Smith, « Alternate symbol inversion for improved symbol synchronisation in convolutionally coded system » , *IEEE Trans. Commun.*, COM-28, Feb. 1980.

[32] Consultative Committee for Space Data Systems, « Recommendation for space and data systems standard: Telemetry channel coding », Blue Book, vol. 101.0-B-4, issue 4, May 1999.

[33] « OFDM-based 802.16.3.SUB 11 Ghz, BWA AIR INTERFACE Physical Layer Proposal », IEEE 802.16.3C-00/30 RL, Oct. 2000.

[34] J. L. Leray, C. Barillot et J. C. Boudenot, « Electronique spatiale : Les Nouveaux défis dus aux ceintures de radiation », *Revue de l'électricité et de l'électronique*, no. 09, pp. 54-62, Oct. 1999.

[35] B. Miller, « Satellites free the mobile phone », *IEEE Spectrum*, vol. 35, pp. 26-35, March 1998.

[36] J. G. WADE, *Codage et Traitement du Signal, L'Exemple des Systèmes Vidéo Numériques*, Ed. Masson, 1991.

[37] G. Ungerboeck and I. Csajka, « On improving data link performance by increasing the channel alphabet and introducing sequence coding », *IEEE Int. Symp. on Inf. Theory*, (Ronneby, Sweeden), June 1976.

[38] G. Ungerboeck, « Channel coding with multilevel/phase signals », *IEEE Trans. on Inf. Theory*, vol. IT-28, no. 1, pp. 55-67, Jan. 1982.

[39] P. Robertson and T. Worz, « A novel coded modulation scheme employing turbo codes », *URSI & ITG Conf.*, (Kleinheubach, Germany), Oct. 1995.

[40] D. Divsalar and F. Pollara, « On the design of turbo codes », *JPL TDA Progress Report 42-123*, (Pasadena, California), pp. 99-121, Nov. 1995.

[41] G. D. Forney, « Concatenated Codes », *MA: MIT Press*, Cambridge, 1966.

[42] ETS, Digital Broadcasting for Television, Sound and Data Service, « Framing structure, channel coding and modulation for 11/12 Ghz sattelite services », PrETS 300 421, Dec. 1994.

[43] ETS, Digital Broadcasting for Television, Sound and Data Service, « Framing structure, channel coding and modulation for terrestrial television , PrETS 300 744, Dec. 1994.

[44] P. Elias, « Error Free Coding », IRE Transaction on Inf. Theory, vol. IT-4, pp. 29-37, September 1954.

[45] Matthew C. Valenti, « An introduction to turbo codes », Bradley Dept. of Elect. Eng., Virginia Polytechnic Inst. S. U., (Blacksburg, Virginia).

[46] D. Divsalar and F. Pollara, « Turbo codes for deep space communications », *IEEE Commun. Theory Workshop*, (Santa Cruz, California), April. 1995.

[47] ITU-R International Mobile Telecommunication (IMT) 2000 Recommendation.

[48] E. Berutto, M. Gudmunson, R. Menolascino, W. Mohr, M. Pizarroso, « *Research activities on UMTS radio interface, network architecture and planning* », *IEEE Commun. Mag.*, vol. 36, Feb. 1998.

[49] « Digital Video Broadcasting (DVB): Interaction Channel for Sattelite Distribution System »,ETSI PR EN 301 790, V1.1.1, April. 2000.

[50] C. Douillart, M. Jezequel, C. Berrou and N. Brengarth, « The Turbo Code Standard for DVB-RCS », *The 2nd Int. Symp. on Turbo Codes*, (Brest, France), pp.535-538, Sept. 2000.

[51] R. Garello, R. Maggiora, G. Montosi, P. Coccia, S. Benedetto and A. Serra, « DSP implementation of turbo decoders for satellite communications », *6th Int. Workshop on Digital Signal Processing Techniques for Space Applications*, (Noordwijk, The Netherlands), Sept. 1998.

[52] H. Feldman and D. V. Ramana, « An introduction to INMARSAT's new mobile multimedia service », *6th Int. Mobile Satellite Conf.*, (Ottawa, Canada), pp. 226-229, June 1999.

[53] E. Trachtman and T. Hart, « Research elements leading to the development of INMARSAT's new mobile multimedia services », *6th Int. Mobile Satellite Conf.*, (Ottawa, Canada) pp. 209-212, June 1999.

[54] J. Broughton, J. Nemes, « Multimedia services for aeronautical mobile satellite applications », *6th Int. Mobile Satellite Conf.*, (Ottawa, Canada), pp. 1-4, June 1999.

[55] S. A. Barbulescu, R. Chang and S. Yaghmour, « Turbo codec - QPSK modem for INTELSAT digital services », *The 2nd Int. Symp. on Turbo Codes*, (Brest, France), pp. 487- 490, Sept. 2000.

[56] C. D. Edwards, C. T. Stelzried, L. J. Deutsch and L. Swanson, « NASA's deep-space telecommunications road map », *JPL TMO Progress Report 42-136*, (Pasadena, USA), pp. 1-20, Feb. 1999.

[57] W. Matsumoto, Y. Mi Yata and Narikawa, « G. gen : Performance evaluation of proposed TTCM (PCCC) with R-S code and without R-S code », *T1E1.4* (Huntsville, Canada), Aug. 2000.

[58] H. R. Sadjapour, R. Sonalkar and G. Jin, « Proposal on using multi-tone turbo trellis coded modulation », *T1E1.4* (Napa, California), Oct. 2000.

[59] « Asymmetric Digital Subsriber Line (ADSL) Transievers », ITU-Recommendation G.922.1, June 1999.

[60] A. Ghrayeb and W. E. Ryan, « Concatenated coding and iterative SOVA decoding with PR4 signaling », *Int. Conf. on Commun.*, pp. 597-601, June 2000.

[61] L. L. McPheters, S. W. McLaughlin and E. C. Hirsch, « Turbo codes for PR4 and EPR4 magnetic recording », *The 32nd Asilomar Conf. on Signals, Systems and Computers*, pp. 1778-1782, 1998.

[62] W. E. Ryan, « Performance of high rate turbo codes on a PR4-equalized magnetic recording channel », *Int. Conf. on Commun.*, pp. 947-951, June 1998.

[63] J. P. Odenwalder, *Error Control Coding handbook*, Linkabit Corporation, 1976.

[64] O. M. Collins, « The subtleties and intricacies of building a constraint lenght 15 convolutional decoder », *IEEE Trans. Commun.* , vol. 40, pp. 1810-1819, Dec. 1992.

[65] D. J. Costello, J. Hagenauer, H. Imai and S. B. Wicker, « Applications of error control coding », *IEEE Trans. Inf. Theory*, vol. 44, pp. 2531-2560, Oct. 1998.

[66] Boris Bartolomé, « Utilisation des turbo codes pour un système de communication multimédia par satellite », Thèse ENST Toulouse, Déc. 1999.

[67] S. Benedetto, and G. Montorsi, « Design of parallel concatenated convolutional codes », *IEEE Trans. on Commun.*, vol. 44, No. 5, pp. 591-600, May 1996.

[68] S. Dolinar and D. Divsalar, « Weight distributions for turbo codes using random and nonrandom permutations », *JPL TDA Progress Report 42-122*, Aug. 1995.

[69] D. Divsalar and R. J. McEliece, « On the design of concatenated coding systems with interleavers », *TMO progress report 42-134*, Aug. 1998.

[70] K. Tang, L. B. Milstein, and P. H. Siegel, « Combined MMSE interference suppression and turbo coding for a coherent DS-CDMA system », *IEEE J. Select. Areas Commun.*, Issue on The Turbo Principle: From Theory to Practice II, vol. 19, no. 9, pp. 1793-1803, Sept. 2001.

[71] T. Souvignier, A. Friedman, M. Oberg, P. H. Siegel, R. E. Swanson and J. K. Wolf, « Turbo codes for PR4: Parallel versus serial concatenation », *Proc. IEEE Int. Conf. on Commun.* , Paper S41-4, (Vancouver, Canada), June 1999.

[72] S. Benedetto and G. Montorsi. « Unveiling turbo codes : Some results on parallel concatenated coding schemes », *IEEE Trans. on Inf. Theory*, 42(2), 409-428, March 1996.

[73] S. Benedetto, and G. Montorsi, « Role of recursive convolutional codes in turbo codes », *Electronics Letters*, vol. 31, no. 11, pp. 858-859, May 1995.

[74] M. Oberg and P. Siegel, « Application of distance spectrum analysis to the improvement of turbo code performance », *Proc. 35th Annual Allerton Conf. on Commun., Control, and Computing*, pp. 701-710, Sept. 1997.

[75] D. Divsalar and F. Pollara, « Multiple turbo codes for deep-space communication », *JPL TDA Progress Report 42-121*, May 1995.

[76] E. Dunscombe and F. C. Piper, « Optimal interleaving scheme for convolutional codes », *Electronics Letters*, vol. 25, no. 22, pp. 1517-1518, Oct. 1989.

[77] K. Koora and H. Betzinger, « Interleaver design for turbo codes with selected inputs », *Commun. Laboratory, University of Technology*, (Dresden, Germany).

[78] Eric K. Hall and Stephen G. Wilson, « Convolutional interleavers for stream-oriented parallel concatenated convolutional codes », *ISIT*, (Cambridge, USA), Aug. 1998.

[79] Hugo M. Tullberg, Paul H. Siegel, « Interleaving techniques for coded modulation for mobile communications », *Center for Wireless Commun. Report.*

[80] M. C. Reed and S. S. Pietrobon, « Turbo code termination scheme and a novel alternative for short frames », *Proc. IEEE PIMRC*, pp. 354-358, 1996.

[81] A. Barbulescu and S. S. Pietrobon, « Terminating the trellis of turbo-codes in the Same State », *Electronics Letters*, vol. 31, pp. 22-23, Jan. 1995.

[82] W. Blackert, E. Hall, and S. Wilson, « Turbo code termination and interleaver conditions », *Electronics Letters*, vol. 31, pp. 2082-2083, Nov. 1995.

[83] A. Barbulescu and S. S. Pietrobon, « Interleaver design for turbo codes », *Electronics Letters*, vol. 30, pp. 2107-2108, Dec. 1994.

[84] C. Berrou, P. Combelles, P. Pénard, B. Talibart, « An IC for turbo-codes encoding and decoding », *Proc. IEEE Int. Solid- State Circuits Conf.*, pp. 90-91, 1995.

[85] A. Barbulescu and S. Pietrobon, « Rate compatible turbo codes » *Electronics Letters*, vol. 31, pp. 535-536, March 1995.

[86] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, « Soft-output decoding algorithms in iterative decoding of turbo codes », *TDA Progress Report 42-124*, Feb. 1996.

[87] P. Robertson, E. Villebrun, and P. Hoeher, « A comparison of optimal and sub-optimal decoding algorithms in the log domain », *Proc. ICC*, (Seattle, USA), pp. 1009-1013, juin 1995.

[88] L. Papke, P. Robertson and E. Villebrun, « Improved decoding with the SOVA in a parallel concatenated (turbo-code) scheme », *IEEE Int. Conf. on Commun.*, pp. 102-106, 1996.

[89] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, « Optimal decoding of linear codes for minimizing symbol error rate », *IEEE Trans. on Inf. Theory*, vol. IT-20, pp. 284-287, March 1974.

[90] O. Joerssen, M. Vaupel and H. Meyer, « High-speed VLSI architectures for soft-output viterbi decoding », *Proc. Application Specific Array Processors*, (Berkeley, CA), pp. 373-384, Aug. 1992.

[91] J. Vogt, K. Koora, A. Finger and G. Fetweis, « Comparison of different turbo decoder realizations for IMT-2000 », *Global Telecommunications Conf.*, pp. 2704-2708, 1999.

[92] H. Dawid and H. Meyer, « Real-time algorithms and VLSI architectures for soft output MAP convolutional decoding », *6th IEEE Int. Symp. on Personal, In-door and Mobile Radio Communications*, vol. 1, pp. 193-197, Sept. 1995.

[93] E. Yeo, S. Augsburger, W. R. Davis, and B. Nikolic, « Implementation of high throughput soft output viterbi decoders », *Proc. IEEE Workshop on Signal Processing Systems*, pp. 146-151, (San Diego, CA), Oct 2002.

[94] O. Joerssen and H. Meyr, « A 40 $Mb/s$ soft-output Viterbi decoder », *IEEE J. of Solid-State Circuits*, vol. 30, no. 7, pp. 812-18, July 1995.

[95] D. Garret and M. Stan, « Low power architecture of the soft-output viterbi algorithm », *Proc. IEEE ISPLED*, (Monterey, CA), pp. 262-7, Aug. 1998.

[96] G. Fettweis and H. Meyr, « High-speed parallel viterbi decoding : Algorithm and VLSI architecture », *IEEE Commun. Mag.*, vol. 29, no. 5, pp. 46-55, May 1991.

[97] H. F. Lin and D. G. Messerschmitt, « Algorithms and architectures for concurrent viterbi decoding », *Proc. ICC*, pp. 836-840, 1989.

[98] Yeung and J. M. Rabaey, « A 210 Mb/s radix-4 bit-level pipelined viterbi decoder », *Proc. IEEE ISPLED*, (Monterey, CA), pp. 88-9, 344, 440, Feb. 1995.

[99] P. J. Black and T. Meng, « A 1-Gb/s, four-state, sliding block viterbi decoder », *IEEE J. of Solid-State Circuits*, vol. 32, no. 6, pp. 797-805, June 1997.

[100] G. Fettweis, R. Karabed, P. H. Siegel and H. K. Thapar, « Reduced-complexity viterbi detector architectures for partial response signaling », *Proc. IEEE GLOBECOM*, pp. 559-563, Singapore, Nov. 1995.

[101] I. Lee and J. L. Sontag, « A New Architecure For the Fast Viterbi Algorithm », *Proc. IEEE GLOBECOM*, pp. 1664-1668, (San Francisco, CA), Nov. 2000.

[102] P. J. Black, Teresa H. Meng, « A 140 Mbits/s, 32 state, radix-4 viterbi decoder », *IEEE J. of Solid-State Circuits*, vol. 27. no. 12, Dec. 1992.

[103] Comatlas CAS 5093 40 Mb/s turbo code codec. Rev 4.1. May 1995.

[104] Small World Communications, « MAP04B 16 State MAP Decoder », Product Specification, Version 1.3 16 August, 1999.

[105] Small World Communications, « MAP04T very high speed MAP decoder », Product Specification, Version 3.2, Jan. 2002.

[106] Small World Communications, « MAP03T very high speed MAP decoder », Product Specification, Version 4.0, Jan. 2003.

[107] Small World Communications, « PCE03V 3GPP/3GPP2 turbo encoder », Product Specification, Version 1.1, Jan. 2003.

[108] Comtech AHA corporation, « Prodcut specification AHA4501 astro 36 Mbits/sec turbo code encoder/decoder »,Specification Product, 1998.

[109] Comtech AHA corporation, « AHA4522 astro LE 2K block turbo product code encoder/decoder », Specification Product.

[110] Comtech AHA corporation, « AHA4524 astro LE 4K block turbo product code encoder/decoder », Specification Product.

[111] Comtech AHA corporation, « AHA4525 IEEE 802.16a compliant turbo product code encoder/decoder », Specification Product, 2003.

[112] Comtech AHA corporation, « AHA4540 astro OC-3 155 Mbits/sec turbo product code encoder/decoder », Specification Product.

[113] P. Mannion, « Satellite systems gear up to meet the challenges of global networks ». Disponible sur http://www.elecdesign.com/magazine/2000, consulté en 2002.

[114] C. Douillard, M. Jezequel, C. Berrou, N. Brengarth, J. Tousch and N. Pham, « The turbo code standard for DVB-RCS », *Int. Symp. on Turbo Codes*, (Brest, France), Sept. 2000,

[115] iCoding Technology Incorporated, Product Summary, Nov. 2002.

[116] Xilinx Logic core, « IEEE 802.16 compatible turbo product code decoder v1.1 », Product Specification DS212, Jan. 2003.

[117] Amphion Semiconductor Ltd., « CS3530 Datasheet ».

[118] Amphion Semiconductor Ltd., « CS3630 Datasheet ».

[119] K. Gracie, S. Crozier and A. Hunt, « Performance of a low-complexity turbo decoder with a simple early stopping criterion implemented on a SHARC processor », *6th Int. Mobile Satellite Conf.*, (Ottawa, canada) pp. 281-286, June 1999.

[120] Amir Chass, Arik Gubeskys and Gideon Kutz, « Efficient software implementation of the Max-Log-MAP turbo decoder on the starCore SC140 DSP », *Int. Conf. on Signal Processing Applications and Technology*, (Dallas, USA), 16-19 oct. 2000.

[121] StarCore SC140 User Manual Rev. 0, Dec. 1999.

[122] G. Masera, G. Piccinini, M. R. Roch and M. Zamboni, « VLSI architectures for turbo codes », *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 3, pp. 369-379, Sept. 1999.

[123] P. Quinton et Y. Robert, *Algorithmes et Architectures Systoliques*, Ed. Masson, 1989.

[124] H. T. Kung, « Why systolic architectures? », *Computer 15, 1*, pp. 37-46, 1982.

[125] H. T. Kung, « Special purpose device for signal and image processing : An opportunity in VLSI », *Proc. SPIE*, Real Time and Signal Processing III, vol. 241, 1980.

[126] Y. Robert and M. Tchuente, « An efficient systolic array for the 1-D convolution problem », *J. of VLSI Computer Systems*, pp. 398-407, 1986.

[127] A. M'sir, A. Dandache, F. Monteiro and B. Lepley, « A high speed encoder for recursive systematic convolutive codes », *8th IEEE Int. On-Line Testing Workshop*, (Isle of Bendor, France), July 2002.

[128] G. Cohen, J. L. Dornstetter and P. Godlewski, *Codes Correcteurs D'Erreurs*, Ed. Masson, 1992.

[129] F. Thomoson Leighton, *Introduction To Parallel Algorithms and Architectures*, Morgan Kaufmann Publishers Inc., 1992.

**Site internet**

[130] http://www.themanualpage.org/glossaire, consulté en 2003.

[131] http://www.ccsds.org, consulté en 2003.

[132] http://www.sworld.com.au, consulté en 2003.

[133] http://www.aha.com, consulté en 2003.

[134] http://www.turboconcept.com/index, consulté en 2003.

[135] http://www.icoding.com, consulté en 2003.

[136] http://www.xilinx.com, consulté en 2003.

[137] http://www.crc.ca/fec, consulté en 2003.

[138] http://www.amphion.com, consulté en 2003.

# Annexe A

# Listes des Abréviations

| | | |
|---|---|---|
| **ACS** | : | Add Compare Select. |
| **ADSL** | : | Asymmetric Digital Subscriber Line. |
| **AMD** | : | Advanced Micro Devices. |
| **APP** | : | A Posteriori Probability. |
| **ARIB** | : | Association of Radio Industries and Business. |
| **ARQ** | : | Automatic Repeat reQuest. |
| **ATM** | : | Asynchronous Transfer Mode. |
| **AWGN** | : | Additive White Gaussian Noise. |
| **ASIC** | : | Application Specific Integrated Circuits. |
| **ASVC** | : | Application Specific Virtual Components. |
| **BCH** | : | Bose Hocquenghem Chaudhuri. |
| **BMC** | : | Branch Metric Calculation. |
| **BER** | : | Bit Error Rate. |
| **BVD** | : | Big Viterbi Decoder. |
| **BPSK** | : | Binary Phase Shift Keying. |
| **BSC** | : | Binary Symetric Channel. |
| **B-GAN** | : | Broadband Global Area Network. |
| **CCSDS** | : | Consultative Committee On Space Data System. |
| **CD** | : | Compact Disk. |
| **CDPD** | : | Cellular Digital Packet Data. |
| **CDMA** | : | Code Division Multiple Access. |
| **CMOS** | : | Complementary Metal-Oxide Semiconductor. |
| **CG** | : | Corps de Galois. |
| **CLs** | : | Cellules logiques. |
| **CRC** | : | Cyclic Redundancy Check. |
| **CSA** | : | Compare Select Add. |

**DVD** : Digital Versatile Disk.
**DVB** : Digital Video Broadcasting.
**DVB-RCS** : Digital Video Broadcasting-Return Channel Satellite.
**DVB-RCT** : Digital Video Broadcasting-Return Channel Terrestrial.
**DMT** : Digital Multitone.
**DSN** : Deep Space Network.
**DSCS** : Defense Satellite Communication System.
**ESA** : European Space Agency.
**ETSI** : European Telecommunications Standards Institute.
**FDDI** : Fiber Distributed Data Interface.
**FEC** : Forward Error Correction.
**FER** : Frame Error Rate.
**FIR** : Finite Impulse Response.
**FPGA** : Field Programmable Gate Array.
**GSM** : Global System for Mobile Communications.
**3GPP** : 3rd Generation Partnership Project.
**GPS** : Global positioning system.
**HIPERLAN** : High Performance Radio LAN.
**IEEE** : Institute of Electrical and Electronics Engineers.
**INTELSAT** : International Telecommunications Satellite.
**IMT-2000** : International Mobile Telecommunications-2000.
**IP** : Intellectual Property.
**QoS** : Quality of Service.
**IIR** : Infinite Impulse Response.
**LAN** : Local Area Network.
**LICM** : Laboratoire Interfaces Capteurs et Micro-électronique.
**LLR** : Log-Likelihood Ratio.
**LRV** : Logarithme du Rapport de Vraisemblance.
**MAP** : Maximum A Posteriori.
**MIPS** : Millions of Instructions Per Second.
**MTO** : Many To One.
**NASA** : National Aeronautics and Space Agency.
**NRSC** : Non-Récursif Systématique Convolutif.
**NRNSC** : Non-Récursif Non-Systématique Convolutif.
**PSDN** : Packet Switching Data Networks.
**OTM** : One To Many.
**RAM** : Random Access Memory.

| | | |
|---|---|---|
| **RM** | : | Reed-Muller. |
| **RS** | : | Reed-Solomon. |
| **RSC** | : | Récursif Systématique Convolutif. |
| **RTL** | : | Register Transfer Level. |
| **Sh** | : | Shannon. |
| **SMU** | : | Survivor Memory Unit. |
| **SoC** | : | System-on-a-Chip. |
| **SOVA** | : | Soft Output Viterbi Algorithm. |
| **SISO** | : | Soft Input Soft Output. |
| **SWP** | : | Sub-Word Parallel. |
| **TCM** | : | Trellis Coded Modulation. |
| **TDF** | : | Télédiffusion De France. |
| **TSMC** | : | Taiwan Semiconductor Manufacturing. |
| **TIA** | : | Telecommunication Industry Association. |
| **TTA** | : | Telecommunication Technology Association. |
| **UMTS** | : | Universal Mobile Telephone System. |
| **WAN** | : | Wide Area Network. |
| **WCDMA** | : | Wideband Code Division Multiple Access. |
| **Wi-Fi** | : | Wireless Fidelity. |
| **xDSL** | : | x Digital Subscriber Line. |

# Annexe B

# Notations

| | | |
|---|---|---|
| $a$, $a_i$ | : | amplitude. |
| $A_f$ | : | affaiblissement. |
| $A_e$, $A_s$ | : | amplitude respectives des signaux d'entrée et de sortie. |
| $A_{eff\ (p)}$ | : | augmentation effective du nombre de bits traités. |
| $B_p$ | : | bande passante. |
| $D(x)$ | : | séquence d'information (polynôme). |
| $E_b$ | : | énergie avant codage. |
| $E_c$ | : | énergie après codage. |
| $E[x]$ | : | espérance mathématique (fonction). |
| $E^g(x)$ | : | exposant de Gallager (fonction). |
| $E(x)$ | : | séquence d'érreur (polynôme). |
| $\mathbf{C}$, $\mathbf{C}_i$ | : | code. |
| $C_C$ | : | capacité du canal de transmission. |
| $CPX_{otm}$ | : | codeur parallel *One To Many*. |
| $CPX_{mto}$ | : | codeur parallel *Many To One*. |
| $CP1_X$ | : | bloc GEA implanté sous forme combinatoire. |
| $CP2_X$ | : | bloc GEA implanté sous forme mixte (combinatoire et pipeline). |
| $C_i$ | : | mot de code. |
| $C(x)$ | : | mot de code (polynôme). |
| **C(j,i)** | : | tableau de mémorisation de la séquence d'état du chemin compétiteur. |
| $D_S$ | : | débit d'information de la source. |
| $D_p$ | : | débit du circuit parallèle. |
| $d_i$ | : | information non codées. |
| $d_{min}$ | : | distance minimale du code. |
| $d_2^p$ | : | distance minimale de parité générée par une séquence de poids 2. |
| $d_{free}$ | : | distance minimale de liberté (code convolutif). |
| $f_{clk,par}$ | : | fréquence d'horloge du circuit parallèle. |
| $f_{clk,ser}$ | : | fréquence d'horloge du circuit série. |

| | | |
|---|---|---|
| $G$,$H$,$P$ | : | matrice génératrice. |
| $G(x)$,$H(x)$ | : | polynômes générateurs. |
| $\mathbf{H}(X)$ | : | entropie de X (fonction). |
| $\mathbf{H}(X,Y)$ | : | entropie conjointe (fonction). |
| $h(x)$ | : | quantité d'information de x. |
| $h(x;y)$ | : | quantité d'information mutuelle. |
| $I(X,Y)$ | : | information mutuelle. |
| $I_k$ | : | matrice identité. |
| $k$ | : | taille du bloc d'entrée (code bloc) ou nombre d'entrée (code convolutif). |
| $K$ | : | taille de l'entrelaceur. |
| $L$ | : | longueur de la séquence d'information incluant les bits de queue (code convolutif). |
| $L'$ | : | longueur de la séquence d'information sans les bits de queue (code convolutif). |
| $l_i$ | : | information intrinsèque. |
| $m$ | : | taille de la mémoire du codeur. |
| $M_p$ | : | matrice de perforation. |
| $n$ | : | taille du bloc de sortie (code bloc) ou nombre de sortie (code convolutif). |
| $N_0$ | : | énergie du bruit. |
| $\wp$ | : | degré de parallélisation. |
| $p(x)$ | : | probabilité de l'événement x. |
| $p_i$ | : | probabilité. |
| $P_e$ | : | probabilité d'erreur de décodage. |
| $P_b$ | : | probabilité d'erreur par bit. |
| $P_w$ | : | probabilité d'erreur par mot. |
| $P_S$ | : | puissance du signal. |
| $PE$ | : | processeur élémentaire. |
| $PE_b$ | : | processeur élémentaire bidirectionnel (systolique). |
| $PE_u$ | : | processeur élémentaire unidirectionnel (systolique). |
| $P_{e0}$ | : | processeur élémentaire orthogonal (systolique). |
| $R$ | : | rendement du code. |
| $\Re(x)$ | : | reste de la division. |
| $r_i$ | : | unité de mémorisation (bascule). |
| $(\aleph)_t$ | : | contenue du registre (ou bascule) $\aleph$ à l'instant t. |
| $s_0(t)$,$s_1(t)$ | : | signaux modulées. |
| $s_i$ | : | état. |
| $S_o$ | : | surface occupée. |
| $S_{(j,i)}$ | : | nœud du treillis. |
| $\mathbf{s}$ | : | séquence d'état. |
| $K_c$ | : | longueur de contrainte. |
| $T$,$T_{sys}$,$T_{clas}$ | : | période. |

| | | |
|---|---|---|
| $T_{max}$ | : | période maximale. |
| $T_t$,$T_{ti}$ | : | nombre de cycle additionnel à la terminaison du treillis. |
| $T_W$,$T_R$ | : | matrice de transition. |
| $T_o$,$T_m$ | : | matrice de transition OTM et MTO. |
| $t_e$ | : | nombre d'erreurs détectables et/ou corrigibles. |
| $X, Y$ | : | variables aléatoires. |
| $z_i$ | : | information a priori |
| $\tau_m$ | : | durée du symbole modulé. |
| $\mu$ | : | moyenne de la variable aléatoire. |
| $\sigma$ | : | écart type. |
| $\xi$ | : | réduction du taux de codage. |
| $\Lambda$ | : | logarithme du rapport de vraisemblance. |
| $\lambda(s_i \rightarrow s_j)$ | : | métrique de la branche $s_i \rightarrow s_j$ (Viterbi). |
| $\lambda_{ij}$ | : | métrique de la branche $s_i \rightarrow s_j$. |
| $\Gamma(j,i)$ | : | tableau des métriques cumulées. |
| $\mathbf{S}(j,i)$ | : | tableau de mémorisation de la séquence d'état. |
| $\Delta(j,i)$ | : | tableau de mémorisation de la différence entre les métriques. |
| | | du chemin survivant et de son compétiteur. |
| $\gamma(si \rightarrow s_{i+1})$ | : | métrique de la branche $si \rightarrow s_{i+1}$ (MAP). |
| $\alpha(j,i)$ | : | tableau de mémorisation des probabilités $\alpha(si)$. |
| $\alpha_\ell(j,i)$ | : | tableau de mémorisation des probabilités $\alpha_\ell(si)$. |
| $\beta(j,i)$ | : | tableau de mémorisation des probabilités $\beta(si)$. |
| $\beta_\ell(j,i)$ | : | tableau de mémorisation des probabilités $\beta(si)_\ell$. |
| $\Omega$ | : | l'ensemble des états $s_{i-1}$ connectés à l'état $s_i$ (Viterbi). |
| $\Omega_f$ | : | l'ensemble des états $s_{i-1}$ connectés à l'état $s_i$ (MAP). |
| $\Omega_b$ | : | l'ensemble des états $s_{i+1}$ connectés à l'état $s_i$ (MAP). |
| $\alpha_\ell(s_i)$ | : | $\ln(\alpha(s_i))$. |
| $\Phi_k$ | : | état anticipé. |
| $\triangle f_{x,\wp}$ | : | pourcentage de dégradation en fonction de $\wp$. |
| $\varrho$ | : | nombre des sommes de contrôle de parité. |
| $\Psi_i$ | : | syndrome. |
| $\psi_i$ | : | bit du syndrome. |
| $()_8$ | : | représentation octale. |
| $()_{16}$ | : | représentation hexadécimale. |

# Annexe C

# Résultats Expérimentaux

# Diviseurs

| | $p$=8 | $p$=8 | $p$=16 | $p$=16 | $p$=32 | $p$=32 |
|---|---|---|---|---|---|---|
| **Implantation** | $DP_{comb}$ | $DP_{pipe}$ | $DP_{comb}$ | $DP_{pipe}$ | $DP_{comb}$ | $DP_{pipe}$ |
| $G_1(x)$ | 14 CLs<br>250 MHz<br>1,95 Gbits/s | 53 CLs<br>250 MHz<br>1,95 Gbits/s | 22 CLs<br>250 MHz<br>3,81 Gbits/s | 169 CLs<br>250 MHz<br>3,81 Gbits/s | 50 CLs<br>181.81 MHz<br>5,41 Gbits/s | 593 CLs<br>250 MHz<br>7,45 Gbits/s |
| $G_2(x)$ | 13 CLs<br>250 MHz<br>1,95 Gbits/s | 60 CLs<br>250 MHz<br>1,95 Gbits/s | 27 CLs<br>222.22 MHz<br>3,39 Gbits/s | 184 CLs<br>250 MHz<br>3,81 Gbits/s | 52 CLs<br>178.57 MHz<br>5,32 Gbits/s | 625 CLs<br>250 MHz<br>7,45 Gbits/s |
| $G_3(x)$ | 19 CLs<br>250 MHz<br>1,95 Gbits/s | 66 CLs<br>250 MHz<br>1,95 Gbits/s | 31 CLs<br>217.39 MHz<br>3,31 Gbits/s | 198 CLs<br>250 MHz<br>3,81 Gbits/s | 62 CLs<br>196.07 MHz<br>5,84 Gbits/s | 654 CLs<br>250 MHz<br>7,45 Gbits/s |
| $G_4(x)$ | 28 CLs<br>243.9 MHz<br>1,90 Gbits/s | 96 CLs<br>250 MHz<br>1,95 Gbits/s | 55 CLs<br>222.22 MHz<br>3,39 Gbits/s | 265 CLs<br>250 MHz<br>3,81 Gbits/s | 87 CLs<br>163.93 MHz<br>4,88 Gbits/s | 778 CLs<br>250 MHz<br>7,45 Gbits/s |
| $G_5(x)$ | 39 CLs<br>212.76 MHz<br>1,66 Gbits/s | 94 CLs<br>188.67 MHz<br>1,47 Gbits/s | 63 CLs<br>166.66 MHz<br>2,54 Gbits/s | 326 CLs<br>178.57 MHz<br>2,72 Gbits/s | 147 CLs<br>86.95 MHz<br>2,59 Gbits/s | 1006 CLs<br>227.27 MHz<br>3,79 Gbits/s |
| $G_6(x)$ | 27 CLs<br>250 MHz<br>1,95 Gbits/s | 88 CLs<br>250 MHz<br>1,95 Gbits/s | 58 CLs<br>185.18 MHz<br>2,82 Gbits/s | 348 CLs<br>188.67 MHz<br>2,87 Gbits/s | 140 CLs<br>97.08 MHz<br>2,89 Gbits/s | 1044 CLs<br>156.25 MHz<br>4,65 Gbits/s |
| $G_7(x)$ | 59 CLs<br>92.97 MHz<br>0,72 Gbits/s | 167 CLs<br>94.33 MHz<br>0,73 Gbits/s | 146 CLs<br>121.95 MHz<br>1,86 Gbits/s | 439 CLs<br>149.25 MHz<br>2,27 Gbits/s | 333 CLs<br>83.33 MHz<br>2,48 Gbits/s | 1615 CLs<br>91.74 MHz<br>2,73 Gbits/s |
| $G_8(x)$ | 59 CLs<br>188.67 MHz<br>1,47 Gbits/s | 126 CLs<br>196.07 MHz<br>1,53 Gbits/s | 99 CLs<br>125 MHz<br>1,90 Gbits/s | 360 CLs<br>172.41 MHz<br>2,63 Gbits/s | 348 CLs<br>68.02 MHz<br>2,02 Gbits/s | 1478 CLs<br>120.48 MHz<br>3,59 Gbits/s |
| $G_9(x)$ | 102 CLs<br>97.08 MHz<br>0,75 Gbits/s | 203 CLs<br>102.04 MHz<br>0,79 Gbits/s | 169 CLs<br>89.28 MHz<br>1,36 Gbits/s | 477 CLs<br>133.85 MHz<br>2,04 Gbits/s | 295 CLs<br>74.62 MHz<br>2,22 Gbits/s | 1639 CLs<br>144.92 MHz<br>4,31 Gbits/s |

TAB. C.1 – *Diviseurs parallèles $DP_{comb}$ $DP_{pipe}$.*

# Annexe D

# Résultats Expérimentaux
# Codeurs convolutifs

| R=1/2 | $\wp$=8 | $\wp$=8 | $\wp$=16 | $\wp$=16 | $\wp$=32 | $\wp$=32 |
|---|---|---|---|---|---|---|
| **Implantation** | $CP1_{mto}$ | $CP2_{mto}$ | $CP1_{mto}$ | $CP2_{mto}$ | $CP1_{mto}$ | $CP2_{mto}$ |
| Codeur(1,2,2)(7,5)$_8$ | 151 CLs<br>227 MHz<br>1.69 Gbits/s | 189 CLs<br>250 MHz<br>1.86 Gbits/s | 495 CLs<br>196.06 MHz<br>2.92 Gbits/s | 636 CLs<br>250 MHz<br>3.72 Gbits/s | 1756 CLs<br>109.89 MHz<br>3.27 Gbits/s | 2301 CLs<br>222 MHz<br>6.61 Gbits/s |
| Codeur(1,2,3)(13,17)$_8$ | 158 CLs<br>250 MHz<br>1.86 Gbits/s | 199 CLs<br>250 MHz<br>1.86 Gbits/s | 512 CLs<br>169.49 MHz<br>2.53 Gbits/s | 661 CLs<br>250 MHz<br>3.72 Gbits/s | 1796 CLs<br>113.63 MHz<br>3.39 Gbits/s | 2360 CLs<br>208.33 MHz<br>6.21 Gbits/s |
| Codeur(1,2,4)(23,33)$_8$ | 165 CLs<br>250 MHz<br>1.86 Gbits/s | 213 CLs<br>250 MHz<br>1.86 Gbits/s | 526 CLs<br>140.84 MHz<br>2.1 Gbits/s | 688 CLs<br>250 MHz<br>3.72 Gbits/s | 1828 CLs<br>95.23 MHz<br>2.84 Gbits/s | 2413 CLs<br>208.33 MHz<br>6.21 Gbits/s |
| Codeur(1,2,4)(23,25)$_8$ | 169 CLs<br>175.43 MHz<br>1.31 Gbits/s | 215 CLs<br>250 MHz<br>1.86 Gbits/s | 528 CLs<br>153.84 MHz<br>2.29 Gbits/s | 689 CLs<br>250 MHz<br>3.72 Gbits/s | 1829 CLs<br>119.04 MHz<br>3.55 Gbits/s | 2414 CLs<br>200 MHz<br>5.96 Gbits/s |

TAB. D.1 – *Codeurs parallèles RSC de rendement* $1/2$.

| R=1/3 | $\wp$=8 | $\wp$=8 | $\wp$=16 | $\wp$=16 | $\wp$=32 | $\wp$=32 |
|---|---|---|---|---|---|---|
| **Implantation** | $CP1_{mto}$ | $CP2_{mto}$ | $CP1_{mto}$ | $CP2_{mto}$ | $CP1_{mto}$ | $CP2_{mto}$ |
| Codeur(1,3,3)(7,5,3)$_8$ | 203 CLs<br>250 MHz<br>1.86 Gbits/s | 241 CLs<br>250 MHz<br>1.86 Gbits/s | 663 CLs<br>250 MHz<br>3.72 Gbits/s | 804 CLs<br>250 MHz<br>3.72 Gbits/s | 2348 CLs<br>99 MHz<br>2.95 Gbits/s | 2893 CLs<br>217.39 MHz<br>6.48 Gbits/s |
| Codeur(1,3,3)(13,17,15)$_8$ | 224 CLs<br>217.39 MHz<br>1.62 Gbits/s | 268 CLs<br>192.30 MHz<br>1.43 Gbits/s | 710 CLs<br>175.43 MHz<br>2.61 Gbits/s | 869 CLs<br>204.08 MHz<br>3.04 Gbits/s | 2447 CLs<br>114.94 MHz<br>3.42 Gbits/s | 3033 CLs<br>131.57 MHz<br>3.92 Gbits/s |
| Codeur(1,3,4)(23,33,37)$_8$ | 227 CLs<br>212.76 MHz<br>1.58 Gbits/s | 274 CLs<br>204.08 MHz<br>1.52 Gbits/s | 707 CLs<br>144.92 MHz<br>2.16 Gbits/s | 873 CLs<br>166.66 MHz<br>2.48 Gbits/s | - CLs<br>- MHz<br>- Gbits/s | - CLs<br>- MHz<br>- Gbits/s |
| Codeur(1,3,4)(23,25,37)$_8$ | 231 CLs<br>185.18 MHz<br>1.38 Gbits/s | 227 CLs<br>217.39 MHz<br>1.62 Gbits/s | 710 CLs<br>161.29 MHz<br>2.40 Gbits/s | 875 CLs<br>227.27 MHz<br>3.39 Gbits/s | 2450 CLs<br>126.58 MHz<br>3.77 Gbits/s | 3039 CLs<br>161.29 MHz<br>4.80 Gbits/s |

TAB. D.2 – *Codeurs parallèles RSC de rendement* $1/3$.

| R=1/4 | $\wp$=8 | $\wp$=8 | $\wp$=16 | $\wp$=16 | $\wp$=32 | $\wp$=32 |
|---|---|---|---|---|---|---|
| **Implantation** | $CP1_{mto}$ | $CP2_{mto}$ | $CP1_{mto}$ | $CP2_{mto}$ | $CP1_{mto}$ | $CP2_{mto}$ |
| Codeur(1,4,3)(13,17,15,11)$_8$ | 227 CLs<br>204.08 MHz<br>1.52 Gbits/s | 323 CLs<br>196.07 MHz<br>1.46 Gbits/s | 882 CLs<br>175.43 MHz<br>2.61 Gbits/s | 1039 CLs<br>181.81 MHz<br>2.71 Gbits/s | 3039 CLs<br>131.57 MHz<br>3.92 Gbits/s | 3631 CLs<br>149.25 MHz<br>4.44 Gbits/s |
| Codeur(1,4,4)(23,35,27,37)$_8$ | 289 CLs<br>163.93 MHz<br>1.22 Gbits/s | 329 CLs<br>250 MHz<br>1.86 Gbits/s | 881 CLs<br>158.73 MHz<br>2.36 Gbits/s | 1045 CLs<br>232.55 MHz<br>3.47 Gbits/s | 3046 CLs<br>114.94 MHz<br>3.42 Gbits/s | 3632 CLs<br>196.07 MHz<br>5.84 Gbits/s |
| Codeur(1,4,4)(23,33,37,25)$_8$ | 284 CLs<br>192.30 MHz<br>1.43 Gbits/s | 329 CLs<br>185.18 MHz<br>1.37 Gbits/s | 877 CLs<br>175.43 MHz<br>2.61 Gbits/s | 1042 CLs<br>172.41 MHz<br>2.57 Gbits/s | 3044 CLs<br>133.33 MHz<br>3.97 Gbits/s | 3631 CLs<br>120.48 MHz<br>3.59 Gbits/s |

TAB. D.3 – *Codeurs parallèles RSC de rendement* 1/4.

| R=1/2 | $\wp$=8 | $\wp$=8 | $\wp$=16 | $\wp$=16 | $\wp$=32 | $\wp$=32 |
|---|---|---|---|---|---|---|
| **Implantation** | $CP1_{mto}$ | $CP2_{mto}$ | $CP1_{mto}$ | $CP2_{mto}$ | $CP1_{mto}$ | $CP2_{mto}$ |
| Codeur(1,2,2)(5)$_8$ | 135 CLs<br>250 MHz<br>1.86 Gbits/s | 172 CLs<br>250 MHz<br>1.86 Gbits/s | 459 CLs<br>250 MHz<br>3.72 Gbits/s | 610 CLs<br>250 MHz<br>3.73 Gbits/s | 1683 CLs<br>250 MHz<br>7.45 Gbits/s | 2242 CLs<br>196.07 MHz<br>5.84 Gbits/s |
| Codeur(1,2,3)(17)$_8$ | 138 CLs<br>250 MHz<br>1.86 Gbits/s | 178 CLs<br>250 MHz<br>1.86 Gbits/s | 462 CLs<br>250 MHz<br>3.72 Gbits/s | 630 CLs<br>250 MHz<br>3.73 Gbits/s | 1686 CLs<br>250 MHz<br>7.45 Gbits/s | 2278 CLs<br>204 MHz<br>6.08 Gbits/s |
| Codeur(1,2,4)(25)$_8$ | 142 CLs<br>250 MHz<br>1.86 Gbits/s | 181 CLs<br>250 MHz<br>1.86 Gbits/s | 466 CLs<br>250 MHz<br>3.72 Gbits/s | 649 CLs<br>250 MHz<br>3.73 Gbits/s | 1690 CLs<br>250 MHz<br>7.45 Gbits/s | 2313 CLs<br>222.22 MHz<br>6.62 Gbits/s |

TAB. D.4 – *Codeurs parallèles NRSC de rendement* 1/2.

| R=1/3 | $\wp$=8 | $\wp$=8 | $\wp$=16 | $\wp$=16 | $\wp$=32 | $\wp$=32 |
|---|---|---|---|---|---|---|
| **Implantation** | $CP1_{mto}$ | $CP2_{mto}$ | $CP1_{mto}$ | $CP2_{mto}$ | $CP1_{mto}$ | $CP2_{mto}$ |
| Codeur(1,3,2)(5,3)$_8$ | 187 CLs<br>250 MHz<br>1.86 Gbits/s | 225 CLs<br>250 MHz<br>1.86 Gbits/s | 627 CLs<br>250 MHz<br>3.72 Gbits/s | 779 CLs<br>250 MHz<br>3.73 Gbits/s | 2275 CLs<br>208.33 MHz<br>6.21 Gbits/s | 2835 CLs<br>212.76 MHz<br>6.34 Gbits/s |
| Codeur(1,3,3)(17,15)$_8$ | 190 CLs<br>250 MHz<br>1.86 Gbits/s | 230 CLs<br>250 MHz<br>1.86 Gbits/s | 630 CLs<br>250 MHz<br>3.72 Gbits/s | 798 CLs<br>250 MHz<br>3.73 Gbits/s | 2270 CLs<br>204.08 MHz<br>6.21 Gbits/s | 2870 CLs<br>208.33 MHz<br>6.21 Gbits/s |
| Codeur(1,3,4)(33,37)$_8$ | 194 CLs<br>250 MHz<br>1.86 Gbits/s | 242 CLs<br>212.76 MHz<br>1.58 Gbits/s | 634 CLs<br>250 MHz<br>3.72 Gbits/s | 834 CLs<br>250 MHz<br>3.73 Gbits/s | 2282 CLs<br>212.76 MHz<br>6.34 Gbits/s | 2930 CLs<br>163.84 MHz<br>4.88 Gbits/s |

TAB. D.5 – *Codeurs parallèles NRSC de rendement* 1/3.

| R=1/4 | $\wp$=8 | $\wp$=8 | $\wp$=16 | $\wp$=16 | $\wp$=32 | $\wp$=32 |
|---|---|---|---|---|---|---|
| **Implantation** | $CP1_{mto}$ | $CP2_{mto}$ | $CP1_{mto}$ | $CP2_{mto}$ | $CP1_{mto}$ | $CP2_{mto}$ |
| Codeur(1,4,3)(17,15,11)$_8$ | 242 CLs<br>250 MHz<br>1.86 Gbits/s | 282 CLs<br>250 MHz<br>1.86 Gbits/s | 798 CLs<br>250 MHz<br>3.72 Gbits/s | 966 CLs<br>250 MHz<br>3.72 Gbits/s | 2870 CLs<br>200 MHz<br>5.96 Gbits/s | 3462 CLs<br>188.67 MHz<br>5.62 Gbits/s |
| Codeur(1,4,4)(35,27,37)$_8$ | 254 CLs<br>250 MHz<br>1.86 Gbits/s | 294 CLs<br>250 MHz<br>1.86 Gbits/s | 818 CLs<br>243.9 MHz<br>3.63 Gbits/s | 1002 CLs<br>250 MHz<br>3.72 Gbits/s | 2906 CLs<br>196.07 MHz<br>5.84 Gbits/s | 3530 CLs<br>200 MHz<br>5.96 Gbits/s |
| Codeur(1,4,4)(33,37,25)$_8$ | 254 CLs<br>250 MHz<br>1.86 Gbits/s | 294 CLs<br>238.09 MHz<br>1.77 Gbits/s | 818 CLs<br>250 MHz<br>3.72 Gbits/s | 1002CLs<br>250 MHz<br>3.72 Gbits/s | 2906 CLs<br>196.07 MHz<br>5.84 Gbits/s | 3530 CLs<br>181.81 MHz<br>5.41 Gbits/s |

TAB. D.6 – *Codeurs parallèles NRSC de rendement* 1/4.

| $CP2_{otm}$ : R=1/2 | $p$=8 | $p$=16 | $p$=32 |
|---|---|---|---|
| Codeur(1,2,16)($G_5(x)$, $H_1(x)$) | 265 CLs<br>204,08 MHz<br>1,52 Gbits/s | 906 CLs<br>185,18 MHz<br>2,75 Gbits/s | 3010 CLs<br>108,69 MHz<br>3,23 Gbits/s |
| Codeur(1,2,16)($G_{14(x)}$, $H_1(x)$) | 272 CLs<br>250 MHz<br>1,86 Gbits/s | 900 CLs<br>200 MHz<br>2,98 Gbits/s | 3022 CLs<br>126,58 MHz<br>3,77 Gbits/s |
| Codeur(1,2,32)($G_7(x)$, $H_2(x)$) | 349 CLs<br>192 MHz<br>1,43 Gbits/s | 1142 CLs<br>156,25 MHz<br>2,32 Gbits/s | 3649 CLs<br>103,09 MHz<br>3,07 Gbits/s |
| Codeur(1,2,32)($G_8(x)$, $H_2(x)$) | 335 CLs<br>250 MHz<br>1,86 Gbits/s | 1086 CLs<br>212,76 MHz<br>3,17 Gbits/s | 3590 CLs<br>128,2 MHz<br>3,82 Gbits/s |

TAB. D.7 – *Codeurs parallèles OTM RSC de rendement* $1/2$.

| $CP2_{otm}$ : R=1/2 | $p$=8 | $p$=16 | $p$=32 |
|---|---|---|---|
| Codeur(1,2,16)($G_5(x)$, $H_1(x)$) | 310 CLs<br>172,47 MHz<br>1,28 Gbits/s | 985 CLs<br>144 MHz<br>2,14 Gbits/s | 3160 CLs<br>107 MHz<br>3,18 Gbits/s |
| Codeur(1,2,16)($G_6(x)$, $H_1(x)$) | 298 CLs<br>227 MHz<br>1,69 Gbits/s | 979 CLs<br>156 MHz<br>2,43 Gbits/s | 3233 CLs<br>119,25 MHz<br>3,55 Gbits/s |
| Codeur(1,2,32)($G_7(x)$,$H_2(x)$) | 472 CLs<br>109 MHz<br>0,81 Gbits/s | 1356 CLs<br>147 MHz<br>2,29 Gbits/s | 4233 CLs<br>105,26 MHz<br>3,13 Gbits/s |
| Codeur(1,2,32)($G_8(x)$, $H_2(x)$) | 444 CLs<br>158 MHz<br>1,17 Gbits/s | 1238 CLs<br>158,73 MHz<br>2,48 Gbits/s | 4056 CLs<br>94,33 MHz<br>2,81 Gbits/s |

TAB. D.8 – *Codeurs parallèles MTO RSC de rendement* $1/2$.

# Annexe E

# Résultats Expérimentaux Turbo-codeurs convolutifs

| K | $\wp$ | TC1 | TC2 | TC3 |
|---|---|---|---|---|
| 256 | 8 | 250 MHz<br>1118 CLs<br>1,86 Gbits/s | 238,09 MHz<br>1684 CLs<br>1,77 Gbits/s | 238 MHz<br>1198 CLs<br>1,77 Gbits/s |
| 256 | 16 | 192,3 MHz<br>2476 CLs<br>2,87 Gbits/s | 151,51 MHz<br>2578 CLs<br>2,26 Gbits/s | 151,51 MHz<br>2624 CLs<br>2,26 Gbits/s |
| 256 | 32 | 138,88 MHz<br>7502 CLs<br>4,15 Gbits/s | 117,64 MHz<br>7698 CLs<br>3,51 Gbits/s | 136,98 MHz<br>7818 CLs<br>4,09 Gbits/s |
| 512 | 8 | 250 MHz<br>1632 CLs<br>1,86 Gbits/s | 232,55 MHz<br>1680 CLs<br>1,73 Gbits/s | 217,39 MHz<br>1712 CLs<br>1,62 Gbits/s |
| 512 | 16 | 204,08 MHz<br>2985 CLs<br>3,04 Gbits/s | 135,13 MHz<br>3087 CLs<br>2,01 Gbits/s | 178,57 MHz<br>3143 CLs<br>2,66 Gbits/s |
| 512 | 32 | 142,85 MHz<br>8003 CLs<br>4,26 Gbits/s | 109 MHz<br>8215 CLs<br>3,25 Gbits/s | 123,45 MHz<br>8319 CLs<br>3,68 Gbits/s |
| 1024 | 8 | 196,07 MHz<br>2626 CLs<br>1,46 Gbits/s | 161,29 MHz<br>2718 CLs<br>1,20 Gbits/s | 156,25 MHz<br>2750 CLs<br>1,16 Gbits/s |
| 1024 | 16 | 227,27 MHz<br>4016 CLs<br>3,39 Gbits/s | 140,88 MHz<br>4130 CLs<br>2,10 Gbits/s | 169,49 MHz<br>4186 CLs<br>2,53 Gbits/s |
| 1024 | 32 | 147,05 MHz<br>9053 CLs<br>4,39 Gbits/s | 104,16 MHz<br>9249 CLs<br>3,11 Gbits/s | 135,13 MHz<br>9369 CLs<br>4,03 Gbits/s |
| | | Code constituant<br>RSC-TC1<br>$(1,2,2)(7,5)_8$ | Code constituant<br>RSC-TC2<br>$(1,2,3)(13,17)_8$ | Code constituant<br>RSC-TC3<br>$(1,2,4)(23,33)_8$ |
| - | 8 | 250 MHz<br>300 CLs<br>1,86 Gbits/s | 227,27 MHz<br>324 CLs<br>1,69 Gbits/s | 250 MHz<br>340 CLs<br>1,86 Gbits/s |
| - | 16 | 250 MHz<br>979 CLs<br>3,73 Gbits/s | 204,08 MHz<br>1030 CLs<br>3,04 Gbits/s | 217,39 MHz<br>1058 CLs<br>3,24 Gbits/s |
| - | 32 | 196,07 MHz<br>3492 CLs<br>5,86 Gbits/s | 169,49 MHz<br>3590 CLs<br>5,06 Gbits/s | 185,18 MHz<br>3650 CLs<br>5,53 Gbits/s |

TAB. E.1 – *Turbo-codeur à deux étages de rendement* $R = 1/3$.

# Annexe F

# Résultats Expérimentaux Décodeurs de syndromes

Matrice d'interconnexion $M1 =$

$$\begin{pmatrix} 10101010 \\ 01010101 \\ 10101010 \\ 01010101 \\ 10101010 \\ 01010101 \\ 10101010 \\ 01010101 \end{pmatrix}$$

| R=1/2 | $\wp$=8 | $\wp$=16 | $\wp$=32 |
|---|---|---|---|
| m=2 | 39 CLs<br>250 MHz<br>1.95 Gbits/s | 79 CLs<br>250 MHz<br>3.9 Gbits/s | 159 CLs<br>250 MHz<br>7.8 Gbits/s |
| m=3 | 45 CLs<br>250 MHz<br>1.95 Gbits/s | 94 CLs<br>250 MHz<br>3.9 Gbits/s | 189 CLs<br>250 MHz<br>7.8 Gbits/s |
| m=4 | 53 CLs<br>250 MHz<br>1.39 Gbits/s | 109 CLs<br>250 MHz<br>2.64 Gbits/s | 221 CLs<br>250 MHz<br>5.48 Gbits/s |
| m=5 | 73 CLs<br>222.22 MHz<br>1.73 Gbits/s | 172 CLs<br>192.30 MHz<br>3.0 Gbits/s | 343 CLs<br>196.07 MHz<br>6.12 Gbits/s |
| m=6 | 101 CLs<br>178.57 MHz<br>1.39 Gbits/s | 214 CLs<br>169.49 MHz<br>2.64 Gbits/s | 437 CLs<br>175.45 MHz<br>5.48 Gbits/s |

TAB. F.1 – *Réseaux de correction de rendement* $1/2$.

| R=1/3 | $\wp$=8 | $\wp$=16 | $\wp$=32 |
|---|---|---|---|
| m=2 | 70CLs<br>250 MHz<br>1.95 Gbits/s | 142 CLs<br>250 MHz<br>3.9 Gbits/s | 586 CLs<br>250 MHz<br>7.8 Gbits/s |
| m=3 | 104 CLs<br>238 MHz<br>1.85 Gbits/s | 220 CLs<br>208 MHz<br>3.25 Gbits/s | 440CLs<br>222 MHz<br>6.93 Gbits/s |
| m=4 | 239 CLs<br>188.1 MHz<br>1.46 Gbits/s | 289 CLs<br>185 MHz<br>2.89 Gbits/s | 597 CLs<br>166.66 MHz<br>5.2 Gbits/s |
| m=5 | 239 CLs<br>100 MHz<br>0.78 Gbits/s | 498 CLs<br>99 MHz<br>1.54 Gbits/s | 1001 CLs<br>93.45 MHz<br>2.92 Gbits/s |
| m=6 | 330 CLs<br>75.75 MHz<br>0.59 Gbits/s | 663 CLs<br>72.46 MHz<br>1.13 Gbits/s | 1354 CLs<br>73.525 MHz<br>2.29 Gbits/s |

TAB. F.2 – *Réseaux de correction de rendement* $1/3$.

| R=1/4 | $\wp$=8 | $\wp$=16 | $\wp$=32 |
|---|---|---|---|
| m=2 | 140 CLs<br>188.67 MHz<br>1.47 Gbits/s | 284 CLs<br>181.81 MHz<br>2.84 Gbits/s | 572 CLs<br>172.41 MHz<br>5.38 Gbits/s |
| m=3 | 169 CLs<br>151.51 MHz<br>1.18 Gbits/s | 362 CLs<br>163.93 MHz<br>2.56 Gbits/s | 721 CLs<br>133.33 MHz<br>4.16 Gbits/s |
| m=4 | 375 CLs<br>64.93 MHz<br>0.507 Gbits/s | 772 CLs<br>61.72 MHz<br>0.96 Gbits/s | 1152 CLs<br>60.60 MHz<br>1.89 Gbits/s |
| m=5 | 463 CLs<br>56.17 MHz<br>0.43 Gbits/s | 926CLs<br>62.11 MHz<br>1.97 Gbits/s | 1354 CLs<br>95.23 MHz<br>2.97 Gbits/s |

TAB. F.3 – *Réseaux de correction de rendement* $1/4$.

| R=1/2 | $\wp$=8 | $\wp$=16 | $\wp$=32 |
|---|---|---|---|
| Décodeur 1<br>$(1,2,2)(7,5)_8$<br>$M1$ | 416 CLs<br>250 MHz<br>1.95 Gbits/s | 1344 CLs<br>196 MHz<br>3.06 Gbits/s | 4749 CLs<br>153 MHz<br>4.78 Gbits/s |
| Décodeur 2<br>$(1,2,3)(13,17)_8$<br>$M1$ | 430 CLs<br>204 MHz<br>1.59 Gbits/s | 1380 CLs<br>222 MHz<br>3.46 Gbits/s | 4809 CLs<br>129 MHz<br>4.03 Gbits/s |
| Décodeur 3<br>$(1,2,4)(23,33)_8$<br>$M1$ | 469 CLs<br>232 MHz<br>1.81Gbits/s | 1457 CLs<br>204 MHz<br>3.14 Gbits/s | 4977 CLs<br>129.87MHz<br>4.05Gbits/s |

TAB. F.4 – *Décodeurs de rendement* $1/2$.

# Annexe G

# Listes des publications

- A. M'sir, F. Monteiro, A. Dandache et B. Lepley, « Design of a high speed parallel encoder for convolutional codes », Microelectronics Journal, Elsevier Science, en cour de publication.
- A. M'sir, A. Dandache, F. Monteiro et B. Lepley, « Designing fast parallel architectures for turbo encoders », *1st Northeast Workshop on Circuits and Systems*, (Montreal, Canada), 17-20 June 2003.
- A. M'sir, A. Dandache, F. Monteiro et B. Lepley, « A high speed architecture for turbo encoders », *6th Int. Workshop on Design and Diagnostics of Electronic Circuits and Systems*, (Poznan, Pologne), 14-16 April 2003.
- A. M'sir, A. Dandache, F. Monteiro et B. Lepley, « Fast parallel architecture for syndrome convolutional decoder », *14th IEEE Int. Conf. on Microelectronics*, (Beyrout, Liban), 11-13 Dec. 2002.
- A. M'sir, F. Monteiro, A. Dandache et B. Lepley, « A parallel pipelined architecture for recursive and non recursive convolutional encoders », *17th Int. Conf. on Design of Circuit and Integrated Systems*, (Santander, Spain),19-22 Nov. 2002.
- A. M'sir, F. Monteiro, A. Dandache et B. Lepley, « A high speed encoder for recursive systematic convolutive Codes », *8th IEEE Int. On-Line Testing Workshop*, (Ile de Bendor, France), 8-10 Jul. 2002.
- F. Monteiro, A. Dandache, A. M'sir et B. Lepley, « A polynomial division pipelined architecture for CRC error detecting codes », *13th IEEE Int. Conf. on Microelectronics*, (Rabat, Maroc), 29-31 Oct. 2001.
- F. Monteiro, A. Dandache, A. M'sir et B. Lepley, « A fast CRC implementation on FPGA using a pipelined architecture for the polynomial division », *8th Int. Conf. on Electronics, Circuit and systems*, (St Julian, Malte), Sept. 2001.

# RÉSUMÉ

En télécommunications, l'intégration des services et la diversité des données échangées (voix, vidéo, cellules ATM, etc.) exigent des systèmes de plus en plus rapides pour traiter des volumes d'information en augmentation constante, tout en conservant des délais d'attente les moins contraignants possibles pour l'utilisateur. ces deux contraintes liées au traitement de l'information s'ajoute la nécessité de protéger les données émises dans des canaux soumis à des bruits perturbateurs. De la qualité du service (QoS) rendu en termes de communication, quelles que soient les conditions de transmission, se dégagent deux mots clés : fiabilité et rapidité. Le travail de cette thèse fait partie d'un projet global mené dans le laboratoire LICM concernant la conception architecturale d'une chaîne de transmission à haut débit et à faible coût. Ce travail se situe dans le cadre d'un contrat CIFRE entre le laboratoire LICM et la société 2MG Communication. L'objectif est de concevoir une architecture rapide pour un système de codage de faible coût et à haut débit permettant une protection optimale des données.

Une étude approfondie sur les stratégies de codage canal dans le domaine des télécommunications a été effectuée. Il en ressort que les stratégies FEC (*Forward Error Correction*) offrent les meilleurs compromis débit/protection. L'emploi de codes convolutifs est l'une des approches FEC les plus répandues : la possibilité de les utiliser, soit comme entités indépendantes, soit dans des structures concaténées avec d'autres codes, soit encore comme éléments constituants de turbo-codes (techniques de codage offrant actuellement les meilleurs niveaux de protection), les rend très attractifs.

A partir de ce constat, quatre types d'architecture de codeurs convolutifs ainsi qu'une architecture de turbo-codeur ont été développés au niveau RTL et validés sur FPGA. Les résultats obtenus sont compris, pour le codeur convolutif haut débit, entre 1 et 7.45 Gbits/s pour des degrés de parallélisme 8 et 32, respectivement. Dans le cas du turbo-code, l'implantation d'un turbo-codeur à deux niveaux avec un rendement $1/3$ a permis d'atteindre des débits compris entre 1 et 2 Gbits/s et entre 3 et 5 Gbits/s pour des degrés de parallélisme de 8 et 32, respectivement. L'étude a été complétée par le développement d'un décodeur de syndrome pour codes convolutifs, exploitant l'architecture parallèle du codeur haut débit, qui a permis d'atteindre des débits de 4.78 Gbits/s pour un rendement de $1/2$.

**Mots clés :** transmission haut débit, codage canal, codes convolutifs, turbo-codes, architectures parallèles, modélisation RTL.

# ABSTRACT

In the telecommunication area, integrating services, exchanging ever increasing volumes of heterogeneous data and preserving acceptable delays to end users, are requisites setting the need for faster and faster systems. Moreover, data must be protected against modifications due to the noise affecting the transmission channels. From the Quality of Service (QoS) of the communication, whatever the transmition conditions are, two keywords can be highlighted: reliability and throughput. The work presented here is a part of a larger project realized within the LICM laboratory concerning the architectural conception of a high-throughput and low-cost data transmission system. It has been carried out as a collaborative work between the LICM laboratory and the 2MG Communication company. The goal is to design a fast architecture for a low-cost and high-speed encoding and decoding system allowing for optimal data protection.

A thorough study of channel coding strategies in the telecommunication domain has been done. FEC strategies (Forward Error Correction) emerged as those providing the best troughput/protection trade-off. Within FEC, convolutional codes are among the most employed: the possibility to use them, either as independent codes, or in concatenated structures with other codes, or even as a constituent parts of turbo-codes (offering presently the highest levels of data protection) makes them very attractive, which motivates this research work.

Hence, four architectural types of convolutional encoders and one turbo-encoder architecture have been designed in VHDL at the RTL level and validated on FPGAs. The results obtained for the convolutional encoders are within 1 and 7.45 Gbits/s with parallelism levels of 8 and 32, respectively. Concerning the turbo-code, a $1/3$-rate split-level parallel turbo-encoder has been implemented, allowing rates within 1 and 2 Gbits/s and within 3 and 5 Gbits/s to be achieved with parallelism levels of 8 and 32, respectively. The study was completed with the implementation of a syndrome decoder based on the parallel architecture of the high-speed convolutional encoder. Data rates up to 4.78 Gbits/s have been achieved with a $1/2$-rate decoder.

**Key words:** high speed transmission, channel encoding, convolutional codes, turbo-codes, parallel architectures, RTL modeling.