ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE LA
MAÎTRISE EN GÉNIE ÉLÉCTRIQUE
M.Ing.

PAR
KHALID BENSADEK

DÉVELOPPEMENT D'UN MODÈLE VHDL SYNTHÉTISABLE D'UN DÉCODEUR DE VITERBI

MONTRÉAL, LE 21 OCTOBRE 2004

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

M. Jean Belzile, directeur de mémoire
Département de génie électrique à l'École de technologie supérieure

M. Claude Thibeault, codirecteur
Département de génie électrique à l'École de technologie supérieure

M. François Gagnon, président du jury
Département de génie électrique à l'École de technologie supérieure

M. Christian Cardinal, membre du jury externe
Département de génie électrique à l'École Polytechnique de Montréal

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 6 OCTOBRE 2004

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

DÉVELOPPEMENT D'UN MODÈLE VHDL SYNTHÉTISABLE D'UN DÉCODEUR DE VITERBI

Bensadek Khalid

SOMMAIRE

Le codage convolutionnel avec le décodage par l'algorithme de Viterbi est couramment utilisé dans les systèmes de communication numérique actuels pour améliorer leurs performances. L'objectif de ce mémoire de maîtrise est de concevoir et de mettre en œuvre un modèle VHDL synthétisable, ("core"), du décodeur de Viterbi ciblant la technologie FPGA. La disponibilité d'un modèle synthétisable donne plus de flexibilité quant à la mise en œuvre des systèmes. En plus, un modèle paramétrable facilite le prototypage du décodeur de Viterbi selon différentes spécifications et permet l'exploitation des performances des différentes implémentations afin de choisir celle qui se trouve à être la plus convenable pour un système de communication donné.

L'évolution récente de la technologie VLSI, notamment pour les circuits FPGA, a permis de faire des réalisations performantes du décodeur de Viterbi sur ce type de circuits. L'architecture du décodeur de Viterbi présentée ici se base sur l'utilisation du treillis radix-4 qui permet de faire deux itérations du treillis ordinaire, radix-2, en une seule. L'utilisation de ce treillis a l'avantage de doubler le débit du décodeur tout en conservant un rapport d'efficacité en surface de 1.

Le modèle VHDL du décodeur de Viterbi, conçu dans le cadre de ce mémoire de maîtrise, décode les codes convolutionnels dont les paramètres sont définis par l'utilisateur. Parmi ces paramètres, il y a la mémoire du code, le nombre de symboles d'entrée, le nombre de niveaux de quantification douce par symbole d'entrée, la largeur des mots des métriques de branche et des métriques d'état et la longueur du "Trace Back". Le calcul des métriques de branche est implémenté sous forme de table de conversion configurable afin de pouvoir adapter le décodeur de Viterbi au type de modulation utilisée et améliorer ainsi les performances d'erreur. Les polynômes générateurs du code sont aussi configurables permettant ainsi de compléter la flexibilité du décodeur. Par exemple, un décodeur compilé pour une mémoire de code donnée, m, peut réaliser le décodage des codes convolutionnels de mémoire de code m_c tel que $m_c \leq m$ avec les polynômes générateurs appropriés. Ce décodeur peut réaliser le décodage pour deux groupes de taux de codage pour la même compilation : *1/S* et *2/S*, où S est le nombre de symboles d'entrée.

DESIGN OF VHDL SYNTHESIZABLE VITERBI DECODER CORE

Khalid Bensadek

ABSTRACT

Convolutional coding with Viterbi decoding is often used in recent digital communication systems to improve their performances. The objective of this master's thesis is to design and implement VHDL synthesizable Viterbi decoder core targeting FPGA technology. The availability of synthesizable core provides more flexibility to the system's implementation. The genericity of the design facilitates the prototyping of the decoder with different specifics, and also facilitates the exploration of the performances of different implementations in order to choose the most suitable one for a specific communication system.

The recent VLSI technology evolution, particularly for FPGA integrated circuits, makes it possible to realize high performance Viterbi decoder implementations on this type of circuits. The architecture of Viterbi decoder presented here is based on the use of the trellis radix-4 which process two radix-2 trellis stages per iteration. Such a trellis has the advantage of doubling the data rate while maintaining an area efficiency ratio of 1.

The VHDL synthesizable Viterbi decoder core, designed during this thesis, decodes convolutional codes whose parameters are user defined. Among these parameters we note the code memory, the number of input symbols, the number of soft quantification levels per input symbol, branch metrics and state metrics words widths and the length of the Trace Back. The branch metrics computation is implemented using a configurable look up table in order to be able to adapt the Viterbi decoder to the type of used modulation and thus to improve its bit error rate. The code generating polynomials are also configurable thus adding more flexibility to the decoder. For example, a decoder compiled for a given code memory m can carry out the decoding of convolutional codes of memory m_c that is $m_c \leq m$ with the appropriate generating polynomials. This decoder can also carries out decoding for two groups of coding rate for the same compilation: *1/S* and *2/S*, where S is the number of input symbols.

REMERCIEMENTS

Je remercie spécialement mon directeur de mémoire M. Jean Belzile ainsi que mon codirecteur M. Claude Thibeault de leurs précieux conseils et du temps qu'ils m'ont accordé.

Je remercie également tous mes camarades du laboratoire LACIME pour leur appui technique, logistique et pour les échanges fructueux de connaissances.

Je remercie ma femme pour son soutien et son appui constant du début jusqu'à la fin et dédie ce mémoire à notre couple de jumeau-jumelle que nous attendons très bientôt.

TABLE DES MATIÈRES

Page

SOMMAIRE i
ABSTRACT ii
REMERCIEMENTS iii
TABLE DES MATIÈRES iv
LISTE DES TABLEAUX vii
LISTE DES FIGURES viii
LISTE DES ABRÉVIATIONS ET SIGLES xi
CHAPITRE 1 INTRODUCTION 1
1.1 Objectifs 3
1.2 Contributions 3
1.3 Contenu du rapport 4
CHAPITRE 2 CODES CONVOLUTIONNELS ET ALGORITHME DE VITERBI 6
2.1 Codeurs convolutionnels 6
2.1.1 Représentation en arbre 8
2.1.2 Représentation en diagramme d'état 9
2.1.3 Représentation en diagramme de treillis 10
2.2 Performances des codes convolutionnels 11
2.3 Quantification du signal reçu 15
2.4 Codes catastrophiques 15
2.5 L'algorithme de Viterbi 16
2.6 Implémentation du décodeur de Viterbi 18
2.6.1 Unité de calcul des métriques de branche BMU 19
2.6.2 Unité du Add-Compare-Select ACSU 19
2.6.3 Unité de la gestion de la mémoire des chemins survivants SMMU 21
2.7 Conclusion 23

CHAPITRE 3 ARCHITECTURE PROPOSÉE 24
3.1 Choix de la technique radix-4 24
3.1.1 Forme générale du radix-4 26
3.2 Module de calcul des métriques de branche BMU 27
3.2.1 Taux de 1/S 27
3.2.2 Taux de 2/S 29
3.3 Module de l'Addition-Comparaison-Sélection ACSU 30
3.3.1 Module ACS à quatre chemins 30
3.3.2 Paire d'addition-comparaison 31
3.3.3 Arithmétique modulo pour le calcul des métriques des chemins 32
3.3.4 Comparaison modifiée 33
3.3.5 Gain en vitesse 36
3.3.6 Flexibilité du module des ACS 36
3.4 Unité du Trace Back TBU 37
3.4.1 Organisation de la mémoire 37
3.4.2 Flux des RAM de la mémoire du TB 38
3.4.3 Principe de décodage 41
3.5 Unité LIFO 44
3.6 Diagramme bloc général du décodeur de Viterbi 45
3.7 Conclusion 48
CHAPITRE 4 IMPL ÉMENTATION, TESTS ET RÉSULTATS 50
4.1 Architecture du FPGA de Xillinx 50
4.1.1 VIRTEX-II de Xillinx 51
4.2 Cycles de design 53
4.3 Modularité et hiérarchie du code VHDL 56
4.4 Optimisation du design 56
4.4.1 Unité BMU et TBU 57
4.4.2 Unité ACSU 57

4.4.3 Unité LIFO 58
4.4.4 Décodeur entier. 60
4.4.4.1 Amélioration de la vitesse du décodeur 63
4.5 Test et résultat 66
4.5.1 Environnement de test 66
4.5.2 Procédures de test 68
4.5.3 Résultats de la simulation du code VHDL 69
4.5.4 Test du design sur circuit FPGA 70
4.5.5 Résultats du test du design sur circuit FPGA 72
4.6 Comparaison avec un modèle existant 73
4.7 Conclusion 76
CHAPITRE 5 CONCLUSION ET RECOMMANDATION 78
5.1 Travaux futurs 79
ANNEXES
1 : Génération des vecteurs de contrôle 80
2 : Résultats de synthèse du décodeur de Viterbi et de ces différentes unités 84
BIBLIOGRAPHIE 111

LISTE DES TABLEAUX

Page

Tableau I Mesure de la rapidité et de la complexité de radix-2^x.....26

Tableau II Exemple de formation d'indice des métriques de branche radix-4 pour R = 1/2.....29

Tableau III Nombre de bits, $\Gamma_{bits,}$ en fonction de N pour R = 1/2.....33

Tableau IV Contrôle de la mémoire du TB.....43

Tableau V Résumé des résultats de la synthèse du code VHDL de la cellule ACS à quatre chemins pour les deux types de description : flot de données et comportemental.....58

Tableau VI Résumé des résultats de la synthèse du code VHDL de l'unité LIFO utilisant des registres et utilisant des blocs RAM.....60

Tableau VII Résumé des résultats de la synthèse des modules du décodeur, (m = 6, R = 1/2 et L = 64) 62

Tableau VIII Résumé des résultats de la synthèse du décodeur entier sans améliorations de la vitesse, avec pipeline et avec balancement des registres pour les opérations d'addition-comparaison-sélection, (m = 6, R = 1/2 et L = 64).....65

Tableau IX Résumé du résultat de placement et routage du décodeur de Viterbi avec balancement des registres,(m = 6, R = 1/2 et L = 64).....66

Tableau X Comparaison des caractéristiques de notre modèle du décodeur de Viterbi avec ceux du modèle de la compagnie Xilinx.....74

Tableau XI Comparaison de la complexité, de la fréquence et du débit de notre modèle (version non flexible) avec celui de Xilinx.....76

LISTE DES FIGURES

Page

Figure 1 Système de communication utilisant un encodeur convolutionnel avec un décodeur de Viterbi pour la correction d'erreurs 2

Figure 2 Codeur convolutionnel de taux R = 1/2, de mémoire m = 2 et ayant les polynômes générateurs (7_8,5_8) 7

Figure 3 Représentation en arbre de l'action du codeur de la figure 2 9

Figure 4 Représentation en machine à états finis de l'action du codeur de la figure 2 10

Figure 5 Diagramme en treillis du codeur de la figure 2 11

Figure 6 Représentation d'un canal binaire symétrique (BSC) 12

Figure 7 Performance d'erreur pour un code de taux R = 1/2 avec le décodage par l'algorithme de Viterbi à entrée non quantifiée 14

Figure 8 Illustration de l'opération ACS par état pour un taux R=1/S 17

Figure 9 Diagramme bloc du décodeur de Viterbi 18

Figure 10 Réalisation d'une opération ACS pour un code de taux R = 1/S 20

Figure 11 Division du treillis en papillons pour un code de taux R = 1/2 et de m = 2 20

Figure 12 Cellule de base (papillon) du module ACSU pour un code de taux R = 1/S 21

Figure 13 Schéma bloc du trace back pour un code de taux W/S et de nombre d'état N 23

Figure 14 Décomposition du treillis de 8 états, R = 1/2 en sous treillis radix-4 25

Figure 15 Forme générale de la décomposition en treillis radix-4 27

Figure 16 Module BMU, R = W/S où W = 1,2 et S = 1,2,3,... 30

Figure 17 Bloc ACS à 4-chemins représentant un état du treillis radix-4 R = W/S où W = 1,2 et S = 1,2,3,... 31

Figure 18 Diagramme bloc illustrant une paire d'addition-comparaition simultanées à 8-bits 32
Figure 19 Diagramme bloc du module ACS à 4-chemins 35
Figure 20 Cellule ACS à 4-chemins, R = W/S où W = 1,2 et S = 1,2,3,... bmr4 = 2^{2s} (ou 2^{s}) et sl = 2S (ou S) si R = 1/S (ou 2/S) 37
Figure 21 Organisation de la mémoire des décisions en tampon cyclique 38
Figure 22 Flot de la mémoire du TB à quatre RAM 40
Figure 23 Diagramme bloc du Trace Back 42
Figure 24 Bloc Mux_logic 43
Figure 25 Schéma fonctionnel de la pile LIFO bidirectionnel 45
Figure 26 Unité LIFO 45

Figure 27 Diagramme bloc du décodeur de Viterbi, R = W où W = 1,2 et S = 1,2,3,...bmr2=2^{S}, bmr4 = 2^{2s} (ou 2^{s}) et sl = 2S (ou S) si R = 1/S (ou 2/S) 47
Figure 28 Chronogramme des phases de fonctionnement du décodeur, R = 1/S 48
Figure 29 Vue générale de l'architecture d'un circuit FPGA Virtex-II 51
Figure 30 Configuration d'une tranche d'un circuit FPGA Virtex-II 52
Figure 31 Organigramme du cycle du design 55
Figure 32 Hiérarchie du design 56
Figure 33 Diagramme bloc de l'unité LIFO utilisant des blocs RAM 60
Figure 34 Amélioration de la vitesse pour l'opération d'addition-comparaisson-sélection du module ACSU 65
Figure 35 Illustration de l'environnement de test du décodeur 67
Figure 36 Seuils et espacement de la quantification douce sur huit niveaux 67
Figure 37 Illustration des procédures de test et prélèvement des résultats 68
Figure 38 Courbe de performance d'erreur du décodeur de Viterbi pour le code de m = 6, R = 1/2, polynômes 171_8 et 133_8 et de L = 64 70
Figure 39 Schéma bloc illustrant les éléments additionnels nécessaires au test

du design réel implémenté sur FPGA 71

Figure 40 Courbes de performance d'erreur mesurées par simulation et par test réel sur circuit FPGA. m=2, R=1/2, polynômes 7_8 et 5_8 et de L=16 72

LISTE DES ABRÉVIATIONS ET DES SIGLES

FPGA	Field Programmable Gate Array
ASIC	Application Specific Integrated Circuit
CPLD	Complex Programmable Array Logic
CMOS	Complementary Metal Oxide Semiconductor
VLSI	Very Large Scale Integration
VHSIC	Very High Speed Integrated Circuit
VHDL	VHSIC High Description Language
3G	Third Generation
WLAN	Wireless Local Area Network
ROM	Read Only Memory
RAM	Random Access Memory
PROM	Programmable Read Only Memory
LACIME	Laboratoire de Communication et d'Intégration de la Micro-Électronique
SNR	Rapport signal à bruit
BER	Bit Error Rate
SOVA	Soft Output Viterbi Algorithm
AWGN	Additive White Gaussian Noise
TCM	Trellis Coded Modulation
BPSK	Binary Phase Shift Keying
BSC	Binary Symmetric Channel
FEC	Forward Error Correction
MSB	Most Significant Bit
LSB	Least Significant bit
LUT	Look Up Table
CLB	Configurable Logic Block
IOB	Input Output Block

E_b	Energie par bit, J/bit
N_o	Densité spectrale du bruit
p	Probabilité de transition
P_b	Probabilité par bit
Q(x)	Fonction d'erreur
d_{free}	Distance libre
i	Itération du temps
K	Longueur de contrainte
m	mémoire de code
N	Nombre d'état
R	Taux de code
L	Longueur du trace back
W	Nombre de bits d'entrée de l'encodeur convolutionnel
S	Nombre de symboles d'entrée du décodeur de Viterbi
Qt	Nombre de niveaux de quantification du signal d'entrée
Γ	Métrique de chemin
λ	Métrique de branche

CHAPITRE 1

INTRODUCTION

Le principe du codage convolutionnel a été introduit en 1955 par Elias comme alternative aux codes blocs [1]. Les codes convolutionnels ajoutent une certaine redondance aux bits de la séquence d'information à transmettre moyennant une opération logique (OU exclusif). L'ajout de cette redondance permet au décodeur, à la réception, de corriger d'éventuelles erreurs lors de la transmission dans le canal. Différents algorithmes de décodages ont été développés. Le plus connu est sans contredit l'algorithme de Viterbi, publié en 1967 [2] et qui a la particularité d'être optimal. Depuis, les codes convolutionnels avec l'algorithme de Viterbi sont de plus en plus utilisés dans les systèmes de communication numérique comme code correcteur d'erreur de la catégorie FEC ("Forward Error Correction"). L'utilisation d'un tel code correcteur d'erreur introduit un gain, par rapport aux systèmes non codés, qui est très attrayant spécialement pour les systèmes de transmission sans fil. Ce gain se traduit par une réduction de la puissance de transmission, une réduction des dimensions des antennes ou par une augmentation du taux de transmission pour la même probabilité d'erreur. La figure 1.1 montre un système de communication utilisant un encodeur convolutionnel avec un décodeur de Viterbi pour la correction d'erreurs introduites par le canal de transmission.

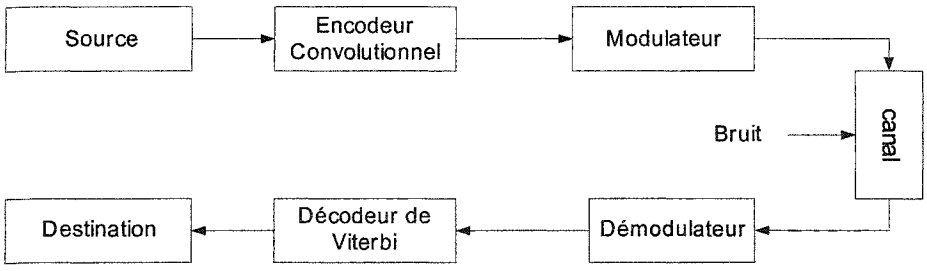


Figure 1 Système de communication utilisant un encodeur convolutionnel avec un décodeur de Viterbi pour la correction d'erreurs

Les applications en télécommunication du décodeur de Viterbi sont nombreuses et différentes. Ceci engendre une nécessité grandissante de contrôler les paramètres du décodeur pour l'adapter à des circonstances particulières. D'autre part, il n'y a pas de règles claires indiquant les valeurs optimales de ces paramètres. Il y a seulement quelques règles conseillant les choix de ces paramètres. La solution optimale qui permet d'explorer ces paramètres est de les rendre génériques. D'un autre côté, il y a un besoin grandissant de dispositifs flexibles pour supporter des standards multiples de communication. Chaque standard requière une configuration différente pour le code convolutionnel utilisé, ce qui change les exigences pour le décodeur de Viterbi [3].

Les performances d'un décodeur utilisant l'algorithme de Viterbi comme technique de correction d'erreurs introduites par le canal de transmission augmentent avec le pouvoir de correction du code. L'augmentation de celle-ci entraîne une augmentation de la complexité du décodeur. C'est l'inconvenant majeur du décodeur de Viterbi. Le développement dans les années récentes de la technologie VLSI (intégration à grande échelle) notamment dans le domaine de la fabrication des circuits numériques complexes à faible coût a permis de réaliser des implémentations très intéressantes du décodeur de Viterbi en jouant sur le compromis performance-complexité.

1.1 Objectifs

L'objectif du présent mémoire consiste à développer un modèle synthétisable, ("core"), en langage de description de matériel, VHDL, du décodeur de Viterbi. Nous voulons réaliser un modèle du décodeur de Viterbi qui soit paramétrable, flexible et reprogrammable pour faciliter le prototypage du décodeur de Viterbi. Nous voulons que la plupart des paramètres du décodeur de Viterbi soient définissables par l'utilisateur à la compilation. Les autres paramètres seront configurables après compilation et implémentation pour donner au décodeur le maximum de flexibilité. Ces mêmes paramètres peuvent, alors, être reprogrammés dynamiquement au besoin. Cette flexibilité permet d'accélérer le prototypage du décodeur de Viterbi. Nous pourrons donc passer d'un décodeur de Viterbi avec des paramètres donnés à un autre décodeur de Viterbi avec d'autres paramètres sans avoir besoin de recompiler notre modèle VHDL.

Notre modèle VHDL du décodeur de Viterbi cible la technologie FPGA. Ceci le rend plus performant que les autres modèles décrits de façon indépendante de la technologie utilisée pour leur implémentation. L'utilisation de la technologie FPGA est justifiée par sa programmabilité qui facilite le prototypage rapide.

1.2 Contributions

Dans les années récentes, l'intérêt porté à la conception et l'utilisation des modèles synthétisables ("core") ne cesse d'augmenter. Les modèles flexibles ou reprogrammables sont encore plus intéressants pour les communications multistandards comme c'est le cas pour les systèmes de communication sans fil de la troisième génération (3G) et pour les réseaux informatiques sans fils (WLAN).

Les contributions apportées par le travail présenté dans ce mémoire de maîtrise, correspondent à la conception, la vérification et la mise en œuvre d'un modèle VHDL synthétisable du décodeur de Viterbi de hautes performances qui est paramétrable, flexible et reconfigurable.

1.3 Contenu du rapport

Dans la première partie du deuxième chapitre, nous présentons tout d'abord le codeur convolutionnel, ses paramètres ainsi que les trois représentations graphiques de son fonctionnement. Par la suite, nous faisons un bref rappel des performances des codes convolutionnels (distance libre, probabilité d'erreur, gain de codage et quantification du signal reçu). Dans la deuxième partie de ce chapitre, nous présentons l'algorithme de Viterbi comme un processus de décodage à maximum vraisemblance des codes convolutionnels. Ensuite, nous présentons l'implémentation classique du décodeur de Viterbi sous forme de trois unités fonctionnelles.

Le troisième chapitre traite de l'architecture proposée pour la réalisation du décodeur de Viterbi sous forme de modèle VHDL paramétrable. Dans un premier temps, nous introduisons le principe de la décomposition du treillis ordinaire, radix-2, en treillis radix-X. Puis nous justifions notre choix du treillis radix-4 sur lequel se base l'architecture du décodeur proposé pour doubler le débit général de celui-ci. Par la suite, nous présentons l'implémentation détaillée du décodeur sous forme de quatre unités distinctes. Des discussions de complexité-performance sont également abordées.

Le quatrième chapitre se divise en deux grandes parties. La première partie présente les différentes étapes d'implémentation et d'optimisation du code VHDL du décodeur de Viterbi pour cibler un FPGA Virtex II de Xilinx. Au début de cette partie, nous présentons une brève description de l'architecture interne d'un FPGA Virtex II. Puis

nous abordons les différents cycles de conception ciblant un circuit FPGA. Après, nous présentons l'étape d'optimisation pour les différents modules du design. Dans cette section, nous discutons les styles d'écriture du code VHDL à prendre en considération pour mieux cibler un circuit FPGA. La deuxième partie présente l'environnement et les résultats de test du décodeur de Viterbi sous forme de code VHDL simulé et sous forme d'implémentation matérielle sur un circuit FPGA Virtex II.

CHAPITRE 2

CODES CONVOLUTIONNELS ET ALGORITHME DE VITERBI

L'objectif de ce chapitre consiste à revoir le principe de codage convolutionnel et du décodage par l'algorithme de Viterbi. Dans un premier temps, à la section 1, nous introduirons le codeur et les principaux paramètres qui le caractérisent. Les trois représentations graphiques du décodeur suivront à la fin de cette section. Dans la section 2, nous présenterons une brève étude de la probabilité d'erreur comme mesure principale de performance d'un code. La notion de gain de codage sera aussi introduite à la fin de cette section. Suivra ensuite, à la section 3, une présentation de l'effet de la quantification du signal reçu sur les performances du décodeur. Dans la section 4, nous définirons les codes catastrophiques, par la suite, nous donnerons les conditions nécessaires et suffisantes pour qu'un code soit catastrophique. L'algorithme de Viterbi sera revu à la section 5. Finalement, la section 6 sera dédiée à l'implémentation conventionnelle de cet algorithme sous forme de trois modules à savoir le module BMU, le module ACSU et le module SMMU.

2.1 Codeurs convolutionnels

Les techniques de codage de contrôle d'erreur jouent un grand rôle dans les systèmes de communications numériques. Les codes convolutionnels ou convolutifs constituent une grande famille de codes correcteurs d'erreurs.

Le principe du codage convolutionnel consiste en l'association du bit d'entrée à transmettre à plusieurs bits précédemment transmis par une opération logique (généralement des OU exclusifs), de façon à retrouver sa valeur en cas d'incident de transmission. Ainsi, cette façon d'introduire de la redondance dans l'information à transmettre donne au code sa capacité à détecter et à corriger les erreurs.

Un codeur convolutionnel est caractérisé par deux paramètres et par les connexions entre les registres à décalage et les additionneurs modulo 2. Les deux paramètres sont la mémoire du code et le taux de codage. La mémoire du code, m, dénote le nombre de cases mémoire minimale pour implémenter les W registres à décalage. Le taux de codage, W/S, est le rapport du nombre de bits d'entrée W sur le nombre de symboles de sortie S par cycle du codeur. Il dénote aussi que le codeur contient S additionneurs modulo 2. Les connexions entre les registres à décalage et les additionneurs sont définies par les S polynômes générateurs g_i du codeur convolutionnel. La figure 2 présente l'exemple d'un codeur convolutionnel de taux R = 1/2, de mémoire m = 2 et avec les polynômes générateurs $(7_8,5_8)$. Les nombres représentés en notation octale 7_8 et 5_8, qui sont en binaire 111 et 101, représentent les connections entre le registre à décalage et les additionneurs modulo 2. Une autre représentation mathématique de ces polynômes est : $g_0 = x^2 + x + 1$ et $g_1 = x^2 + 1$. Ce codeur fera office d'exemple pour la suite de cette section afin de mieux illustrer les différentes représentations de l'action du codeur convolutionnel.

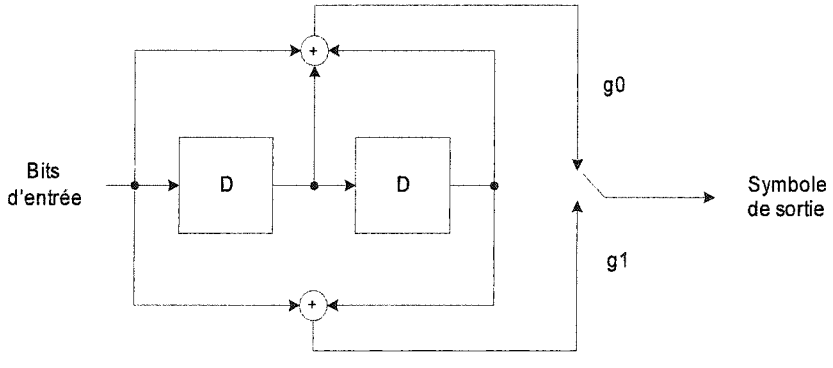


Figure 2 Codeur convolutionnel de taux R = 1/2, de mémoire m = 2 et ayant les polynômes générateurs $(7_8,5_8)$

2.1.1 Représentation en arbre

Une façon pratique de décrire la relation entre les séquences d'entrée et de sortie d'un codeur est la représentation en arbre. Chaque branche représente une des possibilités de W bits d'entrée. De chaque nœud de l'arbre émergent 2^W branches. Chaque branche contient une étiquette représentant les S symboles codés de sortie correspondants aux W bits d'entrée du codeur. La figure 3 présente l'arbre du codeur de la figure 2. Un bit d'entrée '0' correspond à la branche du haut et un bit d'entrée '1' correspond à la branche du bas. Donc, une séquence donnée de bits d'information passe par un chemin unique dans l'arbre. Les symboles, ou mots de code, lus au long de ce chemin forment la séquence correspondante de sortie du codeur. Si on prend pour conditions l'état de départ '00' et la séquence d'entrée 1101 alors, la séquence de sortie sera 11 10 10 00 (la ligne en gras de la figure 3).

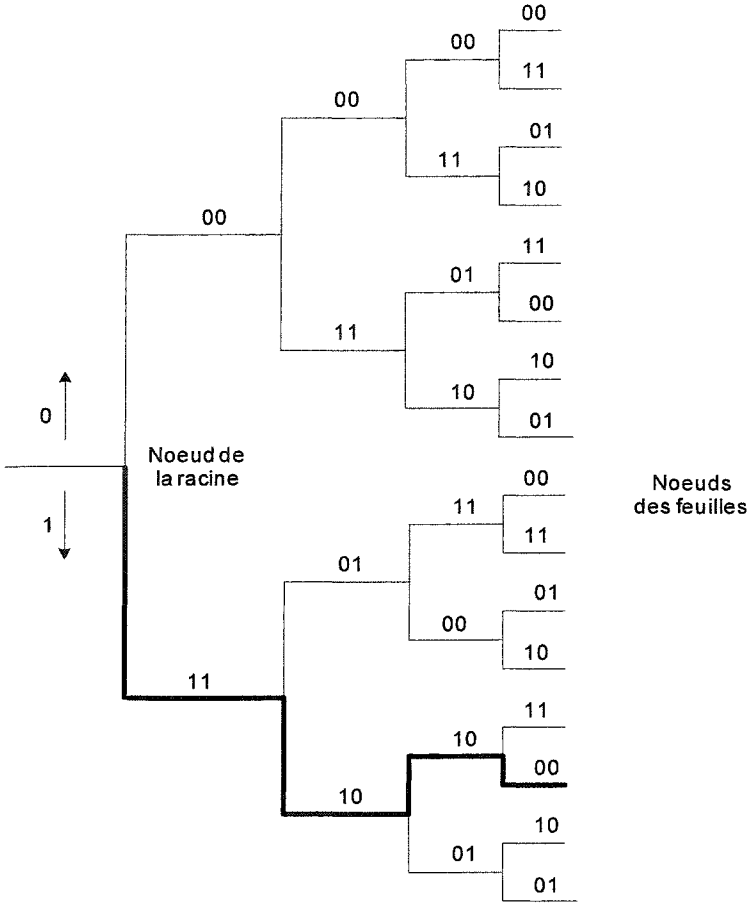


Figure 3 Représentation en arbre de l'action du codeur de la figure 2

2.1.2 Représentation en diagramme d'état

Le codeur est une machine linéaire à états finis qui peut être facilement décrite par son diagramme d'état. Cela est démontré à la figure 4 pour le codeur de la figure 2. Les nœuds ou les états représentent le contenu du registre à décalage. Les branches, qui forment les transitions entre les états, indiquent la valeur du bit d'entrée. Si la branche est une ligne continue, elle indique que le bit d'entrée est un '0'. Si elle est une ligne pointillée, elle indique que le bit d'entrée est un '1'. Le mot de deux bits qui étiquette la

branche de transition représente les deux symboles codés correspondants de sortie du codeur.

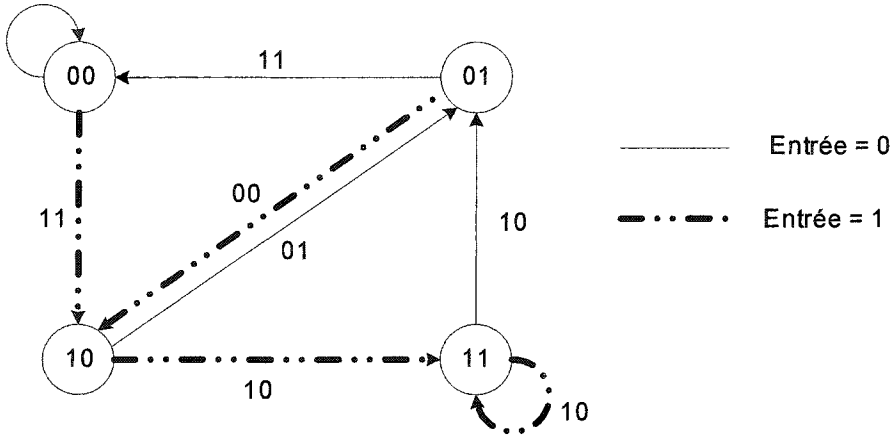


Figure 4 Représentation en machine à états finis de l'action du codeur de la figure 2

2.1.3 Représentation en diagramme de treillis

La réplication infinie du diagramme d'état résulte en une structure appelée le treillis. C'est une version indexée par le temps du diagramme d'état. Chaque nœud correspond à un des 2^m états, à un temps donné. Chaque branche parmi les 2^W branches par état correspond à une transition entre deux états. Elle est associée aux W bits d'entrée et S symboles correspondants de sortie du codeur. N'importe quelle séquence de mots de code correspond à un chemin unique constitué de branches successives dans le diagramme de treillis. La figure 5 présente le treillis du codeur de la figure 2.

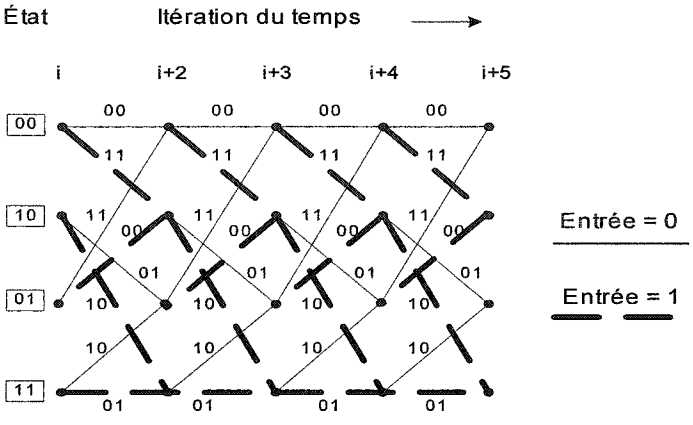


Figure 5 Diagramme en treillis du codeur de la figure 2

2.2 Performances des codes convolutionnels

Les performances d'un code se mesurent principalement par la probabilité d'erreur. Cette dernière dépend de la distance libre, d_{free}. C'est la distance de Hamming minimale entre toutes les paires de mots de code. La distance de Hamming entre deux mots de code est définie comme le nombre de positions dans les quels ils diffèrent.
Dans cette section, nous faisons un survol des formules de calcul de la probabilité d'erreur par bit pour un système utilisant un canal binaire symétrique avec et sans codage. Le lecteur désirant savoir plus sur le calcul de ces formules est prié de se référer aux ouvrages suivants : [2], [4-6].

Le canal binaire symétrique, BSC ("Binary Symmetric Channel"), est un canal dont l'entrée est binaire et dont la sortie est aussi binaire ou quantifiée sur deux niveaux. On l'appelle aussi un canal à décision dure. La probabilité de transition dans un tel canal est p. La figure 6 présente le modèle d'un canal binaire symétrique. Ce modèle de canal est souvent utilisé pour mieux illustrer les performances d'erreur.

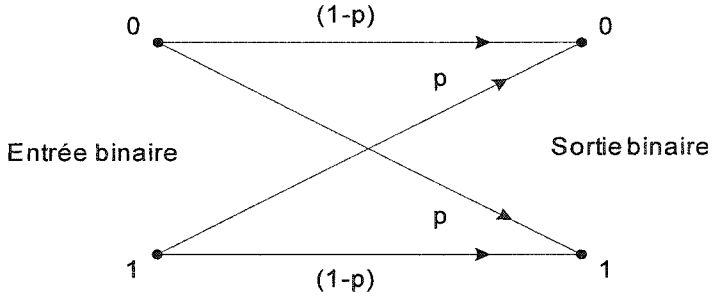


Figure 6 Représentation d'un canal binaire symétrique (BSC)

La valeur de la probabilité de transition, p, pour la modulation BPSK cohérente avec un bruit blanc gaussien additif (AWGN) est donnée par la formule suivante [4] :

$$p = Q\left(\sqrt{\frac{2RE_b}{N_0}}\right) \tag{2.1}$$

où E_b/N_0 est une mesure du rapport de l'énergie d'un bit sur la densité spectrale du bruit, R est le taux de codage et Q(x) est la fonction d'erreur d'une distribution gaussienne.

Pour les systèmes non codés, R = 1, cette probabilité de transition correspond à la probabilité d'erreur par bit. Pour des grandes valeurs de E_b/N_0 elle peut être approximée de la façon suivante [6]:

$$P_b = Q\left(\sqrt{\frac{2E_b}{N_0}}\right) \approx \frac{1}{2} e^{-\frac{E_b}{N_0}} \tag{2.2}$$

La probabilité d'erreur par bit pour une modulation BPSK cohérente, codée et avec un bruit blanc gaussien additif pour un canal BSC (décision dure) et pour des grandes valeurs de E_b/N_0 peut être approximée par la formule suivante [5] :

$$P_b \approx C_{d_{free}} 2^{d_{free}} e^{\left(-E_b/N_0\right)\left(Rd_{free}/2\right)} \tag{2.3}$$

où C_{dfree} est le nombre de bits en erreur pour les chemins ayant un poids de Hamming égale à d_{free} par rapport à la séquence transmise. En comparant les deux équations, (2.2) et (2.3), et en considérant seulement les exposants des expressions on constate que pour le même rapport E_b/N_0, l'exposant de l'expression avec le codage convolutionnel est d'au moins un facteur de $Rd_{free}/2$ plus grand que l'exposant de l'expression sans codage. C'est donc la borne inférieure du gain de codage. *Le gain de codage* est la différence exprimée en dB dans le rapport E_b/N_0 requis pour une probabilité d'erreur donnée, entre une modulation idéale donnée sans codage et la même modulation avec codage. D'autre part, pour un canal AWGN avec une entrée binaire et une sortie non quantifiée la probabilité d'erreur par bit peut être approximée comme suit [6] :

$$P_b \approx C_{d_{free}} e^{\left(-E_b/N_0\right)\left(Rd_{free}\right)} \tag{2.4}$$

En comparant les expressions (2.2) et (2.4) on trouve la borne supérieure du gain de codage, soit Rd_{fee}. Donc, le gain de codage, en utilisant un canal BSC, est borné comme suit [5] :

$$(Rd_{free})/2 \leq \text{gain de codage} \leq Rd_{free} \tag{2.5}$$

Comme on peut le constater, il y a une marge de 3dB séparant les deux bornes de l'expression (2.5). On remarque aussi que pour un taux de codage donné, R, le gain de codage augmente avec l'augmentation de la distance libre. D'autre part, on trouve dans les ouvrages [4], [5] et [12] des tables qui montrent que la distance libre augmente avec la mémoire du code, m. La figure 7 montre le taux d'erreur par bit en fonction du rapport signal à bruit pour des différentes mémoires de code, m, pour un taux R = 1/2 et un signal d'entrée du décodeur de Viterbi non quantifié.

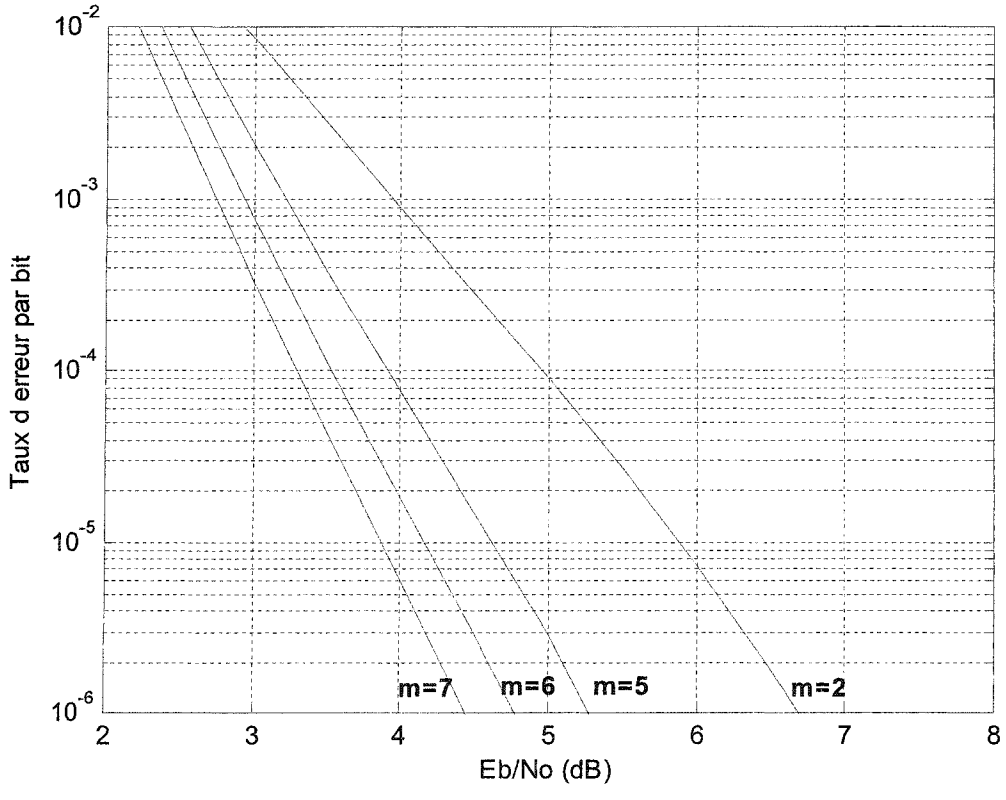


Figure 7 Performance d'erreur pour un code de taux R = 1/2 avec le décodage par l'algorithme de Viterbi à entrée non quantifiée

2.3 Quantification du signal reçu

Théoriquement, un décodeur de Viterbi idéal travaille avec une précision infinie. Dans les systèmes pratiques, on quantifie les symboles reçus sur un bit de précision ou plus pour réduire la complexité du décodeur, sans parler des circuits qui le devancent. Si les symboles reçus sont quantifiés sur un bit (deux niveaux, exemple $< 0V = 1$ et $\geq 0V = 0$), le résultat est appelé *décision dure*. Si les symboles reçus sont quantifiés sur plus d'un bit, le résultat est appelé *décision douce*. La décision dure introduit une dégradation de 2.2 dB par rapport aux systèmes non quantifiés dans lesquels le décodeur opère directement avec le signal reçu [15]. La décision douce sur trois bits (huit niveaux) n'introduit qu'une légère perte de 0.2 dB par rapport à ces même systèmes non quantifiés [5]. Une quantification sur plus de huit niveaux n'apporte qu'une légère amélioration au prix d'une augmentation de complexité non justifiable [5]. D'autre part, un bon espacement entre les niveaux quantifiés est approximativement égal à $\sqrt{N_o/2}$, où N_0 est la densité spectrale du bruit [5]. Cela vient poser une limite maximale sur le nombre de niveaux de quantification.

2.4 Codes catastrophiques

Un code est appelé catastrophique si dans une séquence de bits codés, un nombre fini de symboles en erreur engendre une séquence de bits décodés en erreur qui est arbitrairement longue. En générale, la condition nécessaire et suffisante pour qu'un code de taux R = W/S soit catastrophique est qu'au moins un chemin dans son diagramme d'état forme une boucle fermée avec un poids nul. Massey et Sain [2] ont trouvé une condition nécessaire pour tester si un code de taux 1/S est catastrophique. Il en résulte que les polynômes générateurs doivent avoir un facteur commun pour qu'une telle situation se produise.

2.5 L'algorithme de Viterbi

L'algorithme de Viterbi est dit à maximum vraisemblance. Il explore le treillis en comparant tous les chemins possibles avec la séquence reçue et sélectionne celui qui est le plus vraisemblable.

Soit X la séquence de mots de code transmise sur un canal BSC avec une probabilité de transition p, et Y la séquence correspondante reçue. Supposant que X et Y sont de longueur n et que z est la distance de Hamming entre eux. Puisque le canal est sans mémoire, la fonction log-vraisemblance, logP(Y|X), est donnée par la formule suivante [5] :

$$\log P(Y/X) = \log\left[p^{z}(1-p)^{n-z}\right] \tag{2.6}$$

Si on développe cette l'équation, on obtient :

$$\log P(Y/X) = n\log(1-p) - z\log\left(\frac{1-p}{p}\right)$$

$$= -A - Bz \tag{2.7}$$

où A et B sont des constantes positives car p < 0.5. Ainsi, maximiser la fonction log-vraisemblance revient à minimiser la distance de Hamming. Autrement dit, le décodeur à maximum vraisemblance choisit le chemin U du treillis dont la séquence encodée correspondante $X^{(U)}$ est la plus proche, en distance de Hamming, à la séquence reçue Y.

Dans le diagramme en treillis (figure 5), chacune des branches représente une transition entre deux états. Elle est associée à un poids appelé *métrique de branche*. C'est une mesure de distance entre le mot de code affecté par le bruit et le mot de code correspondant à la transition donnée. Aussi, à chaque état, au temps i, est associée la métrique d'état ou la métrique de chemin. C'est le cumul des métriques de branche tout le long du chemin le plus court menant à cet état. La métrique d'état, au temps i, est calculée récursivement à partir de la métrique d'état de l'itération précédente comme suit :

$$\Gamma_i(b) = \min\left\{\Gamma_{i-1}(a_j) + \lambda_{i-1}(a_j, b)\right\} \tag{2.8}$$

où a_j est l'état précédent de b, $\Gamma_{i-1}(a_j)$ la métrique de l'état a_j à l'instant i-1, et $\lambda_{i-1}(a_j,b)$ la métrique de branche de la transition de l'état a_j à l'état b. Cette expression se traduit comme suit : la métrique de chemin est donnée par l'addition de la métrique d'état avec la métrique de branche. La métrique finale d'état $\Gamma_i(b)$ au temps i est la plus petite métrique de tous les chemins possibles qui entrent dans cet état. L'opération de l'expression (2.8) est appelée "Add-Compare-Select" (ACS). La figure 8 illustre l'exemple de l'opération ACS par état pour un taux de 1/S.

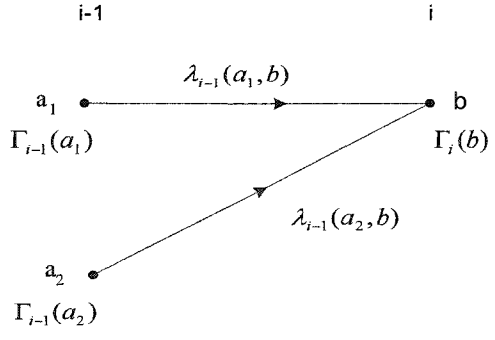


Figure 8 Illustration de l'opération ACS par état pour un taux R = 1/S

La transition la plus probable est celle qui a la plus petite métrique de chemin. Donc à partir de la séquence affectée par le bruit, on peut trouver le chemin à travers le treillis qui a la plus faible métrique de chemin. C'est la séquence transmise la plus probable.

2.6 Implémentation du décodeur de Viterbi

L'implémentation du décodeur de Viterbi (VD) est souvent divisée en trois modules fonctionnels comme présenté à la figure 9 : un module de calcul de métriques de branche (BMU) "Branch Metric Unit", un module d'addition-comparaison-sélection (ACSU) "Add-Compare_Select Unit", et un module de gestion de la mémoire du chemin survivant (SMMU) "Survivor Memory Management Unit". Le module BMU calcule la métrique de branche de chaque transition. Cette métrique de branche est une mesure de la distance entre le code reçu et le code qui correspond à la branche donnée du treillis. Le module ACSU additionne les métriques de branche aux métriques d'état et ensuite compare le résultat pour garder la plus petite métrique d'état. C'est la transition avec la plus grande vraisemblance. Le module SMMU décode la séquence estimée. Il utilise les bits de décision enregistrés par le ACSU pour le chemin le plus court. Ce dernier est le chemin qui a la métrique cumulée d'état la plus faible parmi tous les autres chemins survivants de tous les états possibles. Donc, il représente la séquence la plus vraisemblable.

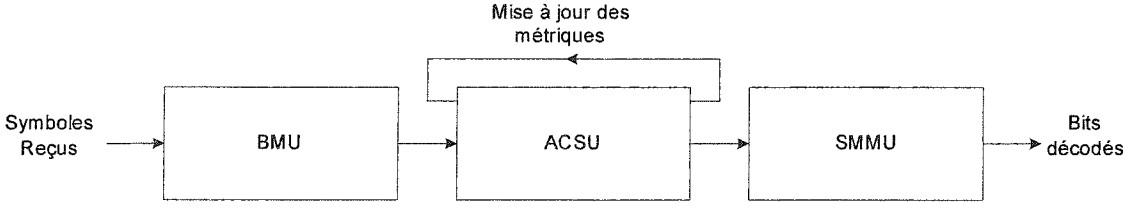


Figure 9 Diagramme bloc du décodeur de Viterbi

2.6.1 Unité de calcul des métriques de branche BMU

Pour chaque itération du treillis, l'unité BMU reçoit S symboles et produit 2^S métriques de branche correspondantes aux 2^S combinaisons possibles de sorties de l'encodeur. Pour le calcul des métriques de branche, plusieurs types de métriques peuvent être utilisés. Les trois métriques les plus utilisées sont la métrique de Hamming pour un canal BSC, la métrique Euclidienne et la métrique de corrélation pour un canal à décision douce. La métrique de Hamming est le nombre de bits différents entre le symbole reçu et un symbole possible. La métrique Euclidienne est la distance géométrique entre ces deux symboles. La métrique de corrélation est le produit de corrélation entre ces deux mêmes symboles.

2.6.2 Unité du Add-Compare-Select ACSU

L'unité ACSU calcule et sélectionne le chemin le plus court des 2^W chemins entrants dans chaque état. Elle fournit aussi W bits de décision indiquant quel chemin a été choisi. Ces bits de décision sont sauvegardés pour ensuite être utilisés par l'unité du SMMU pour le décodage de la séquence estimée. L'opération ACS, décrite dans la section 2.5 par la formule (2.8), peut être implémentée par un ACS à deux chemins, ou ACS radix-2, présenté à la figure 10. $d_i(b)$ est le bit de décision qui indique l'état précédent choisit.

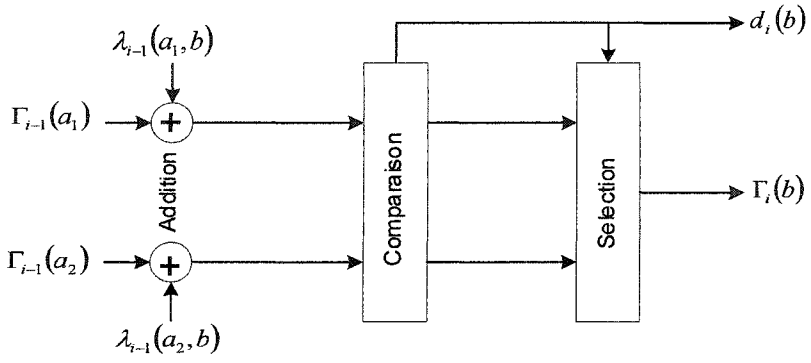


Figure 10 Réalisation d'une opération ACS pour un code de taux R = 1/S

Pour la réalisation d'un module ACSU complet, on divise souvent le treillis en paires d'état qui ont une forme de papillon, (butterfly). La figure 11 illustre une division en papillons pour le cas du treillis de la figure 5.

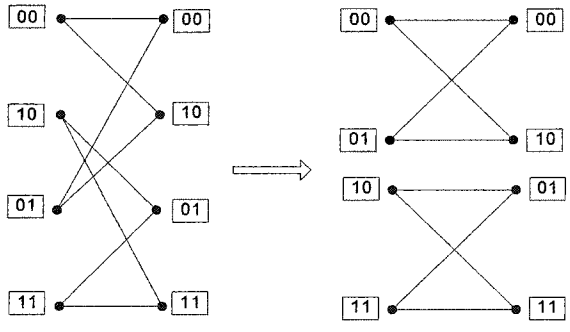


Figure 11 Division du treillis en papillons pour un code de R = 1/2 et de m = 2

Le module ACS en architecture papillon devient donc la cellule de base pour le module ACSU. Cette cellule de base se compose de deux ACS à deux chemins comme présenté à la figure 12.

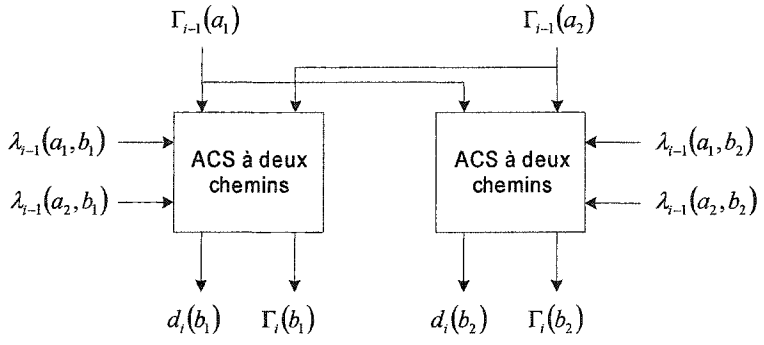


Figure 12 Cellule de base (papillon) du module ACSU pour un code de taux R = 1/S

Pour la réalisation du treillis entier, les deux méthodes les plus utilisées sont la réalisation en série et la réalisation en parallèle. La réalisation en série consiste en l'utilisation d'une unique cellule ACS qui fait les calculs pour chaque paire d'état en papillon à tour de rôle. Cette réalisation permet d'économiser la surface et la consommation en énergie. Cependant, le débit offert est très faible et chute exponentiellement avec la croissance de la mémoire du code. Dans le cas de la réalisation en parallèle, chaque paire d'états est représentée par une cellule ACS de base. Donc le traitement pour tous les états se fait en parallèle offrant ainsi plus de débit. Cependant, le prix à payer est l'augmentation exponentielle du nombre de modules ACSU avec l'augmentation de la mémoire du code.

2.6.3 Unité de la gestion de la mémoire des chemins survivants SMMU

Pour la réalisation de l'unité SMMU, les deux approches les plus utilisées pour le décodage de la séquence estimée sont le "Registre Exchange" (RE) et le "Trace Back" (TB).

L'approche du RE assigne un registre à chaque état. Ces registres s'échangent les bits décodés le long de tous les chemins survivants. Cette approche élimine le besoin du TB

puisque la séquence estimée se trouve dans le dernier registre. Mais, elle n'est pas très efficace en consommation d'énergie car le contenu de tous les registres bascule à la fois à chaque itération donnant lieu à une grande activité de commutation. De plus, la complexité du module SMMU utilisant cette approche augmente exponentiellement avec la mémoire du code, car le diagramme du RE reflète l'image d'une fenêtre de longueur finie du treillis. La longueur de cette fenêtre est la longueur du chemin survivant. Cette méthode est utilisée pour des petites mémoires m et des petites longueurs du chemin survivant.

L'autre approche, TB, utilise la propriété de l'algorithme de Viterbi voulant que tous les chemins survivants de tous les états possibles, à un instant t, convergent avec haute probabilité au temps t-L en arrière. L est appelé la longueur du Trace Back. Elle doit être au moins cinq fois la mémoire du code m [13]. En pratique, on prend souvent L = 5m. Les bits de décision sont divisés en blocs et sauvegardés dans une mémoire partitionnée appelée mémoire du TB. Dès qu'un bloc de longueur L est disponible, le processus du TB est lancé. Il remonte en arrière un des chemins survivants en partant d'un état choisit au hasard. À la fin de ce processus, qui dure L itération, tous les chemins survivants convergent vers un seul état. Cet état initialise le TB et décodage du bloc de L décisions suivantes. Le résultat est une séquence décodée en ordre inverse. Une interface réorganise et fournit à la sortie la séquence estimée. Cette interface de sortie peut être réalisée par un LIFO ("Last In First Out"). Cette approche nécessite moins de connections et provoque moins d'activités de commutation. Cependant, son majeur inconvénient est la latence qu'elle cause. L'état au temps t-L est décodé au temps t avec une latence de L itérations. L'unité SMMU a un impact déterminant sur la latence générale du décodeur de Viterbi. Ce type d'approche est utilisé dans le cadre de ce projet de mémoire. La figure 13 montre le schéma bloc utilisant l'approche du TB.

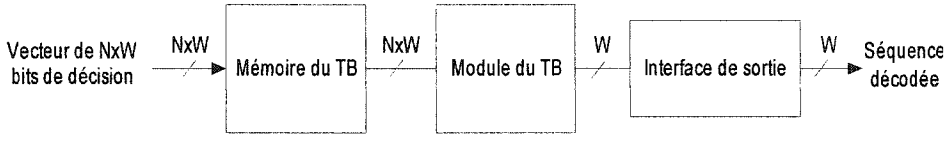


Figure 13 Schéma bloc du trace back pour un code de taux W/S et de nombre d'états N

2.7 Conclusion

Dans ce chapitre, le concept du codage convolutionnel, une brève étude des performances des codes convolutionnels et le décodage à maximum vraisemblance par l'algorithme de Viterbi ont été revus. Les performances d'un système codé s'améliorent avec la croissance de la distance libre, puisque la probabilité d'erreur décroît exponentiellement avec la croissance de cette distance. La distance libre croît presque linéairement avec la croissance de la mémoire du code. D'autre part, une augmentation linéaire de la mémoire du code entraîne une augmentation exponentielle de la complexité.

Dans la suite de ce chapitre nous avons introduit l'implémentation conventionnelle de cet algorithme sous forme de trois principaux modules : BMU, ACSU et SMMU. Nous avons vu que la quantification douce améliore les performances du décodeur par rapport à la quantification dure occasionnant une augmentation de la complexité du décodeur, à savoir une plus grande table des métriques et une augmentation de la mémoire requise pour la sauvegarde des métriques de chemin. Les opérations du module ACSU sont récursives rendant ce module un facteur limitant de la vitesse du décodage. Le chapitre suivant présente l'architecture proposée d'un décodeur de Viterbi paramétrable. La conception de ce décodeur tiendra compte de la complexité du décodeur de Viterbi ainsi que de l'engorgement du débit créé par l'unité ACSU qui fera l'objet d'une particulière attention.

CHAPITRE 3

ARCHITECTURE PROPOSÉE

La structure du décodeur de Viterbi a été présentée en détail au chapitre précédent. La récursion du bloc des Additions-Comparaisons-Sélections, ACS, fait de ce dernier un goulot d'étranglement pour le débit du décodeur. En considérant la nature des opérations qu'il doit effectuer, notamment quelques additions, comparaisons et sélections par état, ce bloc sollicite de plus en plus d'espace sur la surface totale occupée par le décodeur dans le circuit intégré, au fur et à mesure que le nombre d'états augmente. Le choix d'une architecture avec un bon compromis vitesse-surface devient primordial dans la conception d'un tel décodeur.

Dans ce chapitre, nous présentons l'architecture choisie. Cette dernière utilise la technique radix-4 qui permet de doubler le débit de la transmission tout en gardant une bonne efficacité en surface. Elle permet aussi de réduire la complexité du module TBU ("Trace Back Unite") et de la pile LIFO. Une description détaillée de tous les modules qui composent ce décodeur paramétrable et flexible est l'objectif principal de ce chapitre.

3.1 Choix de la technique radix-4

N'importe quel treillis de $N = 2^m$ états peut être divisé en 2^{m-x} sous-treillis et être itéré du temps i - x à i en une seule itération en utilisant un étage du treillis radix-2^x. La division du treillis n'affecte pas les performances du décodeur et le chemin décodé est le même. La figure 16 montre la décomposition d'un treillis de 8 états (radix-2), pour un taux de codage de R = 1/2, en deux sous-treillis de radix-4.

Dans le cas idéal, la décomposition du treillis en sous-treillis radix-2^x donne un gain de vitesse de x pour une augmentation de la complexité de 2^{x-1} par rapport à celle du treillis original radix-2. Un résumé de l'augmentation de la vitesse et de la complexité en fonction du radix du treillis utilisé est donné au tableau I [16]. Nous remarquons que la complexité augmente exponentiellement pour une augmentation linéaire de la vitesse. En pratique, une augmentation linéaire de la vitesse ne peut pas être réalisée due à l'augmentation exponentielle du nombre des métriques des chemins à comparer. Le treillis radix-4 est d'une importance particulière car il permet, théoriquement, de doubler la vitesse tout en gardant un rapport vitesse sur surface (efficacité en surface) de 1. Nous avons donc choisi d'utiliser l'architecture de radix-4

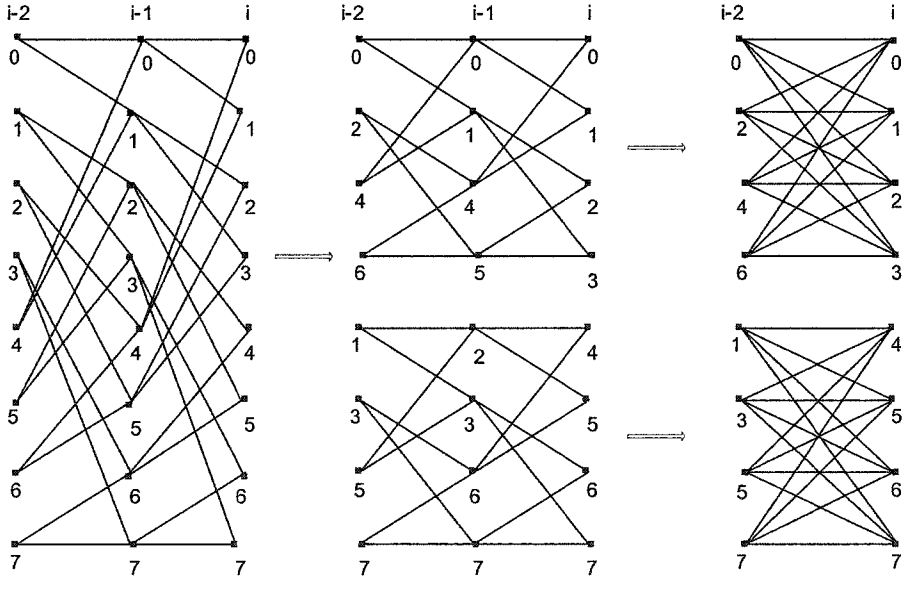


Figure 14 Décomposition du treillis de 8 états, R = 1/2, en sous-treillis radix-4

Tableau I

Mesure de la rapidité et de la complexité de radix-2^x

Radix	x	Croissance de vitesse	Croissance de complexité	Efficacité en surface
2	1	1	1	1
4	2	2	2	1
8	3	3	4	0.75
16	4	4	8	0.50

3.1.1 Forme générale du radix-4

N'importe quel treillis de $N = 2^m$ états peut être décomposé en N/4 sous-treillis radix-4 dont la forme générale est montrée à la figure 17 [17]. Comme on peut le constater, les formules de calcul des numéros des états de départ et d'arrivée peuvent facilement être regroupées en deux formules générales qu'on peut aisément programmer pour générer automatiquement le treillis à partir du nombre d'états N.

- Pour chaque sous-treillis radix-4 avec le numéro d'ordre J, où J = 0,1,2,....(N/4)-1, on calcule les numéros des quatre états de départ par la formule suivante :

$$S = J + l * N/4, \text{ où } l = 0,1,2,3. \tag{3.1}$$

- Pour chaque sous-treillis radix-4 avec le numéro d'ordre J, où J = 0,1,2,....(N/4)-1, on calcule les numéros des quatre états d'arrivée par la formule suivante :

$$S = 4 * J + l, \text{ où } l = 0,1,2,3. \tag{3.2}$$

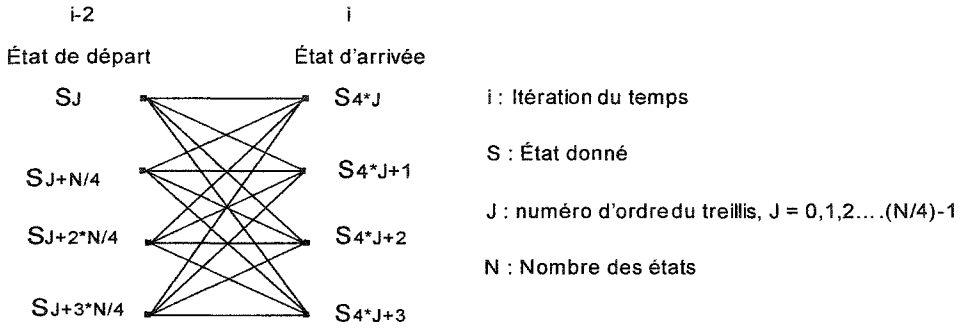


Figure 15 Forme générale de la décomposition en treillis radix-4

3.2 Module de calcul des métriques de branche BMU

Le module BMU calcule la différence entre les symboles reçus et les symboles de chaque branche du treillis. Dans ce module se fait aussi le contrôle du taux de codage. Dans notre design nous pouvons réaliser deux groupes de taux : 1/S et 2/S. Cette affirmation se base sur le fait que le treillis radix-4 d'un code de taux 1/S est le même que le treillis ordinaire, c'est à dire un treillis radix-4, d'un code de taux 2/S.

3.2.1 Taux de 1/S

Dans ce cas, le module BMU travaille à une fréquence qui est le double de la fréquence des autres modules car il combine deux itérations du treillis ordinaire radix-2 en une seule itération radix-4. Il fournit au module suivant les 2^S x 2^S = 2^{2S} métriques de branche nécessaires en combinant les 2^S métriques de branche d'une itération radix-2 avec les 2^S autres de l'itération suivante.

Le module BMU reçoit S symboles de décision douce et calcule les 2^S métriques de branche $\lambda n(0)$, $\lambda n(1)$, $\lambda n(2)$,..... $\lambda n(2^S-1)$, qui correspondent aux 2^S états possibles de

sortie de l'encodeur : 0,1,2,.... 2^S-1 respectivement. Ces métriques de branche sont retardées d'une itération simple de radix-2 puis sont additionnées au résultat de l'itération suivante pour former les 2^{2S} métriques de branche d'une seule itération de radix-4. Le tableau II donne un exemple de formation d'indices des métriques de branche de radix-4 pour un taux de codage de R = 1/2. Ici S = 2, ce qui donne quatre métriques de branche par itération de radix-2. Elles seront combinées avec les quatre autres métriques de branche de l'itération suivante pour donner un ensemble de 16 métriques de branche radix-4.

Les métriques de branche radix-2 sont programmables. Cela rend le décodeur flexible puisqu'il peut utiliser n'importe quels types des métriques de branche. Ces dernières, calculées selon la méthode voulue, seront enregistrées dans une table de conversion ("Look Up Table", LUT) via l'entrée *LUT_in* (voir figure 18). Le signal *RW* sert à la permission d'écriture de ces données. Les S symboles d'entrées concaténés forment les bits d'adresse pour cette LUT.

La programmation des métriques de branche s'effectue selon une séquence de 2^g coups d'horloge puisqu'il y a 2^g vecteurs possibles correspondants aux g bits d'adresse. Ces vecteurs sont formés de 2^S x M bits chacun puisqu'il y a 2^S distances à évaluer par itération de radix-2. Le paramètre M est générique et exprime la largeur du mot de métrique de branche radix-2. La largeur de métrique de branche radix-4 sera égale à M+1 puisqu'elle est composée d'addition de deux métriques de branche de radix-2.

En fonctionnement normal (décodage de Viterbi), les symboles d'entrée du décodeur sont concaténés pour former l'adresse de lecture du vecteur de 2^S métriques de branche radix-2 correspondantes. Ces métriques seront ensuite combinées avec les 2^S métriques de l'itération suivante pour former les métriques de branche de radix-4.

Tableau II

Exemple de formation d'indice des métriques de branche radix-4 pour R = 1/2

	λ(n)			
λ(n-1)	00	01	10	11
00	0000	0001	0010	0011
01	0100	0101	0110	0111
10	1000	1001	1010	1011
11	1100	1101	1110	1111

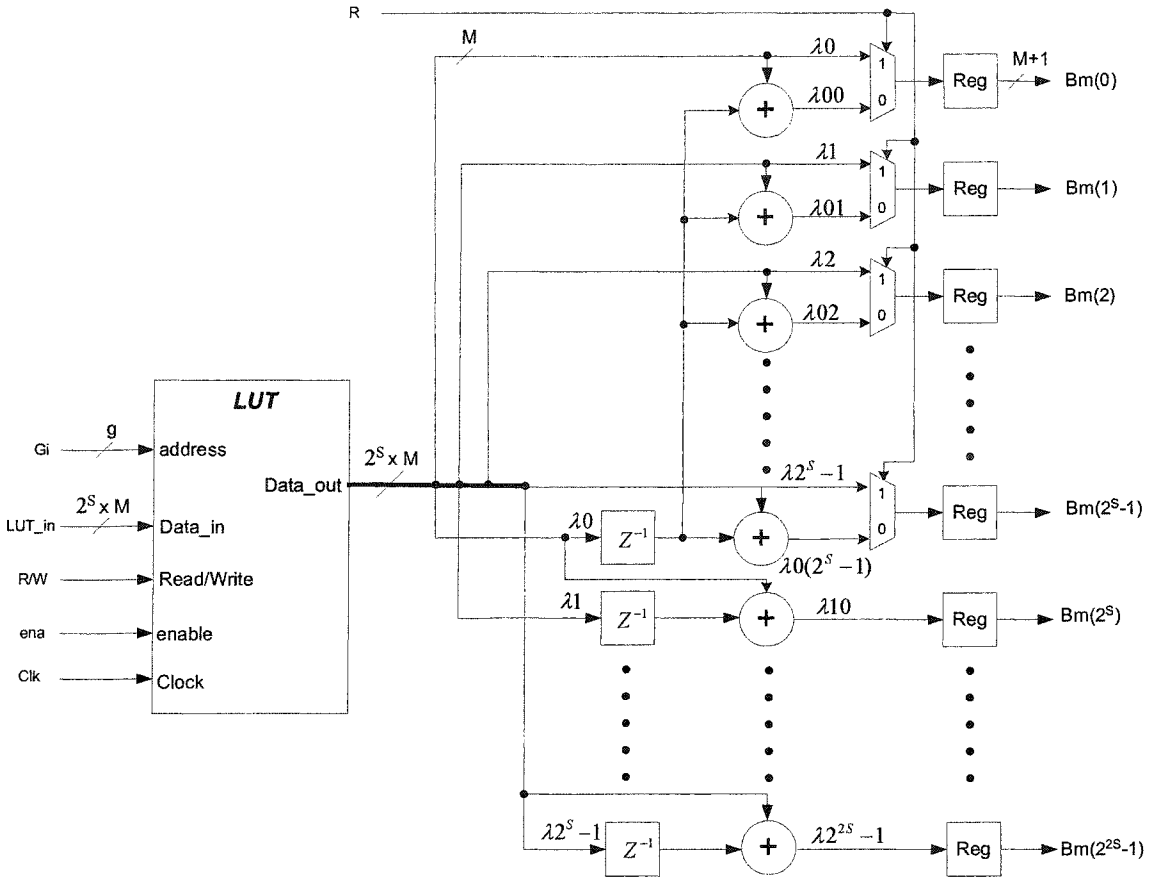
3.2.2 Taux de 2/S

Pour ces taux, le module BMU travaille à la même fréquence que tous les autres modules du décodeur. Il fonctionne de la même façon que pour les taux de 1/S sauf qu'il ne combine pas les 2^S métriques de branche radix-2. À chaque itération radix-2, ce module lit dans la LUT un vecteur de 2^S métriques de branche possibles correspondant aux S symboles d'entrée et le fournit tel qu'il est à la sortie.

Le contrôle des taux se fait par le signal R qui agit sur les multiplexeurs. Ces derniers sélectionnent quelles métriques sont fournis à la sortie du module BMU : soit les métriques combinées pour le taux 1/S ou les métriques directes pour les taux 2/S.

La figure 18 démontre le diagramme bloc de ce module pour un taux de codage de R = W/S où W = 1,2 et S = 1,2,3,....

Remarque : Le module BMU permet de commuter du taux 1/S au taux 2/S sans devoir reprogrammer la table des métriques de branche pour le même S.



M : largeur du mot des métriques de branche radix-2

g : largeur du bus formé par la concaténation des symboles d'entrée

Figure 16 Module BMU, R = W/S où W = 1,2 et S = 1,2,3,…

3.3 Module de l'Addition-Comparison-Sélection ACSU

3.3.1 Module ACS à quatre chemins

Chaque état du treillis radix-4 est réalisé par un module ACS à quatre chemins (4-ways ACS) comme illustré à la figure 19. Quatre chemins entrent à chaque état. Les métriques associées à ces chemins sont additionnées avec les métriques de branche

correspondantes pour donner les métriques des chemins mises à jour. Ces métriques doivent être comparées afin de conserver la plus petite. Deux bits de décision sont issus du comparateur indiquant quel chemin a été choisi.

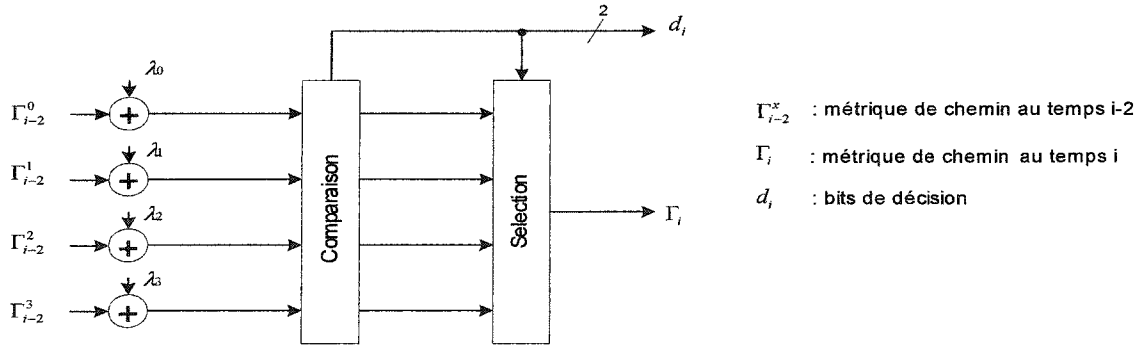


Figure 17 Bloc ACS à 4-chemins représentant un état du treillis radix-4
R = W/S où W = 1,2 et S = 1,2,3,...

3.3.2 Paire d'addition-comparaison

L'addition et la comparaison se font en parallèle du bit le moins significatif (LSB) au bit le plus significatif (MSB). La figure 20 démontre le diagramme bloc d'une paire d'addition-comparaison sur huit bits. L'addition et la comparaison se font en utilisant l'arithmétique appelée "Ripple Carry" ou propagation de retenue. La comparaison est implémentée en utilisant la soustraction au complément à deux. Seulement la chaîne de retenue (Carry) de la soustraction est utilisée pour générer le signe de la soustraction et donc de la comparaison. Ainsi le délai de l'addition-comparaison ensemble est en première approximation égal au délai de l'addition plus celui d'un bit d'additionneur complet (full adder) [18].

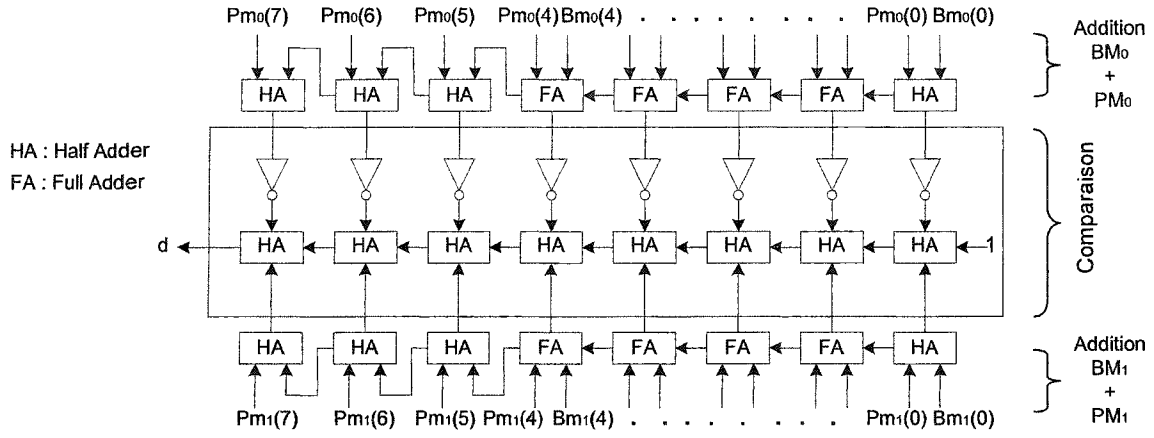


Figure 18 Diagramme bloc illustrant une paire d'addition-comparaison simultanées à 8-bits

3.3.3 Arithmétique modulo pour le calcul des métriques des chemins

Les valeurs des métriques des états et des branches sont toujours positives puisqu'elles correspondent à une mesure de distance. Leur addition continuelle conduira à un certain moment au débordement. Pour éviter la normalisation des métriques d'état, qui est coûteuse en matérielle et en vitesse, on utilise l'approche de l'arithmétique modulo proposée en [16]. Cette approche dit que deux nombres x et y tel que $|x-y| < \Delta$, qui doivent être comparés en utilisant la soustraction, peuvent être comparés comme (x-y) modulo 2Δ sans ambiguïté. L'arithmétique modulo exploite l'effet que l'algorithme de Viterbi limite la plage dynamique maximale de la métrique d'état Δ_{max} comme le démontre la formule suivante :

$$\Delta_{max} \leq \lambda_{max} . \log_2 N \qquad (3.3)$$

où N est le nombre d'états et λ_{max} est la valeur maximale de la métrique de branche. $\lambda_{max} = 14$ pour le cas de la décision douce à 3 bits et un taux de codage de 1/2.

Les métriques des chemins peuvent être additionnées aux métriques de branche et comparées en modulo $2\Delta_{max}$. Donc en choisissant la précision appropriée de la métrique de chemin (nombre de bits), l'arithmétique modulo est implicitement implémentée en ignorant le débordement de la métrique d'état. Cette précision se calcule comme suit :

$$\Gamma_{bits} = \left[\log_2\left(\Delta_{\max} + 2.\lambda_{\max}\right)\right] + 1 \qquad (3.4)$$

où le terme $2.\lambda_{max}$ prend en considération l'augmentation potentielle de la plage dynamique pour le ACS radix-4 par rapport à radix-2, causée par l'addition des métriques de branche.

Le tableau III montre le résultat de calcul de la largeur des bits de la métrique de chemin pour le nombre d'états donné pour une quantification douce sur 3 bits et λmax =14. Comme on peut le constater, la largeur de mot de la métrique des chemins ne varie pas beaucoup en fonction du nombre d'états N. Cela indique que cette technique est appropriée pour la conception d'un décodeur paramétrable.

Tableau III

Nombre de bits, Γ_{bits}, en fonction de N pour R = 1/2

N	8	16	32	64	128	256
Δ_{max}	42	56	70	84	98	112
Γ_{bits}	7	7	8	8	8	8

3.3.4 Comparaison modifiée

La comparaison modifiée vient corriger et compléter l'arithmétique modulo. En effet, due à l'arithmétique modulo, la comparaison ordinaire au complément à 2 n'est pas

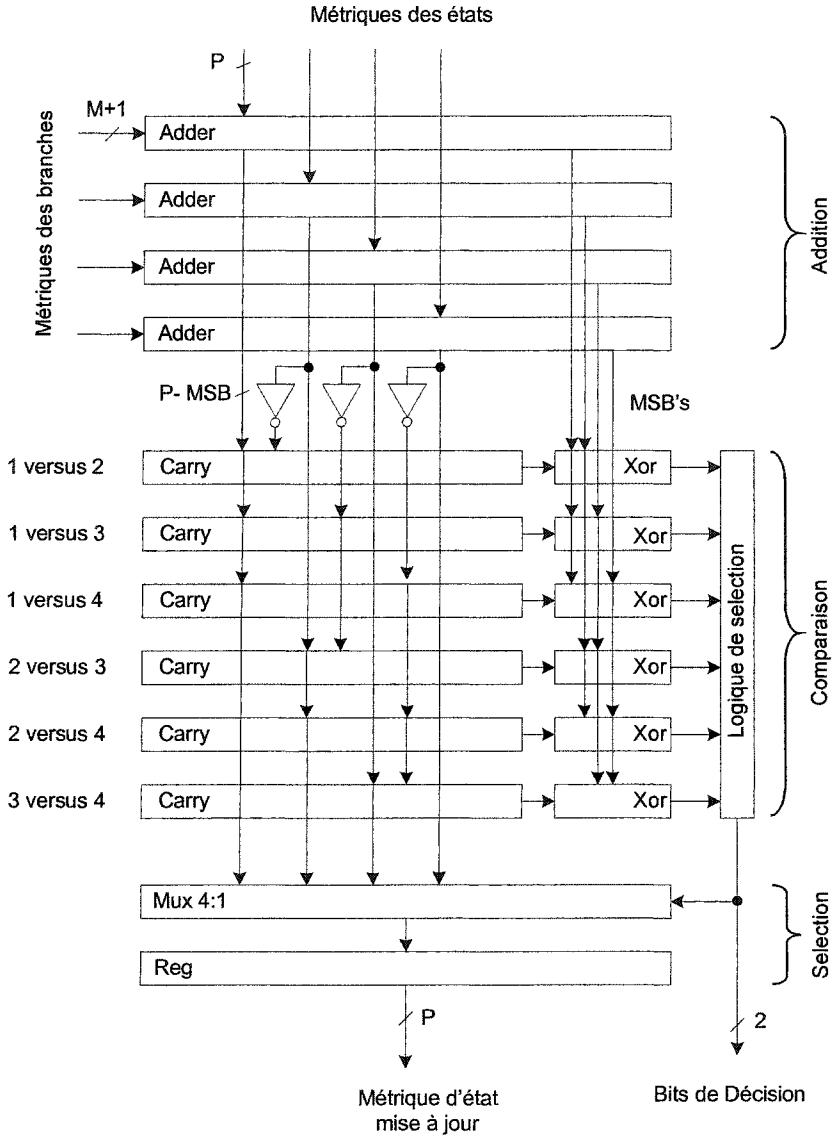
capable de déterminer l'ordre relatif entre deux nombres à comparer si un seul des deux déborde. Par exemple, considérons deux métriques de chemins sur 4 bits chacun $m_1 = 1011$ et $m_2 = 0111$ à mettre à jour avec les métriques de branche correspondantes $bm_1 = 101$ et $bm_2 = 110$ et à comparer. Cela donnera $\overline{m}_1 = 0000$ (débordement) et $\overline{m}_2 = 1101$ et la comparaison sera fausse. Pour remédier à cela, nous mettons en œuvre la comparaison modifiée dont le principe repose sur la formule suivante [34]:

$$Z(\overline{m}_1, \overline{m}_2) = MSB_1 \oplus MSB_2 \oplus y(\overline{m}_1, \overline{m}_2) \tag{3.5}$$

où $Z(\overline{m}_1, \overline{m}_2)$ dénote la comparaison entre les deux nombres $\overline{m}_1$ et $\overline{m}_2$, $y(\overline{m}_1, \overline{m}_2)$ dénote la comparaison entre $\overline{m}_1$ et $\overline{m}_2$ auxquels a été enlevés le bit le plus significatif (MSB) et où MSB_1 et MSB_2 sont respectivement les MSB de $\overline{m}_1$ et $\overline{m}_2$.

En d'autres termes, $Z(\overline{m}_1, \overline{m}_2)$ est égale à $y(\overline{m}_1, \overline{m}_2)$ (ou à l'inverse logique de $y(\overline{m}_1, \overline{m}_2)$) si $\overline{m}_1$ et $\overline{m}_2$ sont de même signe (ou de signes opposés).

Le diagramme bloc final d'un ACS à quatre chemins, qui représente un état du treillis radix-4, est montré à la figure 21



M : largeur du mot des métriques de branche radix-2

P : largeur du mot des métriques de chemin

P-MSB : largeur du mot des métriques de chemin sans le MSB

Figure 19 Diagramme bloc du module ACS à 4-chemins

3.3.5 Gain en vitesse

Comparé à l'ACS de référence à 2 chemins, qui représente un état du treillis radix-2, le délai à travers l'ACS à 4 chemins a augmenté dû aux facteurs suivants : l'augmentation de l'étalement des sorties des additionneurs, la logique additionnelle pour la génération du signal de contrôle du multiplexeur à partir des six couples de comparaison, et le délai additionnel du multiplexeur 4:1 comparé à celui de 2:1. Le délai total du ACS à 4 chemins est de 17% plus long que celui du ACS à 2 chemins de structure similaire, résultant au total en une augmentation de vitesse (taux de données) de 1.7. Ce résultat à été obtenu pour une implémentation en circuit ASIC de technologie CMOS 1.2 μM [16].

3.3.6 Flexibilité du module des ACS

La flexibilité du module des ACS se traduit par la capacité du décodeur à faire le décodage pour tous les codes non récursifs de taux R = W/S avec S = 1,2,3,.. et W = 1,2. Cela est réalisé à l'aide des multiplexeurs de 2^{2S} à 1 si W = 1 ou de 2^{S} à 1 si W = 2. Ces multiplexeurs sélectionnent la métrique de branche correcte pour l'additionneur de l'état correspondant comme le montre la figure 22. Ces multiplexeurs sont contrôlés par des vecteurs de contrôle (bmSelect) sauvegardés dans des registres. Ces vecteurs de contrôle sont calculés à partir des polynômes du décodeur. Le principe de calcul de ces vecteurs de contrôle est présenté dans l'annexe 1.

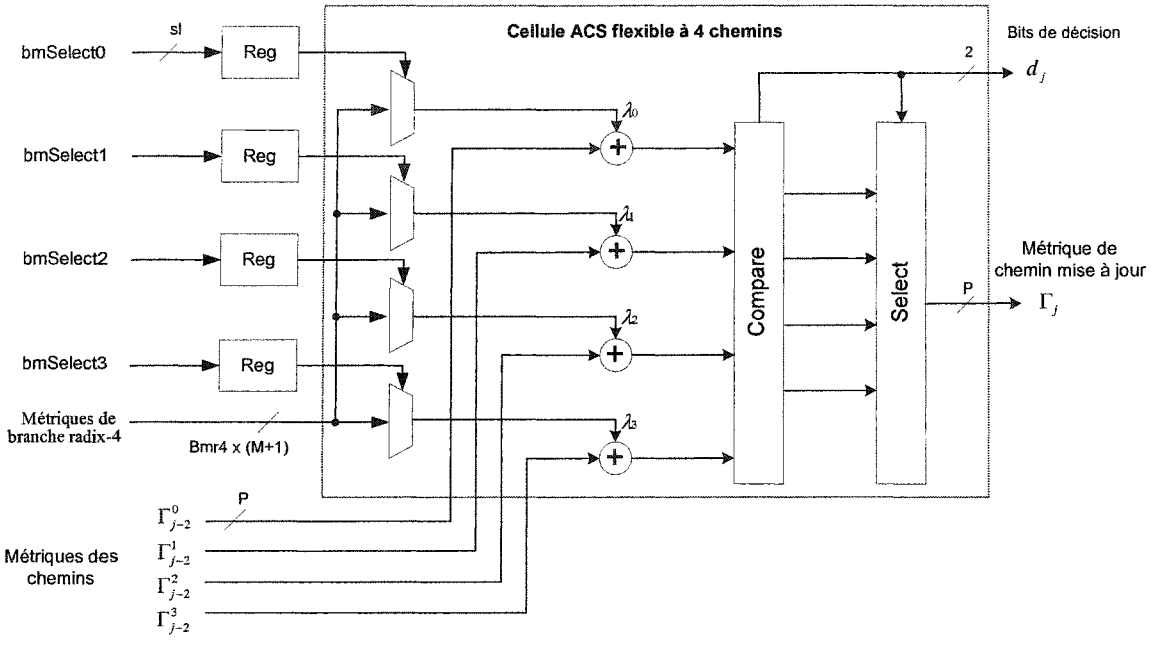


Figure 20 Cellule ACS à 4-chemins, R = W/S où W = 1,2 et S = 1,2,3,... bmr4 = 2^{2S} (ou 2^{S}) et sl = 2S (ou S) si R = 1/S (ou 2/S)

3.4 Unité du "Trace Back" TBU

Le décodeur de Viterbi traite un flot continu des données. Cependant, en pratique et pour le module du Trace Back (TB), nous devons diviser le flot continue des décisions, fournies par l'unité précédente ACSU, en blocs de longueur finie L pouvant être sauvegardés dans la mémoire du TB pour les traiter plus tard et en déduire les bits décodés. L est la longueur du Trace Back.

3.4.1 Organisation de la mémoire

La mémoire des décisions est organisée en tampon cyclique ("cyclic buffer") : elle est conceptuellement partitionnée en régions d'écriture et de lecture comme montré à la figure 23. Durant chaque phase de décodage, les décisions nouvellement fournies par le

bloc ACSU sont écrites dans la région d'écriture pendant que le chemin survivant est tracé et décodé depuis la région de lecture. Le "Trace Back" de la région de lecture procède à une analyse à rebours de la convergence des chemins d'une profondeur de L bits, l'état final étant utilisé pour initialiser le décodage du bloc de lecture de L bits des décisions restantes. Lorsque ce bloc est décodé, les décisions qui y sont écrites seront écrasées et leur place devient une région d'écriture pour la phase suivante.

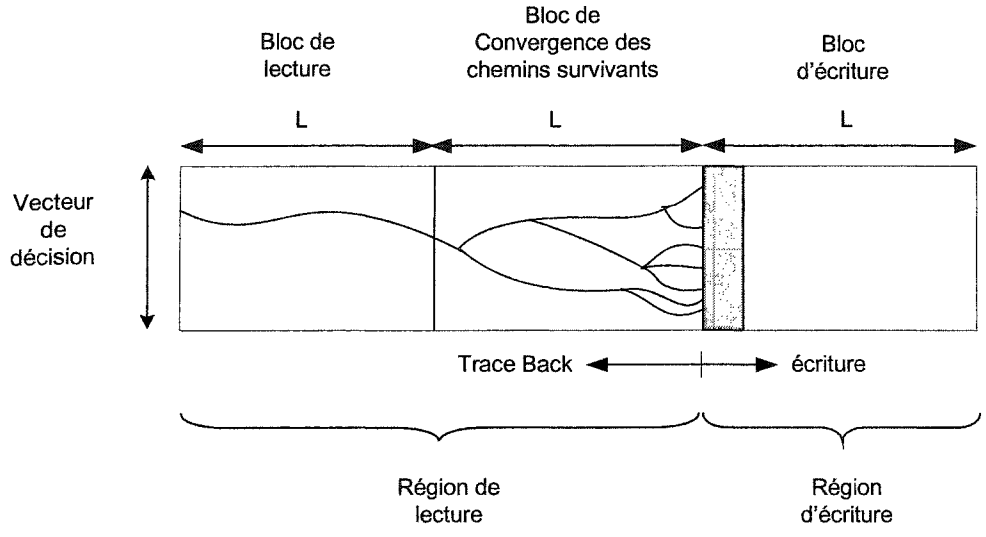


Figure 21 Organisation de la mémoire des décisions en tampon cyclique

3.4.2 Flux des RAM de la mémoire du TB

La mémoire du TB est organisée en blocs de quatre RAM ("Random Access Memory") de longueur effective, $L_{eff} = L/2$, chacune. La figure 24 démontre l'ordre d'accès de ces RAM. Un cycle complet de cette mémoire se fait en quatre périodes. Par simplicité mais sans perte de généralité, on suppose que la période 1 est le début du décodage.

Période 1 : Les bits des décisions, nouvellement fournis par le ACSU, sont enregistrés dans la RAM 1. La lecture des RAM 2 et 4 est pour l'instant inutile puisqu'il n'y a rien d'écrit encore. La RAM 3 est en état de repos (attente).

Période 2 : La RAM 1 devient une zone de convergence, mais sa lecture est pour l'instant inutile puisque à la fin de cette période elle doit fournir l'état pour l'initialisation du TB de la RAM 4. Cependant, la RAM 4 ne contient pas encore de données valides. La RAM 2 sauvegarde la suite des bits des décisions. La lecture de la RAM 3 est inutile. La RAM 4 est en état de repos.

Période 3 : Après $2L_{eff}$ itérations, la RAM 2 joue le rôle du bloc de convergence et cette fois il va nous fournir à la fin de cette période l'état d'initialisation pour le TB et le décodage des bits écrits dans la RAM 1. Les bits des décisions sont maintenant sauvegardés dans la RAM 3. La lecture de la RAM 4 est inutile. La RAM 1 est en état de repos.

Période 4 : Après $3L_{eff}$, le TB et le décodage du contenu de la RAM 1, qui était écrit pendant la période 1, débutent en prenant comme départ l'état fourni à la fin de la période 3 par le bloc de convergence de la RAM 2. Ainsi, le premier bit décodé apparaîtra à la sortie du TBU après une latence de $3L_{eff}$ itérations. La RAM 3 devient un bloc de convergence qui pointera le contenu de la RAM 2. Les nouveaux bits de décision sont sauvegardés cette fois dans la RAM 4. La RAM 2 est en état de repos.

Le cycle se répète, mais toutes les lectures inutiles du premier cycle seront désormais utiles : pendant la période 1, le TB et le décodage prendront place dans la RAM 2 qui a été pointée par le résultat de la convergence dans la RAM 3 à la fin de la période 4, et la RAM 4 contiendra le bloc de convergence pour pointer la RAM 3 et ainsi de suite.

Remarques : Les RAM ayant été en état de repos pendant une certaine période, sont en attente du résultat de la convergence. C'est à partir de ce dernier que sont initialisés le TB et le décodage de la période suivante.

La longueur des RAM est égale à $L_{eff} = L/2$ puisqu'on a deux bits de décision à la fois par état grâce à l'utilisation de la technique radix-4.

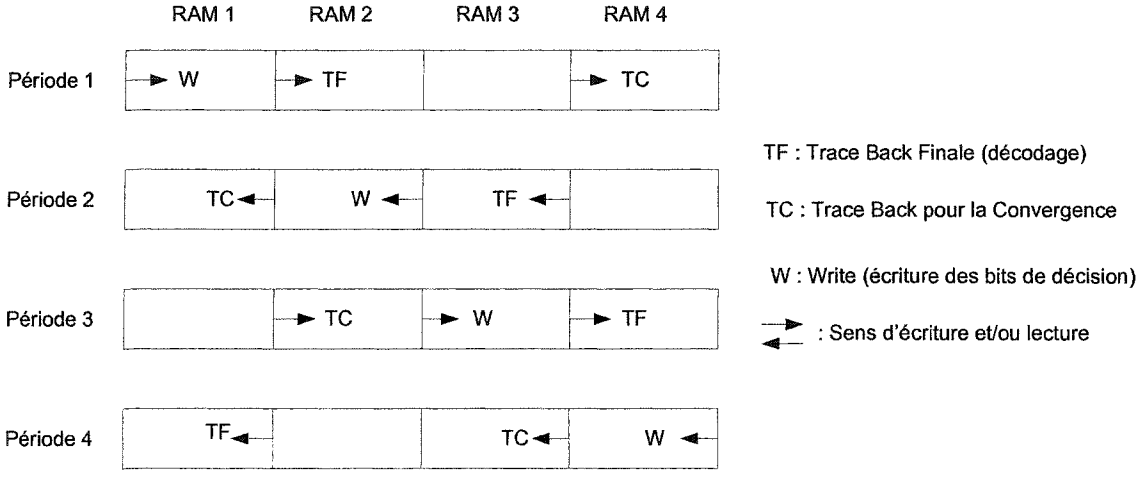


Figure 22 Flot de la mémoire du TB à quatre RAM

Le module du TBU complet est montré à la figure 25. Le bus des adresses *adrs* est commun pour les quatre RAM : si on remarque bien dans la figure 24, les lectures et les écritures se font dans le même sens pour une période donnée. Le bus de 2 bits *MCTRL* sert pour le contrôle de la mémoire du TB comme l'indique le tableau IV. On remarque que l'écriture, la lecture pour la convergence et la lecture finale pour différente RAM se font simultanément. Les quatre RAM sont alimentées par le même bus de données par lequel à chaque itération passe un vecteur des décisions de N x 2 bits. Les bits des décisions lus à partir des RAM sont décodés par le bloc MUX_logic dont le fonctionnement est présenté au paragraphe suivant. Le signal *MCTRL* contrôle aussi un multiplexeur qui sélectionne la sortie du bloc MUX_logic qui fournit des données

valides, en d'autres termes, celui qui est en période de décodage. La sortie *MCTRL(0)* sert à contrôler le LIFO suivant le TB.

3.4.3 Principe de décodage

Le principe de décodage est simple : l'état de départ du processus de convergence est arbitraire. Il se trouve dans un registre à décalage à droite chargeable (figure 26). Le contenu de ce registre sélectionne, dans un multiplexeur de N à 1, l'adresse des 2 bits de décision du chemin qui mène à cet état. Quand ces bits de décision décalent (décalage à droite) le contenu du registre qui représente l'état actuel, l'état précédent est trouvé. Il montre l'adresse des bits de décision précédente et ainsi de suite. À la fin de ce processus l'état de départ du TB est trouvé et il est chargé dans le registre du bloc MUX_logic suivant par l'entrée *load_in* à l'aide du signal de contrôle (load/shift).
Voici un exemple pour mieux illustrer le principe de décodage réalisé par le bloc MUX_logic : Soit S_n = 1001101 est l'état actuel contenu dans le registre et d = 10 est la décision correspondante. Quand d décale à droite le contenu du registre (S_n), l'état précédent est trouvé, soit S_{n-1} = 1010011 et les bits décodés sont 01 (les deux bits les moins significatifs de l'état S_n).

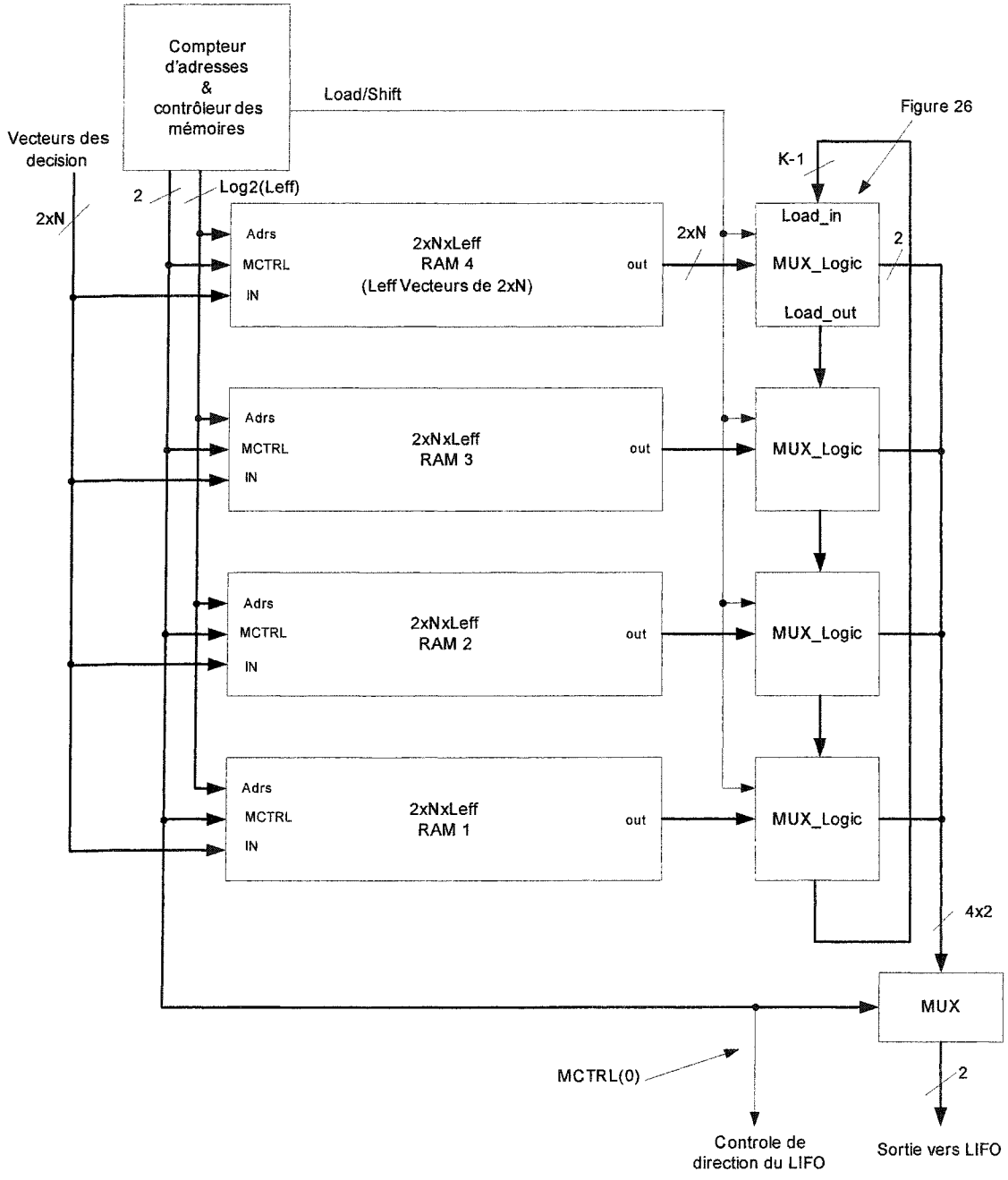


Figure 23 Diagramme bloc du Trace Back

Tableau IV

Contrôle de la mémoire du TB

	Signale de contrôle des RAM, MCTRL			
RAM	00	01	10	11
1	Écriture	Lecture TC	Repos	Lecture TF
2	Lecture TF	Écriture	Lecture TC	Repos
3	Repos	Lecture TF	Écriture	Lecture TC
4	Lecture TC	Repos	Lecture TF	Écriture

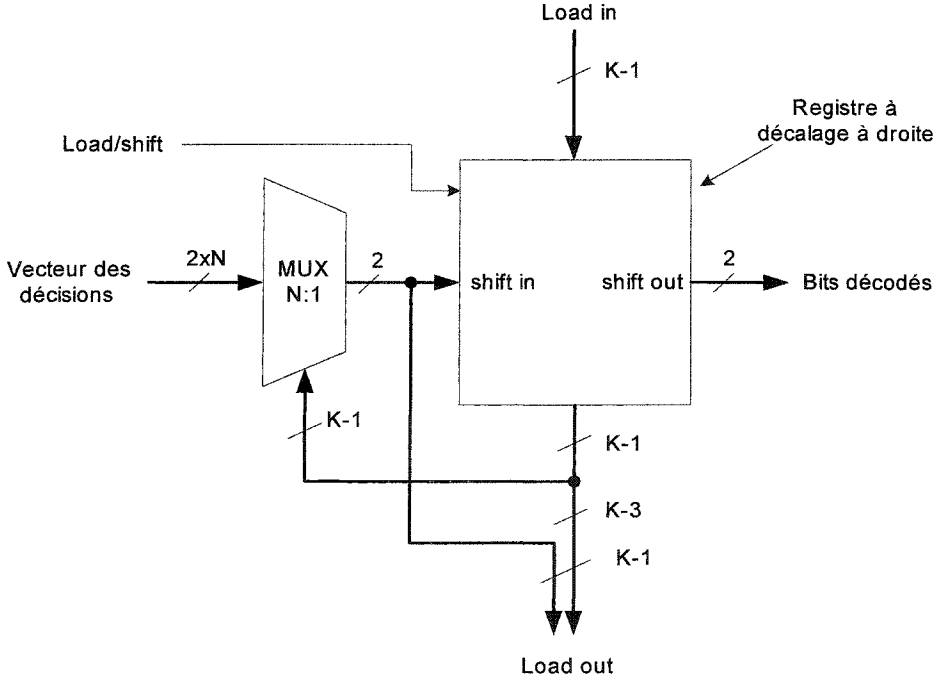
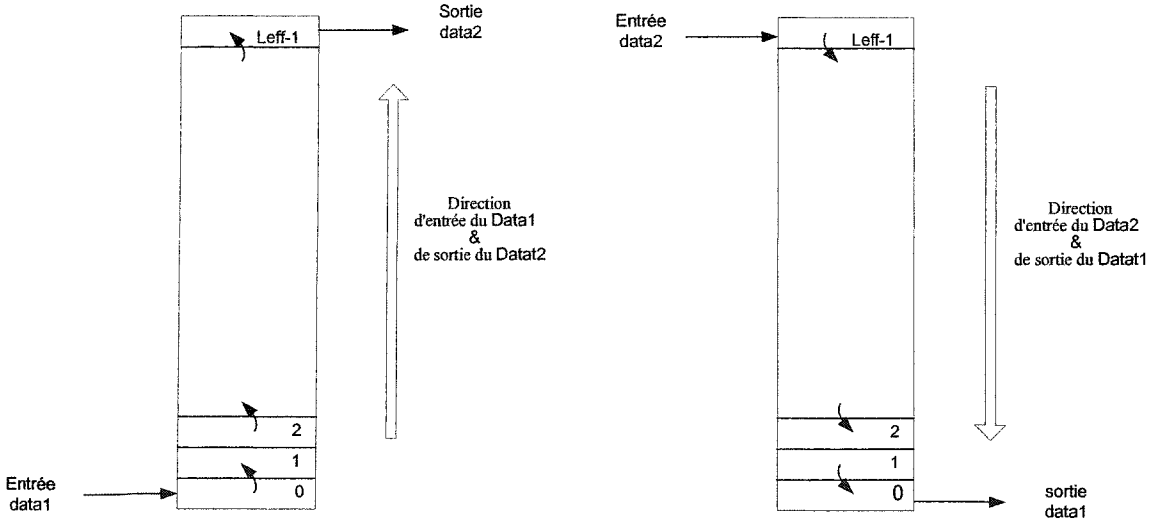


Figure 24 Bloc Mux_logic

3.5 Unité LIFO

Les séquences décodées provenant du TBU se trouvent dans l'ordre inverse, d'où la nécessité de les réorganiser à l'aide d'une pile LIFO (Last In First Out). Ces séquences sont continues, ce qui implique l'utilisation de deux piles LIFO. Lorsque l'une est en train de délivrer la séquence organisée à la sortie, l'autre se remplit avec la suite des données. Pour réduire la quantité de mémoire requise à la moitié nous utilisons une pile LIFO bidirectionnelle dont le schéma fonctionnel est montré à la figure 27. Dans une première étape, et au fur et à mesure que la pile LIFO se remplit des données du data1 dans le sens montant, elle se vide des données du data2 dans le même sens comme le montre la figure 27(a). La deuxième étape commence lorsque la pile s'est vidée de toutes les données du data2 et s'est remplit des données du data1. Pendent cette étape, la pile commence à se vider des données du data1 et à se remplir avec les données du data2 dans le sens descendant comme le montre la figure 27(b). La direction du LIFO, qui doit changer chaque fois qu'un nouveau bloc de longueur L_{eff} vient d'être terminé, est contrôlée par le signal *MCTRL(0)*, issu du module précédent TBU. Ce même signal sert aussi pour le contrôle du démultiplexeur d'entrée et du multiplexeur de sortie comme le démontre la figure 28.

L'utilisation de la technique radix-4 permet de fournir deux bits décodés par itération. Dans le cas des taux 1/S, une itération radix-4 se fait en deux coups d'horloge. Par contre elle se fait en un seul coup d'horloge pour les taux de 2/S. Pour cette raison nous n'avons pas intégré d'interface parallèle-série au décodeur qui aurait fourni une sortie de un bit décodé par coup d'horloge.



(a) Remplissage du data 1 et vidange du data2 (b)Vidange du data 1 et remplissage du data2

Figure 25 Schéma fonctionnel de la pile LIFO bidirectionnelle

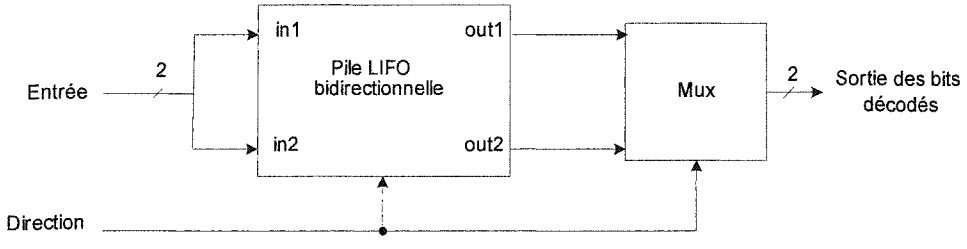


Figure 26 Unité LIFO

3.6 Diagramme bloc général du décodeur de Viterbi

Le diagramme bloc du décodeur est présenté à la figure 29. Ce décodeur doit être programmé avant de commencer le décodage de Viterbi. Il faut programmer les métriques de branche dans la table LUT et les vecteurs de contrôle dans le bloc à registres.

L'écriture des métriques de branche dans le LUT se fait en 2^g coups d'horloge. Les vecteurs des métriques de branche de 2^SxM sont appliqués à l'entrée LUT_in. Les symboles d'entrée sont concaténés ensemble pour former l'adresse G_in de la LUT. Le signal RW autorise l'écriture ou la lecture des métriques de branche à partir de cette LUT.

La sauvegarde dans des registres des vecteurs de contrôle prend Nx4 coups d'horloge, cette période varie donc en fonction du nombre d'état N. Ces vecteurs de contrôle sont appliqués à l'entrée *bmSelect*. L'entrée *Reg_adrs* sert d'adresse pour les registres de sauvegarde. Le signal *W_ena* permet l'écriture dans ces registres.

La figure 30 montre le chronogramme illustrant le fonctionnement du décodeur : période de programmation et période de fonctionnement normal (décodage de Viterbi).

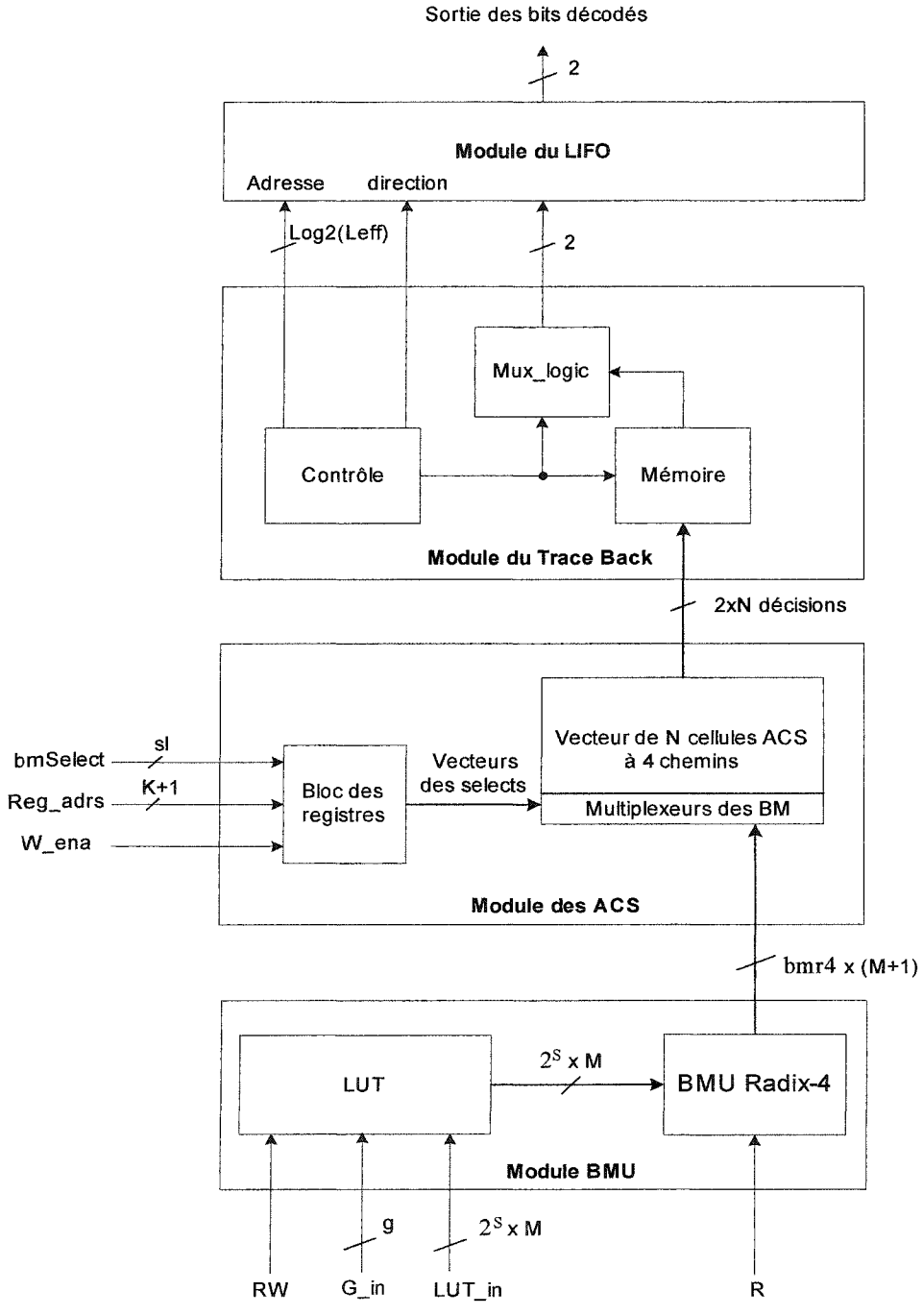


Figure 27 Diagramme bloc du décodeur de Viterbi, R = W/S où W = 1,2 et S = 1,2,3,... bmr2 = 2^S, bmr4 = 2^{2S} (ou 2^S) et sl = 2S (ou S) si R = 1/S (ou 2/S)

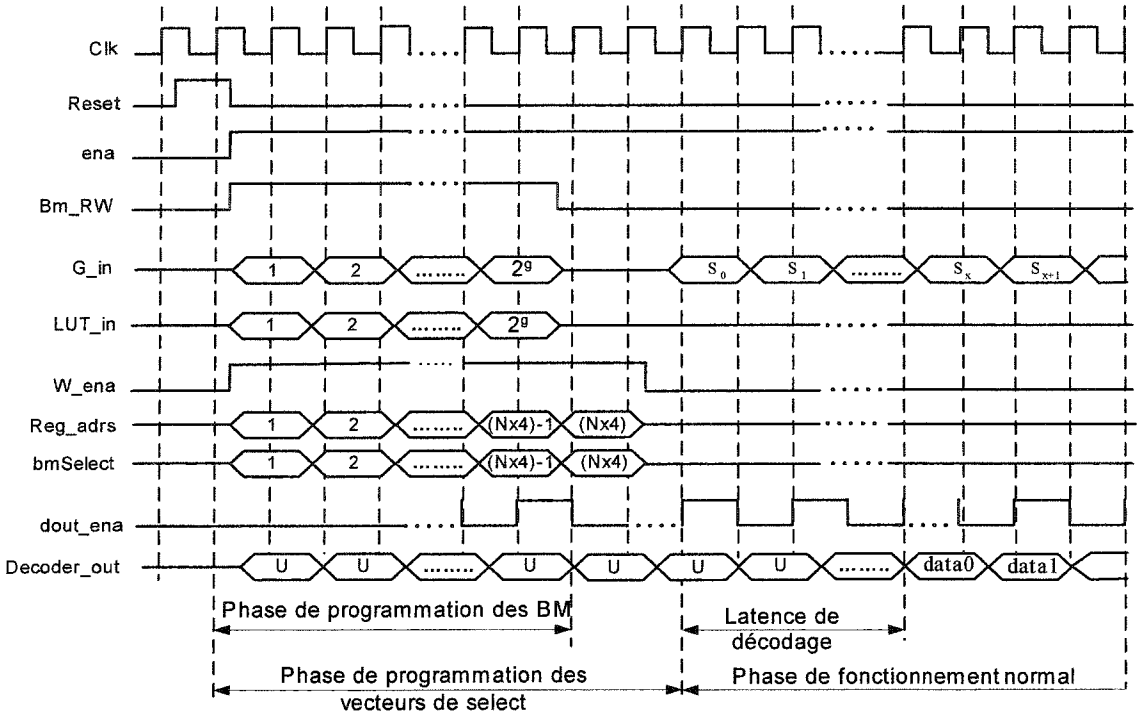


Figure 28 Chronogramme des phases de fonctionnement du décodeur, R = 1/S

3.7 Conclusion

Dans ce chapitre, l'architecture d'un décodeur de Viterbi générique, flexible et utilisant la technique radix-4 a été présentée. Cette technique permet de doubler le débit du décodeur en se basant sur l'utilisation du ACS à 4 chemins opérant à la même fréquence d'itération du ACS ordinaire à 2 chemins. L'utilisation de l'arithmétique modulo pour les calculs des ACS nous permet d'éviter la normalisation des métriques de chemin. Cela réduit la complexité du décodeur et diminue le temps de traitement du module du ACSU.

De plus, l'utilisation de la technique radix-4 réduit la complexité du module du "Trace Back". Les bits de décision, traités par ce module, sont de deux bits par itération. Cela

réduit à moitié la profondeur du Trace Back, résultant en une réduction à moitié de la complexité des compteurs des adresses du module du Trace Back.
Grâce à l'utilisation des multiplexeurs pour l'acheminement de la métrique de branche adéquate à chaque cellule ACS, ce décodeur est capable de faire le décodage selon n'importe quels polynômes.

L'implémentation d'une table (Look Up Table) des métriques de branche en bloc RAM, intégré dans le FPGA, permet de réduire la complexité du module BMU et d'augmenter son temps de traitement. D'autre part, cela rend le décodeur flexible. On peut y programmer le type des métriques de branche adéquat dépendamment de la modulation utilisée afin d'obtenir plus de précision. De plus, ce module est capable de réaliser des différents taux pour la même synthèse.

CHAPITRE 4

IMPLÉMENTATION, TESTS ET RÉSULTATS

Au chapitre précédent, nous avons présenté l'architecture d'un décodeur de Viterbi paramétrable et flexible. Dans le présent chapitre, nous présentons le résultat de l'implémentation de ce décodeur dans un FPGA Virtex-II de Xilinx. Nous modélisons une chaîne de communication numérique opérant en bande de base afin de mesurer les performances de ce décodeur. Le décodeur de Viterbi testé ici accepte des symboles quantifiés sur huit niveaux et réalise le décodage du code standard : m = 6, R = 1/2, P1 = 171_8 et P2 = 133_8. Pour pouvoir juger du gain de codage du décodeur testé, nous traçons aussi la courbe de performance d'un système non codé.

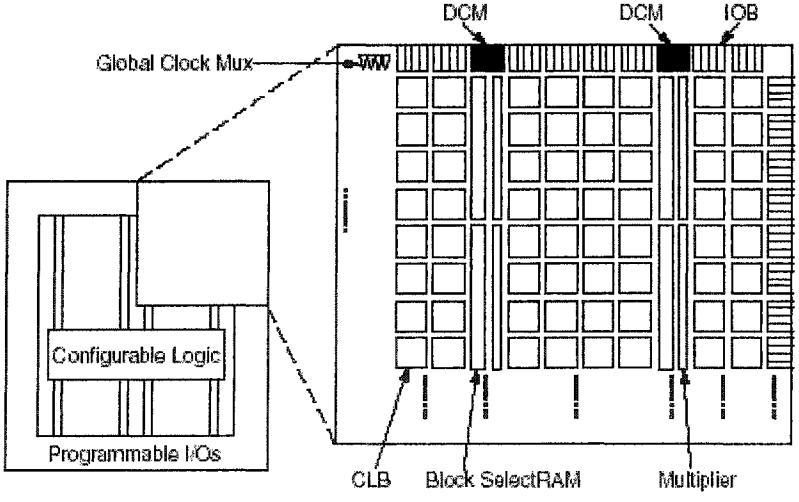
4.1 Architecture du FPGA de Xillinx

Un FPGA ("Field Programmable Gate Arrays") est un circuit prédiffusé contenant de la logique programmable. L'avantage de ce type de circuit est qu'il ne demande pas de fabrication dédiée en usine, ni de développement coûteux lors de l'implémentation d'un design. Il nécessite tout simplement d'être programmé pour réaliser la fonction requise.

Pour pouvoir tirer le maximum d'un circuit FPGA, il faut avoir une bonne connaissance de sa topologie interne. Cette section se veut une brève introduction à la topologie interne du circuit FPGA de type Virtex-II de la compagnie Xillinx. Le choix de ce circuit repose sur les besoins de l'architecture complexe du décodeur de Viterbi en matière de logique performante dédiée aux opérations arithmétiques, de mémoire RAM, de larges multiplexeurs et d'un système d'interconnections performant.

4.1.1 VIRTEX-II de Xillinx

Un circuit FPGA Virtex-II se compose principalement de blocs logiques configurables (CLB), de blocs RAM, de multiplicateurs, de matrices d'interconnections programmables, de lignes d'interconnexions et de blocs d'entrées/sorties IOB (figure 31). Le CLB est le module de base d'un FPGA Virtex-II. Il se compose de quatre tranches ("slices"). Une tranche se compose de deux LUT ("Look Up Table"), deux registres, de la logique de retenue (''Carry logic''), des portes logiques pour les opérations arithmétiques et des multiplexeurs (figure 32). Les LUT, ou générateurs de fonctions, sont des mémoires RAM capables d'implémenter toutes les fonctions logiques combinatoires de quatre entrées. Le délai de propagation est donc indépendant de la fonction implémentée. Dans une seule tranche, nous pouvons combiner à l'aide des multiplexeurs disponibles les deux LUT pour réaliser des fonctions de cinq, six, sept ou huit variables [27]. Les LUT peuvent également être configurées comme éléments de mémoire synchrone, simple ou double-port de 16 bits (RAM16), ou encore comme registre à décalage de longueur programmable jusqu'à 16 bits (SRL16).



DCM : diviseur/multiplicateur d'horloge.

Figure 29 Vue générale de l'architecture d'un circuit FPGA Virtex-II [28]

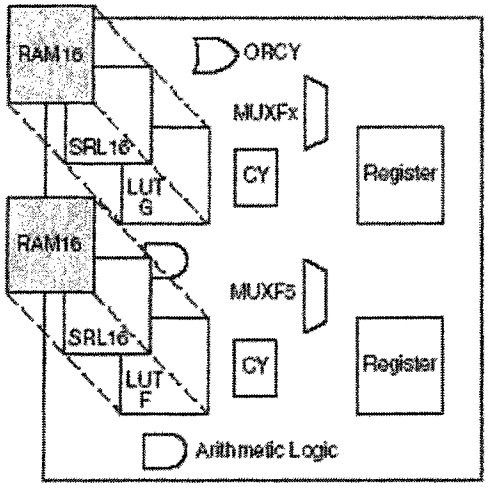


Figure 30 Configuration d'une tranche d'un circuit FPGA Virtex-II [28]

Tel que mentionné précédemment, chaque tranche contient des portes logiques pour la réalisation des opérations arithmétiques (figure 32). Il contient aussi une chaîne dédiée à la propagation de la retenue. Ces éléments bénéficient de ressources de routages dédiées pour garantir la performance de ces opérations. Par exemple, la propagation de la retenue s'effectue via des connexions directes entre tranches adjacentes (pas besoin de passer par les matrices d'interconnexions). Dans une seule tranche nous pouvons implémenter un additionneur complet (full adder) de 2 bits [27].

Une tranche contient aussi deux multiplexeurs 2:1 (MUXF5 et MUXFX) dédiés qui bénéficient de ressources spécifiques de routage garantissant ainsi une meilleure performance. Dans une seule tranche nous pouvons implémenter un multiplexeur 4:1. Dans un seul CLB on peut implémenter un multiplexeur 16:1.

De plus, un FPGA Virtex-II dispose de blocs RAM ("SelectRAM") synchrones de 18 K bits chacun. Ces blocs RAM sont de type double-port. Les dispositions possibles de ces blocs RAM sont : 16K x 1, 8K x 2, 4K x 4, 2K x 9, 1K x 18 et 512 x 36.

Les IOB constituent l'interface entre les broches du circuit et la logique interne développée. Chaque IOB contrôle une broche du circuit. Ainsi, une broche peut être configurée en entrée, en sortie, en entrée/sortie (bidirectionnelles) ou être non utilisée (haute impédance).

Les matrices d'interconnexions sont des matrices programmables qui permettent de relier les diverses lignes d'interconnexions entre elles. Les lignes d'interconnexions sont des segments métallisés répartis horizontalement et verticalement sur la totalité du circuit. Ils permettent le routage des signaux internes du circuit.

4.2 Cycles de design

La figure 33 illustre l'organigramme simplifié du cycle de design pour un circuit ciblant la technologie FPGA. Ils sont constitués des étapes suivantes :

- Spécifications fonctionnelles du projet : descriptions (diagrammes blocs), spécifications et caractéristiques techniques du projet à concevoir.
- Conception en VHDL : description textuelle en langage VHDL du matériel qui doit être implémenté dans le FPGA.
- Simulation fonctionnelle : simulation visant à vérifier la fonctionnalité du design. Elle vérifie si le code VHDL réalise les fonctions requises. C'est une simulation logique (ne prend pas considération des délais de la logique à implémenter).
- Synthèse logique : opération de transformation d'une description textuelle (VHDL) d'un comportement en schéma ou "netlist " (liste des noeuds) composée d'instances de cellules élémentaires.
- Simulation fonctionnelle après synthèse : simulation logique servant à vérifier si le code VHDL a été correctement synthétisé et s'il réalise encore les fonctions requises.

- Placement routage : disposition des cellules élémentaires nécessaires à la réalisation des fonctions de façon à satisfaire les contraintes d'occupation de surface et de synchronisation.
- Analyse de la synchronisation : analyse statistique des chemins critiques du design. Après le placement routage, les temps de propagation à travers la logique implémentée et les lignes d'interconnections sont disponibles. Ces temps de propagation déterminent la fréquence maximale du design. Si nécessaire, il faut réécrire le code VHDL et répéter toutes les étapes décrites ci-dessus pour réduire la longueur des chemins critiques du design.
- Simulation temporelle : simulation du code VHDL contenant en plus l'information sur les temps de propagation. Elle nous permet de vérifier dynamiquement la fonctionnalité du design.
- Génération du fichier "bitstream" : génération du fichier de configuration du circuit FPGA.
- Programmation du FPGA : pour un FPGA de Xilinx, c'est le chargement du fichier de configuration "bitstream"; il est habituellement gardé dans une mémoire PROM ("Programmable Read Only Memory") qui se trouve sur le même circuit imprimé que le FPGA et qui sert à garder le programme de configuration du FPGA après la mise hors tension du système.
- Test du système : test du design dans son vrai environnement (dans le circuit FPGA) pour s'assurer qu'il réalise encore les fonctions requises tout en respectant les contraintes imposées.

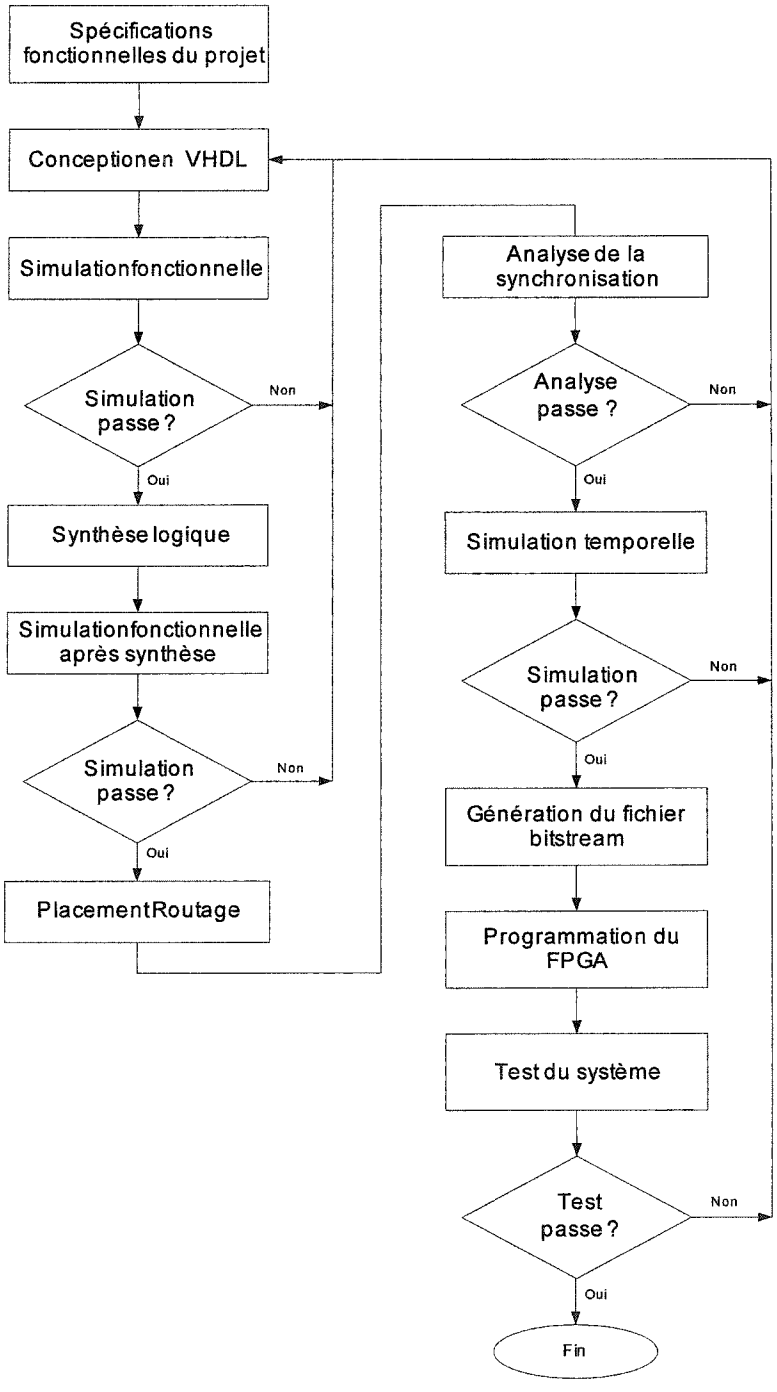


Figure 31 Organigramme du cycle du design

4.3 Modularité et hiérarchie du code VHDL

Notre design est modulaire. Sa hiérarchie VHDL est montrée à la figure 34. La modularité du design permet d'accroître considérablement le rendement du design en terme de temps de développement, de coût et de fiabilité. La division du design complexe en modules simples facilite sa conception. Ces modules réutilisables sont conçus et testés séparément.

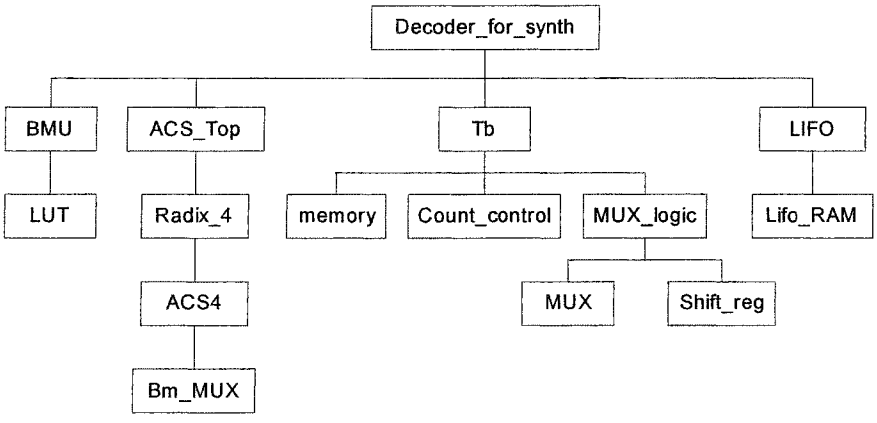


Figure 32 Hiérarchie du design

4.4 Optimisation du design

Le langage VHDL est un langage de description de matériel qui est en théorie portable, donc il est indépendant de la technologie ciblée (ASIC, FPGA, CPLD...). Cependant, pour permettre à l'outil de synthèse de bien optimiser un design et de prendre avantage des particularités de la technologie ciblée, nous devons adapter notre style d'écriture du code VHDL. Cette section traite du résultat de synthèse pour différents styles de codage VHDL des modules du design. Ces résultats justifient notre choix. Les modifications apportées à l'architecture du décodeur de Viterbi, proposé au chapitre 3, seront présentées. Notons que les résultats de synthèse présentés dans ce chapitre ont été

obtenus en utilisant l'outil de synthèse XST ("Xilinx Synthesis Tool") intégré au logiciel ISE de Xilinx. Le circuit FPGA ciblé est un Virtex-II de Xilinx, le XC2V6000, avec un grade de vitesse de –5.

4.4.1 Unité BMU et TBU

Ces deux unités se composent en grande partie de blocs de mémoire. L'unité BMU contient une table de conversion implémentée en mémoire RAM. L'unité TBU contient quatre mémoires servant à stocker les bits des décisions. Pour réduire l'occupation en surface du FPGA, nous utilisons les mémoires blocs RAM, "SelectRAM Block", disponibles dans ce type ce circuit. Les résultats de synthèse de ces deux unités seront présentés dans les sections qui suivent.

4.4.2 Unité ACSU

Cette unité, critique en terme de vitesse et d'occupation de surface, utilise beaucoup d'opérations d'additions et de comparaisons. La solution pour implémenter une paire d'addition-comparaison rapide, proposée dans [18] et présentée à la section 3.3.2 du chapitre 3, cible une application ASIC. Ici, l'utilisation d'un FPGA nous incite à adapter notre style de design afin que les outils de synthèse et de placement routage puissent mieux l'optimiser. Comme il a été mentionné précédemment, un FPGA Virtex-II contient des chaînes dédiées à la retenue et des éléments logiques pour la réalisation des opérations arithmétiques. La compagnie Xilinx montre dans son "Application Note" XAPP215 qu'une addition ou une comparaison sur 8 bits occupe seulement 4 tranches dans un FPGA Virtex-II. Ce résultat est obtenu si le code VHDL utilise un domaine de description comportementale (utilise des signes mathématiques pour décrire les opérations arithmétiques) et s'il est correctement synthétisé [29]. C'est pourquoi nous utilisons les signes mathématiques "+" et ">" plutôt qu'une description de flot de

données dans notre code VHDL pour la description des paires d'addition-comparaison. Le tableau V présente un résumé des résultats de la synthèse des deux types de description du code VHDL de la cellule ACS à quatre chemins de la figure 21 (chapitre 3). Les rapports 1 et 2 montrant respectivement les résultats de la synthèse du code VHDL utilisant une description de flot de données et les résultats de la synthèse du code VHDL utilisant une description comportementale, sont montrés à l'annexe 2. Pour simplifier la lecture des rapports, nous avons présenté seulement les parties se rapportant sur l'occupation de surface et sur la synchronisation. Dans le rapport 2, nous constatons que l'outil de synthèse a utilisé de la logique dédiée pour les opérations d'addition-comparaison (XORCY et MUXCY). Cela a permis d'économiser des tranches et d'obtenir une plus grande fréquence d'opération maximale.

Tableau V

Résumé des résultats de la synthèse du code VHDL de la cellule ACS à quatre chemins pour les deux types de description : flot de données et comportementale

Type de description	Nombre de Tranches	Nombre de registres	Nombre de LUT	Fréquence maximale (Mhz)
Flot de données	93	62	162	103
Comportementale	81	62	99	111

4.4.3 Unité LIFO

L'architecture de l'unité LIFO proposée à la section 3.5 (chapitre 3) peut être avantageuse pour une réalisation ASIC puisqu'elle utilise une seule pile LIFO constituée de L_{eff} registres de deux bits chacun, ($L_{eff} = L/2$). Dans le cas de notre design, qui utilise un FPGA Virtex-II, il est plus avantageux de tirer profit des blocs mémoires RAM qui y sont présents. Une autre alternative de conception d'une pile LIFO consiste en

l'utilisation d'une mémoire RAM et d'un compteur "up/down". Ce dernier génère les adresses d'écriture et de lecture. La séquence de bits inversés stockés dans la pile LIFO est lue dans le sens inverse. La figure 35 montre le diagramme bloc d'une unité LIFO utilisant ce type de pile LIFO. Ici, nous utilisons deux piles LIFO de longueur de L_{eff} bits chacune pour gérer le flot continu de données à l'entrée. En effet, l'utilisation d'une seule pile LIFO bidirectionnelle (chapitre 3, section 3.5) demande de la logique de contrôle complexe. D'autre part, la quantité de mémoire demandée pour la réalisation d'une pile LIFO est assez faible. Pour une longueur du TB de L = 64, (ce qui est souvent utilisé pour une mémoire de code m = 6), la quantité de mémoire demandé est de 2 x 64/2 bits = 64 bits, (L_{eff} = L/2).

Afin d'optimiser notre design, nous utilisons le même compteur "up/down" qui se trouve dans l'unité précédente, TBU, pour générer les adresses de lecture et d'écriture de l'unité LIFO.

Le tableau VI présente un résumé des résultats de la synthèse des deux architectures de l'unité LIFO discutés ci-dessus. Les rapports 3 et 4 montrant respectivement les résultats de synthèse (occupation en surface seulement) d'une unité LIFO utilisant des registres et d'une unité LIFO utilisant des blocs RAM, sont présentés à l'annexe 2.

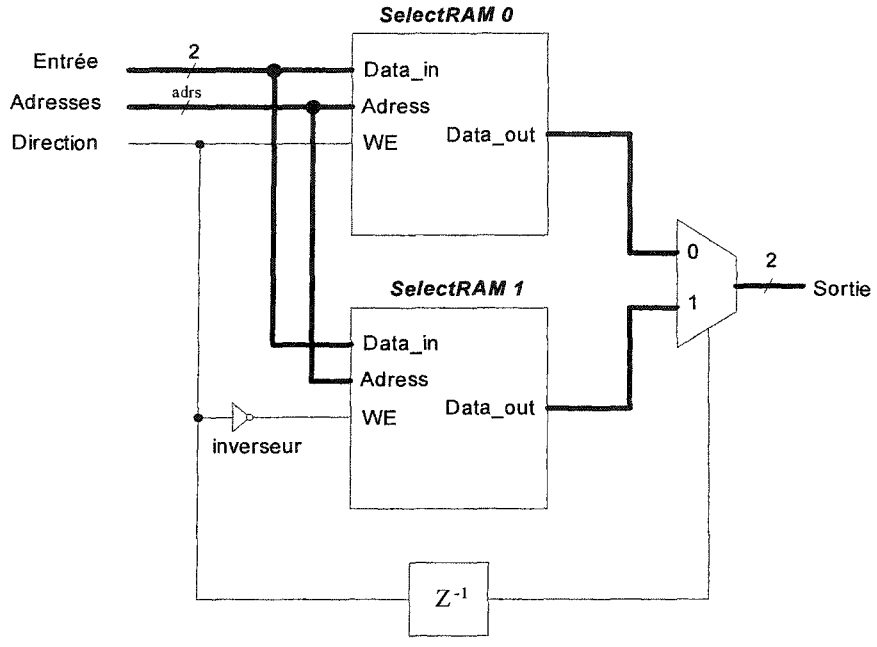


Figure 33 Diagramme bloc de l'unité LIFO utilisant des blocs RAM

Tableau VI

Résumé des résultats de la synthèse du code VHDL
de l'unité LIFO utilisant des registres et utilisant des blocs RAM

Type d'unité LIFO	Nombre de Tranches	Nombre de registres	Nombre de LUT	Nombre de bloc RAM
registres	37	70	68	0
Blocs RAM	3	5	3	1

4.4.4 Décodeur entier

Chaque unité du design a été synthétisée et analysée individuellement afin de trouver l'unité la plus critique en occupation de surface et en vitesse. La synthèse a été faite pour

le code de mémoire m = 6, de taux de codage R = 1/2 et de longueur du TB de L = 64. Un résumé des résultats est présenté au tableau VII. Les chiffres présentés entre parenthèses correspondent au pourcentage d'occupation en surface. Les rapports correspondants, (de 5 à 9), sont placés à l'annexe 2. Ces résultats confirment que le module ACSU est le module le plus critique en occupation de surface et en vitesse. Le module ACSU crée un goulot d'étranglement pour le débit général du décodeur à cause de ses opérations lentes et récursives. Il a aussi le pourcentage le plus élevé d'occupation de surface à cause de sa complexité due au fait qu'il reflète le treillis du code donnée. C'est ce module qui fait la majorité des calculs d'un décodeur de Viterbi. D'autre part, ce module contient de la logique additionnelle qui donne un certain niveau de flexibilité au décodeur. Cette flexibilité permet au décodeur de faire le décodage selon n'importe quel polynôme générateur. Cette logique est constituée de registres et de multiplexeurs (chapitre 3, section 3.3.6). Afin de savoir combien de ressources supplémentaires elle nécessite, nous avons aussi synthétisé une version du module ACSU non flexible. Cette logique additionnelle occupe 10532 LUT. Cette valeur paradoxale est presque de 50% plus grande que la valeur d'occupation en surface du module ACSU non flexible. Cela peut être explicable par un calcul estimatif de la complexité de la logique de flexibilité. Comme il a été mentionné précédemment, un multiplexeur 16:1 de 1 bit de largeur occupe 1 CLB. Nous avons besoin de 4 multiplexeurs 16:1 de 5 bits de largeur par état. Dans le cas de l'implémentation donnée, N = 64 états, le nombre de CLB nécessaire pour implémenter tous les multiplexeurs indispensables à la flexibilité du module ACSU est :

$$4 \times 64 \times 5 = 1280 \text{ CLB}$$

Un CLB de Virtex-II contient 4 tranches, chacune contenant 2 LUT. Donc, le nombre nécessaire de LUT pour implémenter les multiplexeurs en question est :

$$1280 \times 4 \times 2 = 10240 \text{ LUT}$$

Aussi, nous avons besoin de 4 registres de 4 bits par état. Le nombre nécessaire de registre de 1 bit de largeur est :

$$4 \times 64 \times 4 = 1024 \text{ registres}$$

L'outil de synthèse utilise les registres des mêmes CLB déjà utilisés pour l'implémentation des multiplexeurs, puisqu'il y en a 2 par tranche et donc 8 par CLB. Nous estimons la ressource nécessaire à l'adressage des 256 registres de 4 bits à 2 LUT par registre, ce qui donne le nombre de 512 LUT au total. Ceci vient porter le nombre de LUT nécessaire à l'implémentation de la logique de flexibilité à 10752 LUT. Évidement, l'outil essaie de simplifier la logique requise, ce qui explique la différence entres les deux valeurs trouvée et estimée.

Tableau VII

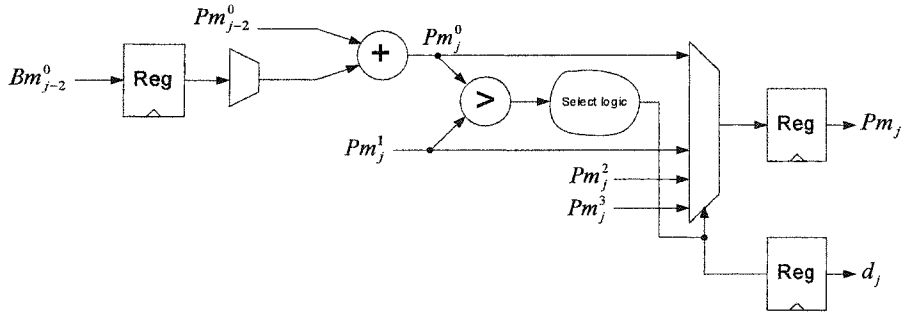
Résumé des résultats de la synthèse des modules du décodeur, (m = 6, R = 1/2 et L=64)

Module	BMU	ACSU	TBU	LIFO	ACSU non flexible
Occupation en surface (LUT)	90	16804 (24%)	312	3	6272 (9%)
Utilisation des blocs RAM	1	0	8 (5%)	1	0
Fréquence maximale (MHZ)	168	84	147	293	104

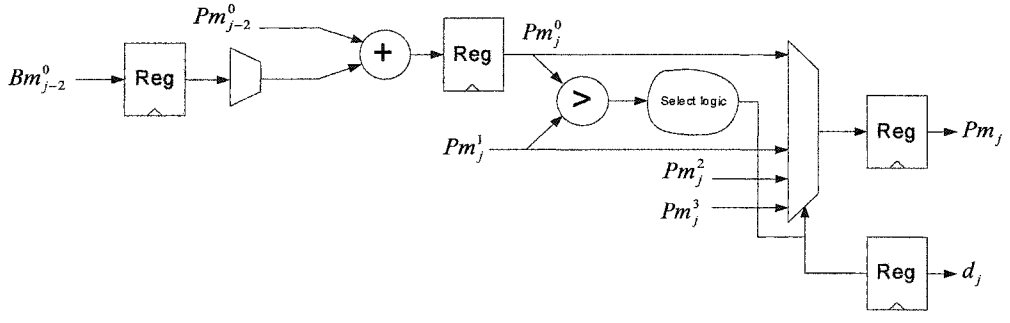
4.4.4.1 Amélioration de la vitesse du décodeur

Comme nous l'avons constaté précédemment, le module ACSU limite la vitesse du décodeur. Donc, améliorer la vitesse du décodeur revient à améliorer celle de ce module. Parmi les façons les plus utilisées pour améliorer la vitesse d'un design, il y a le pipeline et le balancement des registres ("retiming"). Le pipeline consiste à diviser la logique combinatoire en de plus petites parties et à insérer entre eux des registres pour réduire le chemin critique à travers cette logique combinatoire. Le balancement des registres est le déplacement de ces derniers à travers la logique combinatoire de façon à équilibrer le chemin entre eux. Cependant, dû à la récursivité des métriques des états, nous ne pouvons appliquer le pipeline qu'au décodage avec les taux de R = 1/S. Dans le cas du décodage avec ces taux, le module ACSU reçoit des nouvelles métriques de branche à chaque deux coups d'horloge. Cela nous donne la possibilité d'insérer un étage de pipeline dans les opérations d'addition-comparaison-sélection. Dans le cas où le décodage se fait pour des taux de R = 2/S, le module des ACSU reçoit des nouvelles métriques de branche à chaque coup d'horloge. Cela nous empêche d'introduire du pipeline puisque les métriques de chemin doivent être fournies, elles aussi, à chaque coup d'horloge. Ici, nous utilisons le balancement des registres comme façon d'améliorer la vitesse du module en question. La compagnie Synplicity montre dans ses "Application Note" "Syndicated" [31] et "Using Retiming" [32] que le balancement des registres améliore les performances du design jusqu'à 20 % pour la synthèse. La figure 36 (b) montre l'insertion d'un étage de pipeline dans l'opération d'addition-comparaison-sélection. La figure 36 (c) montre le balancement des registres appliqué à l'opération d'addition-comparaison-sélection. Les résultats de la synthèse du décodeur entier sans amélioration de la vitesse, avec pipeline et avec balancement des registres pour les opérations d'addition-comparaison-sélection sont présentés au tableau VIII. Les rapports correspondants, 10, 11 et 12 sont placés à l'annexe 2. Nous remarquons une augmentation de fréquence de 60 Mhz équivalente à une augmentation de débit de donnée de 120 Mb/s pour le décodeur avec pipeline. Pour le décodeur avec balancement

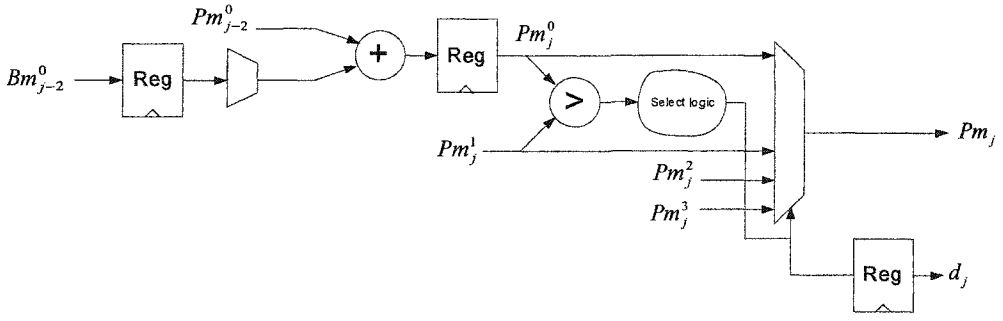
des registres, nous remarquons une augmentation de fréquence de 20 Mhz équivalente à une augmentation du débit de 40 Mb/s. Dans les deux cas, l'amélioration de la vitesse est réalisée au prix d'une faible augmentation du nombre de tranche et de registres, 4 tranches et 3 % de registres pour le pipeline, 7 tranches et 2 % de registres pour le balancement des registres.



(a) Opération d'addition-comparaison-sélection sans amélioration pour la vitesse.



(b) Opération d'addition-comparaison-sélection avec pipeline pour les taux de R = 1/S.



(c) Opération d'addition-comparaison-sélection avec balancement des registres pour les taux de R = 2/S.

Figure 34 Amélioration de la vitesse pour l'opération d'addition-comparaison-sélection du module ACSU.

Tableau VIII

Résumé des résultats de la synthèse du décodeur entier sans améliorations de la vitesse, avec pipeline et avec balancement des registres pour les opérations d'addition-comparaison-sélection, (m = 6, R = 1/2 et L = 64)

Décodeur de Viterbi	Nombre de tranches	Nombre de registres	Nombre de LUT	Nombre de bloc RAM	Fréquence maximale (Mhz)
Sans amélioration	9151 (27%)	1835 (2%)	17213 (25%)	10 (6%)	83
Avec pipeline	9155 (27%)	3886(5%)	17213 (25%)	10 (6%)	143
Avec balancement	9158 (27%)	3376 (4%)	17213 (25%)	10 (6%)	103

Les valeurs des résultats de la synthèse du décodeur de Viterbi présentés ci-dessus pour l'implémentation donnée sont des valeurs estimatives. Afin de savoir l'occupation en surface, la fréquence maximale d'opération et le dédit réelles du décodeur, nous avons

procédé au placement et routage de notre design avec pipeline et avec balancement. Un résumé des résultats est présenté au tableau IX. Les rapports correspondants 13 et 14 sont placés à l'annexe 2.

Tableau IX

Résumé du résultat de placement et routage du décodeur de Viterbi avec balancement des registres, (m = 6, R = 1/2 et L = 64)

Décodeur de Viterbi	Occupation en surface (tranches)	Nombre de bloc RAM	Nombre d'entrées/sorties	Fréquence maximale (Mhz)	Débit net de donnée (Mbps)
Avec pipeline	9361 (27%)	10 (6%)	43	100.5	201
Avec balancement	9365 (27%)	10 (6%)	43	81	162

4.5 Test et résultat

4.5.1 Environnement de test

La figure 37 illustre la méthode de test utilisée pour tester le décodeur de Viterbi conçu afin de tracer la courbe du taux d'erreur, BER, ("Bit Error Rate"). Un générateur de bits aléatoires fournit au codeur convolutionnel un flot de bits aléatoires. Les symboles codés de sortie du codeur convolutionnel subissent une transposition antipodale de bande de base. Un bit '0' est représenté par un '+1V' et un bit '1' est représenté par un '-1V'. Ensuite, le bruit blanc gaussien est ajouté au signal résultant. Le résultat est quantifié sur huit niveaux (3 bits). Les seuils et l'espacement pour la quantification douce sur huit niveaux sont représentés à la figure 38. Il a été démontré qu'une quantification uniforme avec un espacement de 0.5 pour huit niveaux est très proche de l'optimum [30]. Le

décodeur de Viterbi reçoit les symboles codés, bruités et quantifiés et livre à la sortie les bits décodés. Ces bits décodés sont comparés avec les bits retardés du message initial provenant du générateur de bits aléatoires. Un compteur compte le nombre d'erreurs trouvées et calcul le BER par la formule suivante :

$$BER = \frac{nombre \quad d'erreurs}{nombre \quad total \quad des \quad bits \quad envoyés} \tag{4.1}$$

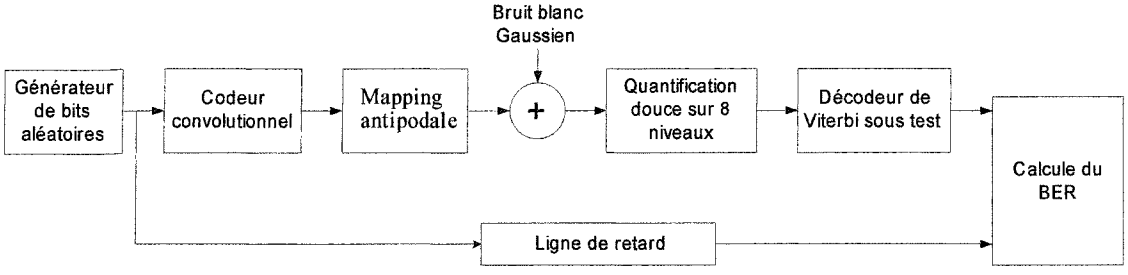


Figure 35 Illustration de l'environnement de test du décodeur.

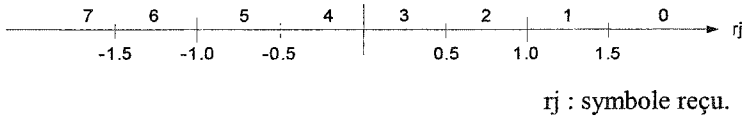


Figure 36 Seuils et espacement de la quantification douce sur huit niveaux.

4.5.2 Procédures de test

L'organigramme de la figure 39 illustre les procédures de test utilisées pour tracer la courbe du BER en fonction du SNR (rapport signal à bruit).

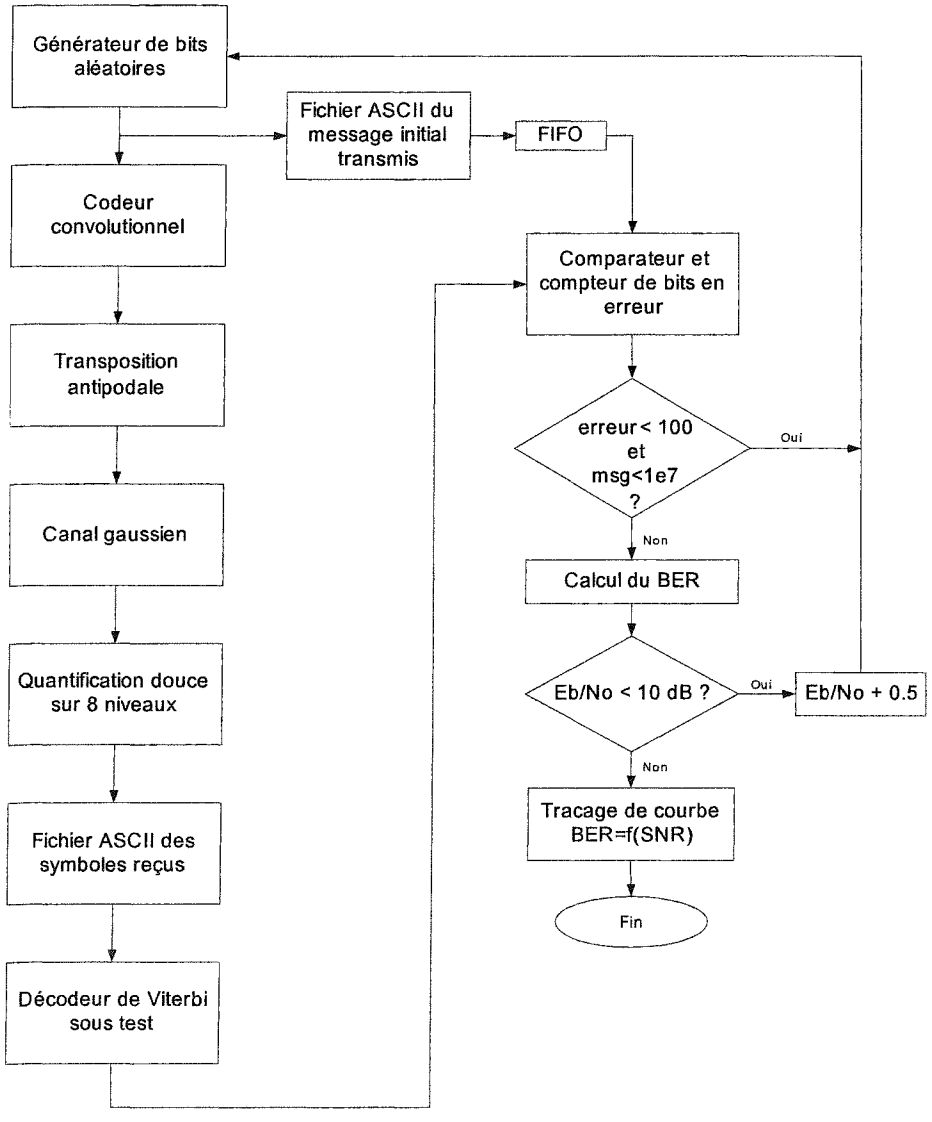


Figure 37 Illustration des procédures de test et prélèvement des résultats.

Le générateur de bits aléatoires, le codeur convolutionnel, la transposition antipodale, le bruit gaussien et la quantification douce sont réalisés par le programme de calculs mathématiques Matlab. Le résultat est placé dans un fichier ASCII. Il est sous forme de deux fichiers : un qui contient le message initial transmis et l'autre qui contient les symboles codés, bruités et quantifiés. Le décodeur de Viterbi reçoit ces symboles et fournit les bits décodés qui seront comparés avec les bits retardés du message initial. Si le nombre d'erreurs atteint 100 ou si le nombre de bits du message initial atteint 10^7, la simulation s'arrête pour le SNR donnée. Ces valeurs découlent du fait qu'en communication numérique au moins une centaine d'erreurs doit être observée avant qu'une mesure du BER soit valide. Le BER sera calculé par la formule (4.1). Ensuite, le SNR est incrémenté d'un pas de 0.5 dB. La simulation se répète pour la nouvelle valeur du SNR. Si ce dernier atteint la valeur de 10 dB la simulation est arrêtée et nous procédons au traçage de la courbe du BER = f(SNR).

4.5.3 Résultats de la simulation du code VHDL

La figure 40 montre la courbe du BER en fonction du SNR. Le décodeur de Viterbi testé ici est de mémoire de code m = 6, de taux R = 1/2, de polynômes 171_8 et 133_8, et de longueur de trace back L = 64. Pour pouvoir juger du gain de codage nous traçons aussi la courbe du BER du système non codé. On remarque un gain d'environ 5 dB pour un taux d'erreur de 10^{-5}. Pour le même code mais avec une longueur de trace back infinie, dans [4] on rapporte un gain de 5.1 dB au niveau d'un taux d'erreur de 10^{-5}.

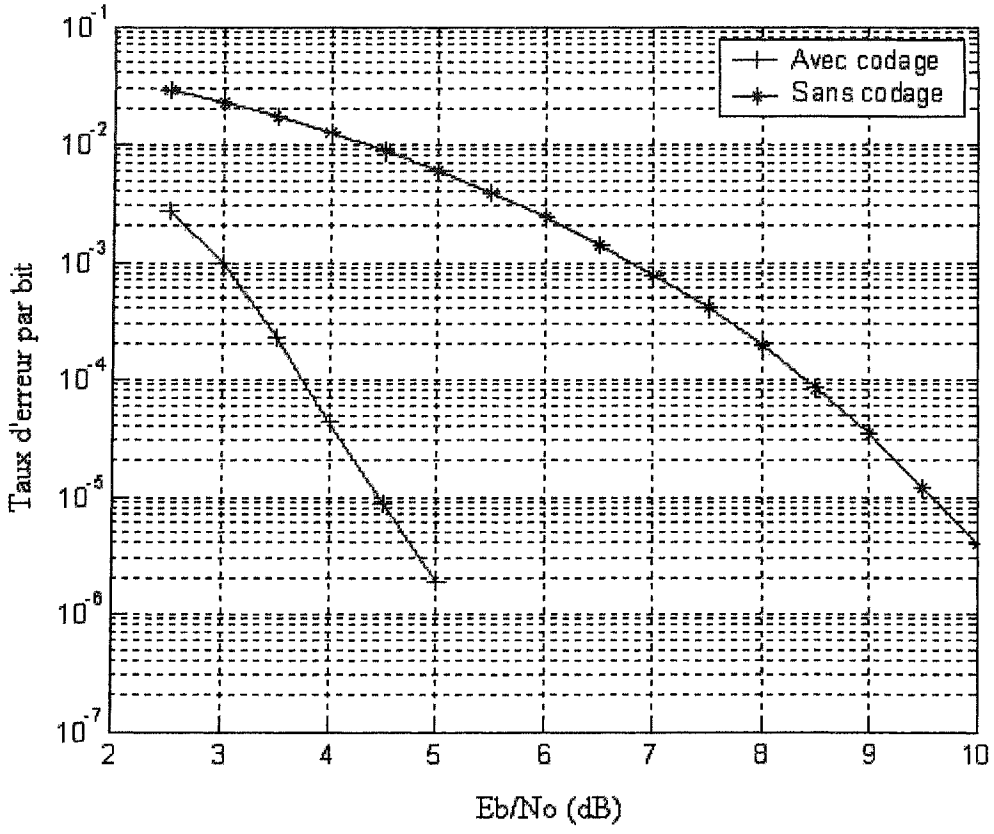


Figure 38 Courbe de performance d'erreur du décodeur de Viterbi pour le code de m = 6, R = 1/2, polynômes 171_8 et 133_8 et de L = 64

4.5.4 Test du design sur circuit FPGA

Le test du design sur circuit FPGA consiste à effectuer le test après la compilation et l'implémentation du design dans le circuit FPGA ciblé. Le but de ce test est de vérifier le fonctionnement réel du design. Un environnement de test (figure 37) est nécessaire à sa réalisation.

Le test a été réalisé en utilisant les logiciels Matlab et Simulink avec le "toolbox" "System generator" de Xilinx. Les vecteurs de test sont appliqués simultanément au

design réel ("Hardware in the loop") et à la simulation de son code VHDL importé sous forme de boite noire ("Black Box"). Ils sont créés dans l'environnement de travail ("workspace") de Matlab et lus par des blocs de Simulink. Le test réel nécessite l'ajout de certains éléments pour gérer le contrôle du design et pour configurer les tables de conversion du décodeur. La figure 41 montre un schéma bloc illustrant ces éléments additionnels. Ces derniers sont soit des blocs de Simulink, de Xilinx ou des boites noires de code VHDL conçu pour fin de test (Mémoires contenants les tables de conversion). La machine d'états finis lance les compteurs d'adresses des deux ROM ("Read Only Memory") en premier pour configurer les tables de conversions du décodeur. Après, elle initialise la lecture du message initial transmis et des symboles reçus créés précédemment dans le "workspace" par un programme Matlab.

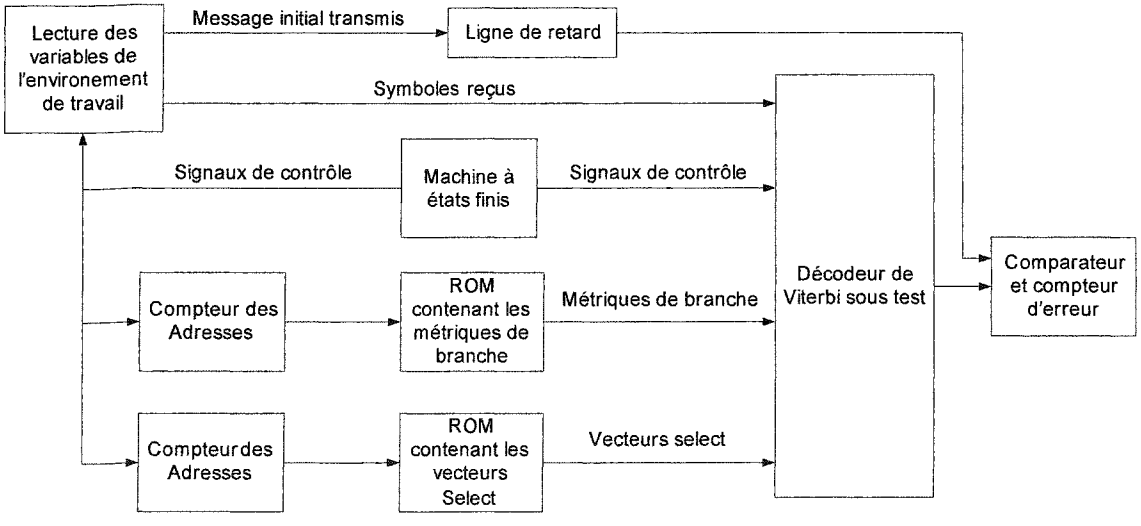


Figure 39 Schéma bloc illustrant les éléments additionnels nécessaires au test du design réel implémenté sur FPGA

4.5.5 Résultats du test du design sur circuit FPGA

Puisque le but ultime de ce test est de vérifier le fonctionnement réel du design, nous avons choisi de faire le test du décodeur de Viterbi pour une petite mémoire de code (m = 2). Cela réduit le temps alloué à la compilation, à la programmation du FPGA et au test. Le décodeur testé ici est de mémoire de code m = 2, de taux R = 1/2, de polynômes 7_8 et 5_8, et de longueur de trace back L = 16. Les courbes de performance d'erreur présentées à la figure 42 montrent bien que la courbe de la simulation du code VHDL coïncide parfaitement avec la courbe du test réel du design sur FPGA.

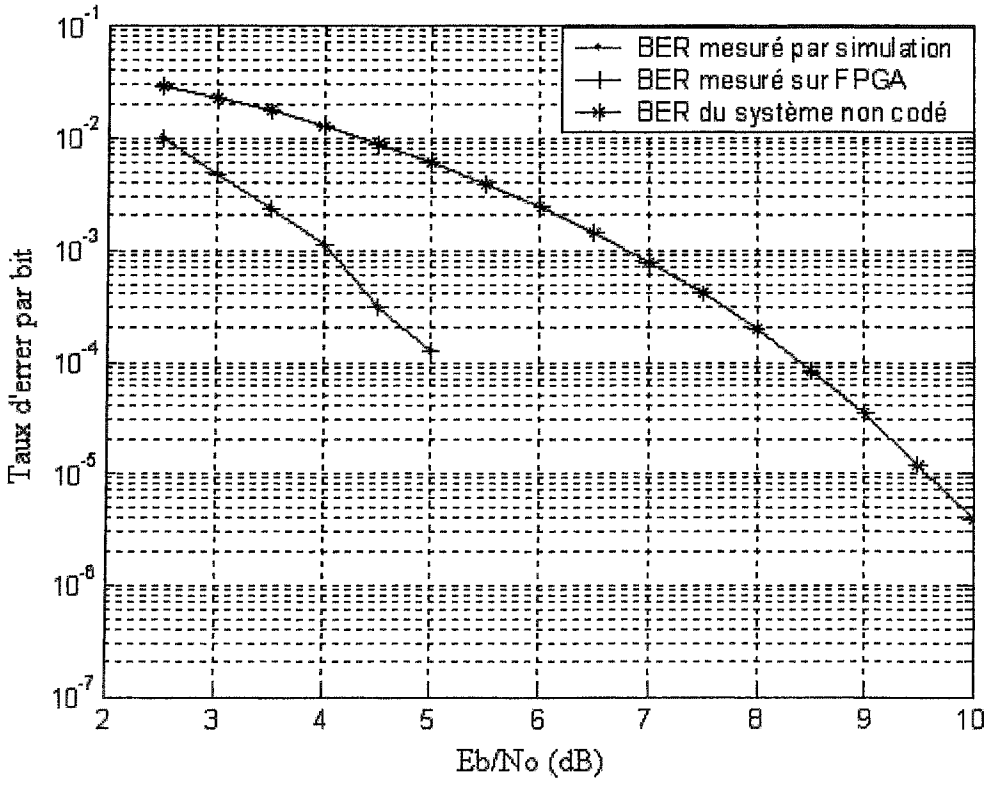


Figure 40 Courbes de performance d'erreur mesurées par simulation et par test réel sur circuit FPGA. m = 2, R = 1/2, polynômes 7_8 et 5_8 et de L = 16.

4.6 Comparaison avec un modèle existant

Nous avons choisi de comparer notre modèle du décodeur de Viterbi avec celui de la compagnie Xilinx dont la version V3 est la plus récente. Les caractéristiques de ce modèle sont disponibles sur le site internet de Xilinx [33]. La comparaison des caractéristiques des deux modèles est présentée au tableau X.

Tableau X

Comparaison des caractéristiques de notre modèle
du décodeur de Viterbi avec ceux du modèle de la compagnie Xilinx

Paramètre	Modèle de Xilinx	Notre Modèle
Mémoire de code : m	Paramétrable (2 à 8)	Paramétrable (2 et plus)
Polynôme générateur	Paramétrable	Flexible
Taux de code	Paramétrable 1/2 à 1/7	Flexible 1/S et 2/S
Longueur du trace back : L	Paramétrable multiple de 6, de 6 à 126	Paramétrable multiple de 2, Jusqu à 256 pour $m \geq 6$ (Pire cas)
Latence	4L	4L
Nombre de bits par symbole d'entrée (quantification douce)	Paramétrable 3 à 8	Flexible 2 et à g/2
Nombre de codes supportés par compilation	2 pour même m	Flexible
Types des métriques de branche	Hamming si quantification dure et euclidienne si quantification douce	Programmable
Largeur du mot de métrique de branche	Fixe	Paramétrable
Largeur du mot de métrique de chemin	Fixe	Paramétrable

Le tableau XI présente la comparaison de la complexité, de la fréquence et du débit de notre modèle, en version non flexible, avec celui de Xilinx [33]. Le modèle de Xilinx utilise la technique conventionnelle radix-2. Cette comparaison est faite pour deux différentes implémentations, m = 4 et m = 6, ciblant un FPGA Virtex-II, le XC2V3000 avec un grade de vitesse de -6. La comparaison avec le modèle de Xilinx dans le contexte d'une implémentation FPGA reste à titre indicatif seulement. En effet, la complexité de notre design est aux alentours du double de celle d'un design utilisant la technique radix-2. Les interconnections sont donc aussi aux alentours du double de ceux du design de référence. Il est normale que l'outil de placement et routage, PAR, ne donne pas d'aussi bons résultats d'implémentation pour de tel design comparativement à un design d'une complexité de 50 % de moins, tout en utilisant le même FPGA. L'ampleur de la différence de ces résultats croît très rapidement avec m à cause de la nature exponentielle de la croissance de la complexité du décodeur de Viterbi. D'autre part, la comparaison avec le modèle de Xilinx n'est pas tout à fait réaliste puisque nous n'avons pas tous les détails de son architecture interne (la normalisation des métriques d'état, les largeurs des mots utilisés pour les métriques de branche et pour les métriques d'état, ect...). Ajoutons à cela que les résultats de l'implémentation de Xilinx sont fournis par cette dernière comme guide [33]. Xilinx affirme qu'ils peuvent varier selon la version de l'outil d'implémentation utilisé et les options d'implémentation choisies [33]. Néanmoins, si nous considérons les résultats du tableau XI, nous pouvons conclure que nous avons atteint une efficacité de surface moyenne de 0.88. Nous avons atteint une augmentation moyenne du débit de 89 %. Ce résultat est nettement supérieur aux augmentations de débit réalisées dans le cadre des implémentations ASIC dans plusieurs références qui montrent les avantages de l'utilisation de la technique radix-4. L'implémentation décrite en [16] affirme avoir réalisé une augmentation du débit de 70 % pour m = 5 dans le cadre d'une implémentation ASIC de technologie CMOS 1.2 µm. L'implémentation décrite en [20] prétend avoir atteint une augmentation du débit de 71 % pour m = 6 pour une implémentation ASIC de technologie CMOS 0.35 µm.

Tableau XI

Comparaison de la complexité, de la fréquence et du débit de notre modèle (version non flexible) avec celui de Xilinx.

Paramètres	Modèle	Complexité (tranches)	Nombre de blocs RAM	Fréquence (Mhz)	Débit (Mbps)
m = 4 L = 30 R = ½ Qt = 8	Xilinx	452	2	163	163
	Le nôtre	870	4	159	318
	Facteur d'augmentation	1.93	2	0.98	1.95
m = 6 L = 42 R = ½ Qt = 8	Xilinx	1496	4	152	152
	Le nôtre	3623	8	138	276
	Facteur d'augmentation	2.42	2	0.91	1.82

Qt est le nombre de niveaux de quantification du signal d'entrée.

4.7 Conclusion

L'optimisation du code VHDL du design, sa simulation et le test de son implémentation sur un circuit FPGA ont été accomplies dans ce chapitre. L'adaptation du style de codage VHDL a permis d'optimiser notre design et de profiter des particularités de la

technologie FPGA. La simulation du code VHDL du décodeur montre un gain d'environ 5 dB pour un taux d'erreur de 10^{-5} par rapport à un système non codé. Ce gain est réalisé pour un décodeur de Viterbi de mémoire de code m = 6, de taux R = 1/2, de polynômes 171_8 et 133_8, d'une largeur du symbole d'entrée de 3 bits et de longueur de trace back L = 64. Cela est cohérent avec les courbes de performance que l'on trouve dans la plupart des références [30].

L'implémentation et le test de notre design sur un circuit FPGA montre qu'il réalise les fonctions requises avec les performances demandées.

CHAPITRE 5

CONCLUSION ET RECOMMANDATION

Les applications du décodeur de Viterbi sont nombreuses et différentes engendrant ainsi une grande nécessité de contrôler ses paramètres afin de pouvoir mieux l'adapter à des circonstances données. D'autre part, il n'y a pas de règles claires indiquant les valeurs optimales de ces paramètres. L'objectif visé par ce mémoire est la conception et la mise en œuvre d'un modèle VHDL synthétisable d'un décodeur de Viterbi de hautes performances.

L'architecture de ce décodeur de Viterbi se base sur l'utilisation du treillis radix-4 au lieu du treillis traditionnel radix-2. Cela nous permet ainsi de doubler le débit du décodeur au prix d'une augmentation quasi linéaire de la complexité de celui-ci. Ce débit double est obtenu tout en fonctionnant à la même fréquence qu'un décodeur de Viterbi d'architecture similaire utilisant le treillis de référence radix-2. Ceci peut être attrayant pour des implémentations sur des circuits moins rapides.

Une bonne adaptation du style de codage VHDL ciblant les circuits FPGA a contribué à l'optimisation du code du modèle profitant ainsi des différentes ressources disponibles dans un FPGA de type Virtex-II.

Des simulations du code VHDL ont montré que nous avons atteint nos objectifs en performances d'erreur. Un gain de codage de l'ordre de 5 dB a été atteint pour un décodeur de longueur de contrainte K = 7, de taux de codage R = 1/2, de polynômes générateurs P1 = 171_8 et P2 = 133_8 et de longueur du Trace Back L = 64. Aussi, des taux de codage de 1/3 et de 2/3 ont été simulés. Une implémentation sur FPGA nous a permis de confirmer les mesures de performance d'erreur de la simulation du code VHDL.

Cette implémentation nous a permis aussi de confirmer les performances de vitesse et de la complexité.

En plus d'avoir des paramètres génériques, le décodeur de Viterbi généré par notre modèle est flexible. Nous avons vérifié, par la simulation du code VHDL, qu'un décodeur compilé pour une mémoire de code donnée m peut réaliser le décodage des codes convolutionnels de mémoire de code m_c tel que $m_c \leq m$ avec les polynômes générateurs appropriés.

Le module des BMU est flexible et les métriques de branche sont implémentées sous forme de table de conversion. Cela permet au décodeur de réaliser deux groupes de taux de codage pour la même compilation: *1/S* et *2/S*, où S est le nombre de symboles d'entrée du décodeur. Nous avons vérifié, par la simulation du code VHDL, qu'un décodeur compilé pour travailler à un taux de codage de *1/S* peut réaliser tous les taux de codage de $1/S_p$ et de $2/S_p$ tel que $S_p \leq S$.

5.1 Travaux futurs

La logique nécessaire à la flexibilité du décodeur de Viterbi généré par notre modèle occupe une grande partie de la surface destinée au décodeur. En plus, cette logique additionnelle réduit la vitesse du décodeur. Il nous apparaît pertinent de travailler sur la réduction et l'optimisation de cette logique, afin d'améliorer les performances (vitesse et surface) du décodeur.

Aussi, il sera intéressant de faire les modifications nécessaires sur notre "core" afin d'implémenter l'algorithme de Viterbi à décision douce à la sortie, SOVA, ("Soft Output Viterbi Algorithm"). Cet algorithme est utilisé dans des applications avec codage concaténé ou itératif. Le lecteur désirant en savoir plus est prié de se référer aux ouvrages [7-11].

ANNEXE 1

Génération des vecteurs de contrôle

Les vecteurs de contrôle des multiplexeurs doivent êtres préalablement calculés puis sauvegardés dans les registres du décodeur conçus pour cette fin.

Pour faciliter la compréhension de la méthode de calcul de ces vecteurs de contrôle, on prend l'exemple d'un treillis avec un taux de R=1/2.

La métrique de branche est une mesure de la distance entre le symbole reçu et celui supposé avoir été émis. Le symbole reçu vient lire dans la table des métriques de branche le vecteur correspondant. Ce vecteur contient les quatre possibilités de métrique de branche. Pour une branche donnée, l'adresse de la métrique correspondante à ce vecteur est le symbole de deux bits qui étiquette cette branche du treillis. Ce symbole est le résultat du passage d'un état de cette branche à l'autre. Donc pour trouver les vecteurs de contrôle des multiplexeurs, nous imitons le fonctionnement du codeur convolutionnel avec les polynômes voulus. La sortie de ce codeur est le symbole formant l'adresse de la métrique de branche correspondante à la branche par laquelle passe ce codeur. Si on combine deux symboles, résultants du passage d'un état à un autre en radix-4, on obtient les quatre bits d'adresse nécessaires pour sélectionner la métrique de branche radix-4 correcte parmi les 16 métriques de branche fournies par le module BMU. La figure A.1 démontre l'exemple de passage de l'état 0 à tous les autres pour N=8, R=1/2 et les polynômes p1=1011 et p2=1101. Chaque branche est étiquetée avec les deux bits nécessaires pour le passage de l'état 0 à un état donné, et l'adresse de la métrique de branche correspondante à ce passage.

Le schéma fonctionnel d'une réalisation de la méthode de calcul des vecteurs de contrôle est présenté à la figure A.2. À partir de la partie du treillis qui représente les états possibles obtenus après le départ d'un état connu, on déduit la séquence des bits à donner au codeur convolutionnel pour qu'il parte d'un état donné, et arrive à tous les quatre autres états successivement. Par exemple pour passer de l'état Si à l'état S_{4i+1} il faut entrer au codeur un bit '0' suivi d'un bit '1'. Donc pour passer d'un état donné à

tous les autres par ordre croissant, nous devons introduire au codeur la séquence suivante : 00 01 10 11 du MSB au LSB. À la sortie du codeur nous obtenons les symboles non combinés pour les taux de 2/S. Les symboles combinés pour les taux de 1/S sont obtenus à l'aide d'une interface série/parallèle qui combine deux résultats successifs du codeur. Le bloc générateur d'état permet, chaque fois que le codeur convolutionnel change d'état, après chaque deux bits d'entrée, de le forcer à l'état de départ donné. Le compteur d'adresse sert à générer les adresses de 0 à 4*N-1 pour adresser les registres présents dans le bloc à registres.

La durée de cette période de programmation est variable, et dépend du nombre d'état du décodeur. Elle est égale à Nx4 car il y a quatre multiplexeurs par état.

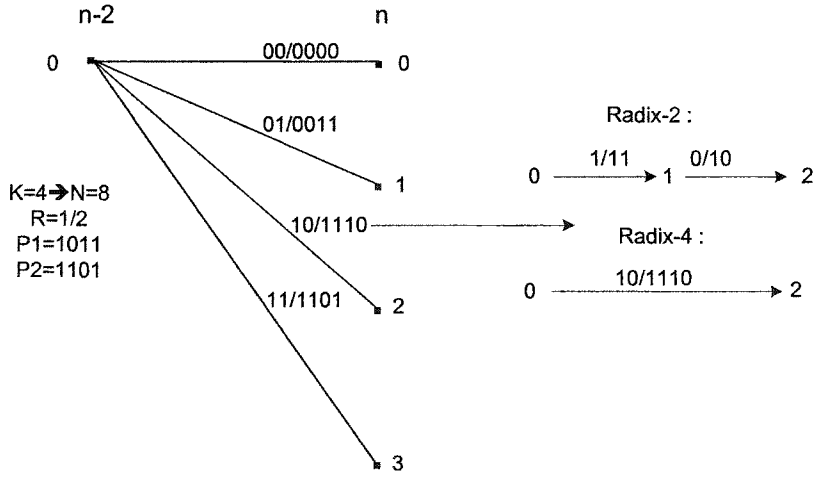


Figure A-1 Passage d'un état aux quatre autres, N = 8, P1 = 1011, P2 = 1101 et R = 1/2

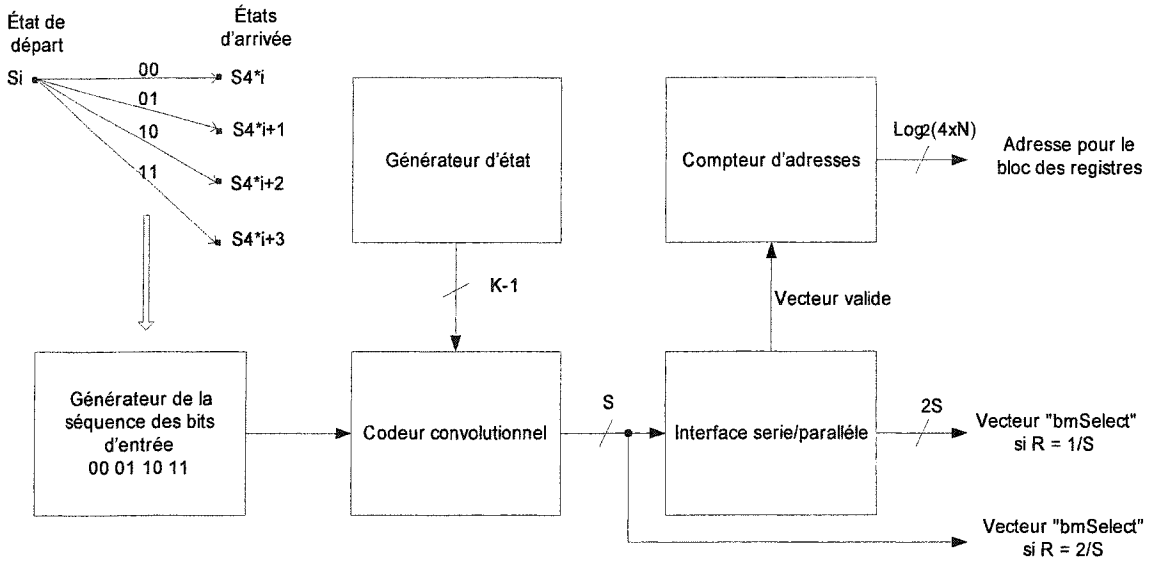


Figure A-2 Schéma fonctionnel pour la méthode de la génération des vecteurs "select"

ANNEXE 2

Résultats de synthèse du décodeur de Viterbi et de ces différentes unités

Rapport 1 Cellule ACS utilisant une description de flot de donnée

```
=========================================================================
*                            Final Report
*
=========================================================================
Final Results
RTL Top Level Output File Name     : acs4_1.ngr
Top Level Output File Name         : acs4_1
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 65

Macro Statistics :
# Registers                        : 10
#      2-bit register              : 1
#      5-bit register              : 4
#      8-bit register              : 5
# Multiplexers                     : 1
#      8-bit 4-to-1 multiplexer    : 1
# Xors                             : 26
#      1-bit xor3                  : 26

Cell Usage :
# BELS                             : 170
#      LUT2                        : 9
#      LUT2_D                      : 2
#      LUT2_L                      : 4
#      LUT3                        : 22
#      LUT3_D                      : 3
#      LUT3_L                      : 10
#      LUT4                        : 79
#      LUT4_D                      : 23
#      LUT4_L                      : 10
#      MUXF5                       : 8
# FlipFlops/Latches                : 62
#      FDE                         : 2
#      FDRE                        : 60
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 2v6000ff1152-5

 Number of Slices:                       93  out of  33792     0%
 Number of Slice Flip Flops:             62  out of  67584     0%
 Number of 4 input LUTs:                162  out of  67584     0%

=========================================================================
```

```
TIMING REPORT

Timing Summary:
---------------
Speed Grade: -5

   Minimum period: 9.673ns (Maximum Frequency: 103.381MHz)
   Minimum input arrival time before clock: 1.200ns
   Maximum output required time after clock: 0.494ns
   Maximum combinational path delay: No path found
```

Rapport 2 Cellule ACS utilisant une description comportementale

```
=========================================================================
*                            Final Report
*
=========================================================================
Final Results
RTL Top Level Output File Name     : acs4_v2_np.ngr
Top Level Output File Name         : acs4_v2_np
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 65

Macro Statistics :
# Registers                        : 10
#      2-bit register              : 1
#      5-bit register              : 4
#      8-bit register              : 5
# Multiplexers                     : 1
#      8-bit 4-to-1 multiplexer    : 1
# Adders/Subtractors               : 4
#      8-bit adder                 : 4
# Comparators                      : 6
#      7-bit comparator greater    : 6
# Xors                             : 6
#      1-bit xor3                  : 6

Cell Usage :
# BELS                             : 206
#      GND                         : 1
#      LUT1_L                      : 8
#      LUT2                        : 2
#      LUT2_D                      : 5
#      LUT2_L                      : 53
#      LUT3                        : 19
#      LUT4                        : 10
#      LUT4_D                      : 2
#      MUXCY                       : 70
#      MUXF5                       : 8
#      XORCY                       : 28
# FlipFlops/Latches                : 62
#      FDE                         : 2
#      FDRE                        : 60
=========================================================================
```

```
Device utilization summary:
---------------------------

Selected Device : 2v6000ff1152-5

 Number of Slices:                        81  out of  33792     0%
 Number of Slice Flip Flops:              62  out of  67584     0%
 Number of 4 input LUTs:                  99  out of  67584     0%


=========================================================================

TIMING REPORT

Timing Summary:
---------------
Speed Grade: -5

   Minimum period: 8.974ns (Maximum Frequency: 111.433MHz)
   Minimum input arrival time before clock: 1.200ns
   Maximum output required time after clock: 0.494ns
   Maximum combinational path delay: No path found
```

Rapport 3 Occupation en surface de l'unité LIFO utilisant des registres

```
=========================================================================
*                            Final Report
*
=========================================================================
Final Results
RTL Top Level Output File Name     : lifo_block.ngr
Top Level Output File Name         : lifo_block
Output Format                      : NGC
Optimization Goal                  : Area
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 8

Macro Statistics :
# Registers                        : 36
#      1-bit register              : 2
#      2-bit register              : 34
# Multiplexers                     : 33
#      2-to-1 multiplexer          : 33

Cell Usage :
# BELS                             : 68
#      LUT2                        : 2
#      LUT3                        : 66
# FlipFlops/Latches                : 70
#      FD                          : 1
#      FDE                         : 69
# Clock Buffers                    : 1
#      BUFGP                       : 1
# IO Buffers                       : 7
#      IBUF                        : 4
#      OBUF                        : 3
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 2v6000ff1152-5

 Number of Slices:                       37  out of  33792     0%
 Number of Slice Flip Flops:             70  out of  67584     0%
 Number of 4 input LUTs:                 68  out of  67584     0%
 Number of bonded IOBs:                   7  out of    824     0%
 Number of GCLKs:                         1  out of     16     6%
```

Rapport 4 Occupation en surface de l'unité LIFO utilisant des blocs RAM

```
=========================================================================
*                            Final Report
*
=========================================================================
Final Results
RTL Top Level Output File Name     : lifo_block_v2.ngr
Top Level Output File Name         : lifo_block_v2
Output Format                      : NGC
Optimization Goal                  : Area
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 13

Macro Statistics :
# Registers                        : 4
#      1-bit register              : 3
#      2-bit register              : 1
# Multiplexers                     : 1
#      2-to-1 multiplexer          : 1

Cell Usage :
# BELS                             : 5
#      GND                         : 1
#      LUT1                        : 1
#      LUT3                        : 2
#      VCC                         : 1
# FlipFlops/Latches                : 5
#      FD                          : 2
#      FDE                         : 3
# RAMS                             : 1
#      RAMB16_S2_S2                : 1
# Clock Buffers                    : 1
#      BUFGP                       : 1
# IO Buffers                       : 12
#      IBUF                        : 9
#      OBUF                        : 3
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 2v6000bf957-5

 Number of Slices:                        3  out of  33792     0%
 Number of Slice Flip Flops:              5  out of  67584     0%
 Number of 4 input LUTs:                  3  out of  67584     0%
 Number of bonded IOBs:                  12  out of    684     1%
 Number of BRAMs:                         1  out of    144     0%
 Number of GCLKs:                         1  out of     16     6%
```

Rapport 5 Résultats de synthèse de l'unité BMU

```
=========================================================================
*                            Final Report
*
=========================================================================
Final Results
RTL Top Level Output File Name     : bmu.ngr
Top Level Output File Name         : bmu
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 108

Macro Statistics :
# RAM                              : 1
#      64x16-bit single-port block RAM: 1
# Registers                        : 23
#      1-bit register              : 3
#      4-bit register              : 4
#      5-bit register              : 16
# Multiplexers                     : 5
#      2-to-1 multiplexer          : 5
# Adders/Subtractors               : 16
#      4-bit adder carry out       : 16

Cell Usage :
# BELS                             : 203
#      GND                         : 1
#      LUT1                        : 1
#      LUT2                        : 4
#      LUT2_L                      : 64
#      LUT3                        : 1
#      LUT3_L                      : 4
#      LUT4                        : 4
#      LUT4_L                      : 12
#      MUXCY                       : 64
#      XORCY                       : 48
# FlipFlops/Latches                : 99
#      FD                          : 1
#      FDE                         : 97
#      FDR                         : 1
# RAMS                             : 1
#      RAMB16_S36                  : 1
# Clock Buffers                    : 1
#      BUFGP                       : 1
# IO Buffers                       : 107
#      IBUF                        : 26
#      OBUF                        : 81
=========================================================================
```

```
Device utilization summary:
---------------------------

Selected Device : 2v6000ff1152-5

 Number of Slices:                       66  out of  33792     0%
 Number of Slice Flip Flops:             99  out of  67584     0%
 Number of 4 input LUTs:                 90  out of  67584     0%
 Number of bonded IOBs:                 107  out of    824    12%
 Number of BRAMs:                         1  out of    144     0%
 Number of GCLKs:                         1  out of     16     6%


=========================================================================
TIMING REPORT

Timing Summary:
---------------
Speed Grade: -5

   Minimum period: 5.944ns (Maximum Frequency: 168.237MHz)
   Minimum input arrival time before clock: 3.237ns
   Maximum output required time after clock: 4.840ns
   Maximum combinational path delay: No path found
```

Rapport 6 Résultats de synthèse de l'unité ACSU

```
=========================================================================
*                            Final Report
*
=========================================================================
Final Results
RTL Top Level Output File Name     : acs_top_v2_np.ngr
Top Level Output File Name         : acs_top_v2_np
Output Format                      : NGC
Optimization Goal                  : Area
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 225

Macro Statistics :
# Registers                        : 385
#      1-bit register              : 1
#      2-bit register              : 64
#      4-bit register              : 256
#      8-bit register              : 64
# Multiplexers                     : 320
#      5-bit 16-to-1 multiplexer   : 256
#      8-bit 4-to-1 multiplexer    : 64
# Adders/Subtractors               : 256
#      8-bit adder                 : 256
# Comparators                      : 384
#      7-bit comparator greater    : 384
# Xors                             : 384
#      1-bit xor3                  : 384

Cell Usage :
# BELS                             : 32677
#      GND                         : 1
#      LUT1                        : 768
#      LUT2                        : 3908
#      LUT3                        : 11456
#      LUT4                        : 672
#      MUXCY                       : 4480
#      MUXF5                       : 5760
#      MUXF6                       : 2560
#      MUXF7                       : 1280
#      XORCY                       : 1792
# FlipFlops/Latches                : 1665
#      FD                          : 1
#      FDE                         : 1152
#      FDRE                        : 512
# Clock Buffers                    : 1
#      BUFGP                       : 1
# IO Buffers                       : 224
#      IBUF                        : 95
```

```
#      OBUF                          : 129
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 2v6000ff1152-5

 Number of Slices:                     8917  out of  33792    26%
 Number of Slice Flip Flops:           1665  out of  67584     2%
 Number of 4 input LUTs:              16804  out of  67584    24%
 Number of bonded IOBs:                 224  out of    824    27%
 Number of GCLKs:                         1  out of     16     6%


=========================================================================
TIMING REPORT

Timing Summary:
---------------
Speed Grade: -5

   Minimum period: 11.866ns (Maximum Frequency: 84.276MHz)
   Minimum input arrival time before clock: 12.270ns
   Maximum output required time after clock: 4.840ns
   Maximum combinational path delay: No path found
```

Rapport 7 Résultats de synthèse de l'unité TBU

```
=========================================================================
*                            Final Report
*
=========================================================================
Final Results
RTL Top Level Output File Name     : trace_back.ngr
Top Level Output File Name         : trace_back
Output Format                      : NGC
Optimization Goal                  : Area
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 140

Macro Statistics :
# Registers                        : 22
#      1-bit register              : 6
#      2-bit register              : 7
#      5-bit register              : 5
#      6-bit register              : 4
# Multiplexers                     : 10
#      2-bit 4-to-1 multiplexer    : 1
#      2-bit 64-to-1 multiplexer   : 4
#      2-to-1 multiplexer          : 5
# Xors                             : 8
#      1-bit xor3                  : 8

Cell Usage :
# BELS                             : 556
#      GND                         : 1
#      LUT1                        : 3
#      LUT2                        : 7
#      LUT3                        : 296
#      LUT4                        : 6
#      MUXF5                       : 130
#      MUXF6                       : 64
#      MUXF7                       : 32
#      MUXF8                       : 16
#      VCC                         : 1
# FlipFlops/Latches                : 66
#      FD                          : 3
#      FDE                         : 56
#      FDRE                        : 7
# RAMS                             : 8
#      RAMB16_S36_S36              : 8
# Clock Buffers                    : 1
#      BUFGP                       : 1
# IO Buffers                       : 139
#      IBUF                        : 130
#      OBUF                        : 9
```

```
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 2v6000bf957-5

 Number of Slices:                      167  out of  33792     0%
 Number of Slice Flip Flops:             66  out of  67584     0%
 Number of 4 input LUTs:                312  out of  67584     0%
 Number of bonded IOBs:                 139  out of    684    20%
 Number of BRAMs:                         8  out of    144     5%
 Number of GCLKs:                         1  out of     16     6%


=========================================================================
TIMING REPORT


Timing Summary:
---------------
Speed Grade: -5

   Minimum period: 6.784ns (Maximum Frequency: 147.406MHz)
   Minimum input arrival time before clock: 2.915ns
   Maximum output required time after clock: 4.840ns
   Maximum combinational path delay: No path found
```

Rapport 8 Résultats de synthèse de l'unité LIFO

```
=========================================================================
*                            Final Report
*
=========================================================================
Final Results
RTL Top Level Output File Name     : lifo_block_v2.ngr
Top Level Output File Name         : lifo_block_v2
Output Format                      : NGC
Optimization Goal                  : Area
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 13

Macro Statistics :
# Registers                        : 4
#      1-bit register              : 3
#      2-bit register              : 1
# Multiplexers                     : 1
#      2-to-1 multiplexer          : 1

Cell Usage :
# BELS                             : 5
#      GND                         : 1
#      LUT1                        : 1
#      LUT3                        : 2
#      VCC                         : 1
# FlipFlops/Latches                : 5
#      FD                          : 2
#      FDE                         : 3
# RAMS                             : 1
#      RAMB16_S2_S2                : 1
# Clock Buffers                    : 1
#      BUFGP                       : 1
# IO Buffers                       : 12
#      IBUF                        : 9
#      OBUF                        : 3
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 2v6000bf957-5

 Number of Slices:                        3  out of  33792     0%
 Number of Slice Flip Flops:              5  out of  67584     0%
 Number of 4 input LUTs:                  3  out of  67584     0%
 Number of bonded IOBs:                  12  out of    684     1%
 Number of BRAMs:                         1  out of    144     0%
 Number of GCLKs:                         1  out of     16     6%
```

```
================================================================================

TIMING REPORT


Timing Summary:
---------------
Speed Grade: -5

   Minimum period: 3.414ns (Maximum Frequency: 292.912MHz)
   Minimum input arrival time before clock: 2.804ns
   Maximum output required time after clock: 4.840ns
   Maximum combinational path delay: No path found
```

Rapport 9 Résultats de synthèse de l'unité ACSU non flexible

```
=========================================================================
*                            Final Report
*
=========================================================================
Final Results
RTL Top Level Output File Name     : acs_top1.ngr
Top Level Output File Name         : acs_top1
Output Format                      : NGC
Optimization Goal                  : Area
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 212

Macro Statistics :
# Registers                        : 129
#      1-bit register              : 1
#      2-bit register              : 64
#      8-bit register              : 64
# Multiplexers                     : 64
#      8-bit 4-to-1 multiplexer    : 64
# Adders/Subtractors               : 256
#      8-bit adder                 : 256
# Comparators                      : 384
#      7-bit comparator greater    : 384
# Xors                             : 384
#      1-bit xor3                  : 384

Cell Usage :
# BELS                             : 13185
#      GND                         : 1
#      LUT1                        : 768
#      LUT2                        : 3648
#      LUT3                        : 1216
#      LUT4                        : 640
#      MUXCY                       : 4480
#      MUXF5                       : 640
#      XORCY                       : 1792
# FlipFlops/Latches                : 641
#      FD                          : 1
#      FDE                         : 128
#      FDRE                        : 512
# Clock Buffers                    : 2
#      BUFGP                       : 2
# IO Buffers                       : 210
#      IBUF                        : 81
#      OBUF                        : 129
=========================================================================
```

```
Device utilization summary:
---------------------------

Selected Device : 2v6000ff1152-5

 Number of Slices:                     3568  out of  33792    10%
 Number of Slice Flip Flops:            641  out of  67584     0%
 Number of 4 input LUTs:               6272  out of  67584     9%
 Number of bonded IOBs:                 210  out of    824    25%
 Number of GCLKs:                         2  out of     16    12%
=========================================================================
TIMING REPORT

Timing Summary:
---------------
Speed Grade: -5

   Minimum period: 9.598ns (Maximum Frequency: 104.188MHz)
   Minimum input arrival time before clock: 9.971ns
   Maximum output required time after clock: 4.840ns
   Maximum combinational path delay: No path found
```

Rapport 10 Résultats de synthèse du décodeur sans amélioration de la vitesse

```
=========================================================================
*                            Final Report
*
=========================================================================
Final Results
RTL Top Level Output File Name     : decoder_for_synth.ngr
Top Level Output File Name         : decoder_for_synth
Output Format                      : NGC
Optimization Goal                  : Area
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 43

Macro Statistics :
# RAM                              : 1
#      64x16-bit single-port block RAM: 1
# Registers                        : 434
#      1-bit register              : 13
#      2-bit register              : 72
#      4-bit register              : 260
#      5-bit register              : 21
#      6-bit register              : 4
#      8-bit register              : 64
# Multiplexers                     : 336
#      2-bit 4-to-1 multiplexer    : 1
#      2-bit 64-to-1 multiplexer   : 4
#      2-to-1 multiplexer          : 11
#      5-bit 16-to-1 multiplexer   : 256
#      8-bit 4-to-1 multiplexer    : 64
# Adders/Subtractors               : 272
#      4-bit adder carry out       : 16
#      8-bit adder                 : 256
# Comparators                      : 384
#      7-bit comparator greater    : 384
# Xors                             : 392
#      1-bit xor3                  : 392

Cell Usage :
# BELS                             : 33441
#      GND                         : 1
#      LUT1                        : 773
#      LUT2                        : 3920
#      LUT3                        : 11696
#      LUT4                        : 824
#      MUXCY                       : 4544
#      MUXF5                       : 5890
#      MUXF6                       : 2624
#      MUXF7                       : 1312
#      MUXF8                       : 16
```

```
#      VCC                         : 1
#      XORCY                       : 1840
# FlipFlops/Latches                : 1835
#      FD                          : 7
#      FDE                         : 1308
#      FDR                         : 1
#      FDRE                        : 519
# RAMS                             : 10
#      RAMB16_S2_S2                : 1
#      RAMB16_S36                  : 1
#      RAMB16_S36_S36              : 8
# Clock Buffers                    : 1
#      BUFGP                       : 1
# IO Buffers                       : 42
#      IBUF                        : 39
#      OBUF                        : 3
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 2v6000bf957-5

 Number of Slices:                     9151  out of  33792    27%
 Number of Slice Flip Flops:           1835  out of  67584     2%
 Number of 4 input LUTs:              17213  out of  67584    25%
 Number of bonded IOBs:                  42  out of    684     6%
 Number of BRAMs:                        10  out of    144     6%
 Number of GCLKs:                         1  out of     16     6%


=========================================================================

TIMING REPORT


Timing Summary:
---------------
Speed Grade: -5

   Minimum period: 12.046ns (Maximum Frequency: 83.015MHz)
   Minimum input arrival time before clock: 4.892ns
   Maximum output required time after clock: 4.840ns
   Maximum combinational path delay: No path found
```

Rapport 11 Résultats de synthèse du décodeur avec pipeline

```
=========================================================================
*                            Final Report
*
=========================================================================
Final Results
RTL Top Level Output File Name     : decoder_for_synth.ngr
Top Level Output File Name         : decoder_for_synth
Output Format                      : NGC
Optimization Goal                  : Area
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 43

Macro Statistics :
# RAM                              : 1
#      64x16-bit single-port block RAM: 1
# Registers                        : 691
#      1-bit register              : 14
#      2-bit register              : 72
#      4-bit register              : 260
#      5-bit register              : 21
#      6-bit register              : 4
#      8-bit register              : 320
# Multiplexers                     : 336
#      2-bit 4-to-1 multiplexer    : 1
#      2-bit 64-to-1 multiplexer   : 4
#      2-to-1 multiplexer          : 11
#      5-bit 16-to-1 multiplexer   : 256
#      8-bit 4-to-1 multiplexer    : 64
# Adders/Subtractors               : 272
#      4-bit adder carry out       : 16
#      8-bit adder                 : 256
# Comparators                      : 384
#      7-bit comparator greater    : 384
# Xors                             : 392
#      1-bit xor3                  : 392

Cell Usage :
# BELS                             : 33443
#      BUF                         : 2
#      GND                         : 1
#      LUT1                        : 773
#      LUT2                        : 4304
#      LUT3                        : 11824
#      LUT4                        : 312
#      MUXCY                       : 4544
#      MUXF5                       : 5890
#      MUXF6                       : 2624
#      MUXF7                       : 1312
```

```
#      MUXF8                          : 16
#      VCC                            : 1
#      XORCY                          : 1840
# FlipFlops/Latches                   : 3886
#      FD                             : 10
#      FDE                            : 1308
#      FDR                            : 1
#      FDRE                           : 2567
# RAMS                                : 10
#      RAMB16_S2_S2                   : 1
#      RAMB16_S36                     : 1
#      RAMB16_S36_S36                 : 8
# Clock Buffers                       : 1
#      BUFGP                          : 1
# IO Buffers                          : 42
#      IBUF                           : 39
#      OBUF                           : 3
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 2v6000bf957-5

 Number of Slices:                     9155  out of  33792    27%
 Number of Slice Flip Flops:           3886  out of  67584     5%
 Number of 4 input LUTs:              17213  out of  67584    25%
 Number of bonded IOBs:                  42  out of    684     6%
 Number of BRAMs:                        10  out of    144     6%
 Number of GCLKs:                         1  out of     16     6%
=========================================================================

TIMING REPORT


Timing Summary:
---------------
Speed Grade: -5

   Minimum period: 7.001ns (Maximum Frequency: 142.837MHz)
   Minimum input arrival time before clock: 4.892ns
   Maximum output required time after clock: 4.840ns
   Maximum combinational path delay: No path foud
```

Rapport 12 Résultats de synthèse du décodeur avec balancement des registres

```
=========================================================================
*                            Final Report
*
=========================================================================
Final Results
RTL Top Level Output File Name     : decoder_for_synth.ngr
Top Level Output File Name         : decoder_for_synth
Output Format                      : NGC
Optimization Goal                  : Area
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 43

Macro Statistics :
# RAM                              : 1
#      64x16-bit single-port block RAM: 1
# Registers                        : 627
#      1-bit register              : 14
#      2-bit register              : 72
#      4-bit register              : 260
#      5-bit register              : 21
#      6-bit register              : 4
#      8-bit register              : 256
# Multiplexers                     : 336
#      2-bit 4-to-1 multiplexer    : 1
#      2-bit 64-to-1 multiplexer   : 4
#      2-to-1 multiplexer          : 11
#      5-bit 16-to-1 multiplexer   : 256
#      8-bit 4-to-1 multiplexer    : 64
# Adders/Subtractors               : 272
#      4-bit adder carry out       : 16
#      8-bit adder                 : 256
# Comparators                      : 384
#      7-bit comparator greater    : 384
# Xors                             : 392
#      1-bit xor3                  : 392

Cell Usage :
# BELS                             : 33445
#      BUF                         : 4
#      GND                         : 1
#      LUT1                        : 773
#      LUT2                        : 4304
#      LUT3                        : 11824
#      LUT4                        : 312
#      MUXCY                       : 4544
#      MUXF5                       : 5890
#      MUXF6                       : 2624
#      MUXF7                       : 1312
```

```
#      MUXF8                          : 16
#      VCC                            : 1
#      XORCY                          : 1840
# FlipFlops/Latches                   : 3376
#      FD                             : 12
#      FDE                            : 1308
#      FDR                            : 1
#      FDRE                           : 2055
# RAMS                                : 10
#      RAMB16_S2_S2                   : 1
#      RAMB16_S36                     : 1
#      RAMB16_S36_S36                 : 8
# Clock Buffers                       : 1
#      BUFGP                          : 1
# IO Buffers                          : 42
#      IBUF                           : 39
#      OBUF                           : 3
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 2v6000bf957-5

 Number of Slices:                     9158  out of  33792    27%
 Number of Slice Flip Flops:           3376  out of  67584     4%
 Number of 4 input LUTs:              17213  out of  67584    25%
 Number of bonded IOBs:                  42  out of    684     6%
 Number of BRAMs:                        10  out of    144     6%
 Number of GCLKs:                         1  out of     16     6%


=========================================================================
TIMING REPORT


Timing Summary:
---------------
Speed Grade: -5

   Minimum period: 9.675ns (Maximum Frequency: 103.359MHz)
   Minimum input arrival time before clock: 4.892ns
   Maximum output required time after clock: 4.840ns
   Maximum combinational path delay: No path found
```

Rapport 13 Résultats de placement et routage du décodeur avec pipeline

```
C:/Xilinx/bin/nt/par.exe -w -intstyle ise -ol med -t 1
decoder_for_synth_map.ncd decoder_for_synth.ncd decoder_for_synth.pcf

Constraints file: decoder_for_synth.pcf

Loading device database for application Par from file
"decoder_for_synth_map.ncd".
   "decoder_for_synth" is an NCD, version 2.38, device xc2v6000,
package bf957,
speed -5
Loading device for application Par from file '2v6000.nph' in
environment
C:/Xilinx.

Device utilization summary:
---------------------------

   Number of External IOBs                    43 out of   684      6%
   Number of LOCed External IOBs               8 out of    43     18%
   Number of RAMB16s                          10 out of   144      6%
   Number of SLICEs                         9361 out of 33792     27%
   Number of BUFGMUXs                          1 out of    16      6%

Generating "par" statistics.

**************************
Generating Clock Report
**************************

+---------+----------+--------+--------+--------------+---------------+
|Clock Net| Resource | Locked | Fanout | Net Skew(ns) | Max Delay(ns) |
+---------+----------+--------+--------+--------------+---------------+
|clk_BUFGP| BUFGMUX4S|   No   | 2306   |     0.572    |     2.207     |
+-----------------+------------------+----------+----------+---------

   The Delay Summary Report

   The SCORE FOR THIS DESIGN is: 382


The NUMBER OF SIGNALS NOT COMPLETELY ROUTED for this design is: 0

   The AVERAGE CONNECTION DELAY for this design is:          2.003
   The MAXIMUM PIN DELAY IS:                                11.696
   The AVERAGE CONNECTION DELAY on the 10 WORST NETS is:     9.072
```

```
   Listing Pin Delays by value: (nsec)

d < 2.00  < d < 4.00  < d < 6.00  < d < 8.00  < d < 12.00  d >= 12.00
 -------   --------    --------    ---------   ---------   ---------
  40239     31485        4015         966         419          0

Timing Score: 0

Asterisk (*) preceding a constraint indicates it was not met.
   This may be due to a setup or hold violation.

--------------------------------------------------------------------------------
 Constraint                          |   Requested    |   Actual    | Logic
                                     |                |             | Levels
--------------------------------------------------------------------------------
  TS_clk = PERIOD TIMEGRP "clk" |    10.000ns    | 9.948ns     | 6
  10 nS   HI  GH 50.000000 %    |                |             |
--------------------------------------------------------------------------------


All constraints were met.

Data Sheet report:
-----------------
All values displayed in nanoseconds (ns)

Clock to Setup on destination clock clk
-------------------+-----------+-----------+-------------+---------+
                   | Src:Rise  | Src:Fall  | Src:Rise    | Src:Fall|
Source Clock       | Dest:Rise | Dest:Rise | Dest:Fall   |Dest:Fall|
-------------------+-----------+-----------+-------------+---------+
 clk               |   9.948   |           |             |         |
-------------------+-----------+-----------+-------------+---------+


Timing summary:
---------------

Timing errors: 0  Score: 0

Constraints cover 388754 paths, 0 nets, and 76281 connections

Design statistics:
   Minimum period:   9.948ns (Maximum frequency: 100.523MHz)


Analysis completed Mon Jul 12 12:37:08 2004
--------------------------------------------------------------------------------

Peak Memory Usage: 263 MB
```

Rapport 14 Résultats de placement et routage du décodeur avec balancement des registres

```
C:/Xilinx/bin/nt/par.exe -w -intstyle ise -ol med -t 1
decoder_for_synth_map.ncd decoder_for_synth.ncd decoder_for_synth.pcf


Constraints file: decoder_for_synth.pcf

Loading device database for application Par from file
"decoder_for_synth_map.ncd".
   "decoder_for_synth" is an NCD, version 2.38, device xc2v6000,
package bf957,
speed -5
Loading device for application Par from file '2v6000.nph' in
environment
C:/Xilinx.

Device utilization summary:

   Number of External IOBs           43 out of 684     6%
   Number of LOCed External IOBs     8 out of 43      18%
   Number of RAMB16s                 10 out of 144     6%
   Number of SLICEs                  9365 out of 33792  27%
   Number of BUFGMUXs                1 out of 16       6%

*************************
Generating Clock Report
*************************

+-----------------+----------+------+------+------------+-------------+
|Clock Net        | Resource |Locked|Fanout|Net Skew(ns)|Max Delay(ns)|
+-----------------+----------+------+------+------------+-------------+
|clk_BUFGP        | BUFGMUX5S| No   | 1793 |   0.568    |  2.203      |
+-----------------+----------+------+------+------------+-------------+

The Delay Summary Report

The SCORE FOR THIS DESIGN is: 362


The NUMBER OF SIGNALS NOT COMPLETELY ROUTED for this design is: 0

   The AVERAGE CONNECTION DELAY for this design is:        1.884
   The MAXIMUM PIN DELAY IS:                              10.959
   The AVERAGE CONNECTION DELAY on the 10 WORST NETS is:   8.670
```

```
   Listing Pin Delays by value: (nsec)

d < 2.00   < d < 4.00  < d < 6.00  < d < 8.00  < d < 11.00  d >= 11.00
---------  ---------  ---------  ---------  ---------  ---------
 46064       20911       6967        992        148          0

Timing Score: 0

Asterisk (*) preceding a constraint indicates it was not met.
   This may be due to a setup or hold violation.

--------------------------------------------------------------------------------
Constraint                              | Requested  | Actual     | Logic
                                        |            |            | Levels
--------------------------------------------------------------------------------
TS_clk = PERIOD TIMEGRP "clk"           | 12.340ns   | 12.340ns   | 11
14 nS   HI GH 50.000000 %               |            |            |
--------------------------------------------------------------------------------


All constraints were met.
Generating Pad Report.

All signals are completely routed.

Data Sheet report:
-----------------
All values displayed in nanoseconds (ns)

Clock to Setup on destination clock clk
---------------+---------+---------+---------+---------+
               | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
---------------+---------+---------+---------+---------+
clk            |   12.340|         |         |         |
---------------+---------+---------+---------+---------+


Timing summary:
---------------

Timing errors: 0  Score: 0

Constraints cover 2502802 paths, 0 nets, and 74239 connections

Design statistics:
   Minimum period:  12.340ns (Maximum frequency:  81.037MHz)


Analysis completed Mon Jul 12 17:15:13 2004
--------------------------------------------------------------------------------

Peak Memory Usage: 259 MB
```

BIBLIOGRAPHIE

[1] Elias, P. (1955). *Coding for Noisy Channels*. IRE Conv. Rec., Part 4, pp. 37-47.

[2] Viterbi, A. J. (April 1967). *Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm*. IEEE Trans. Inf. Theory, IT-13, pp. 260-269.

[3] Chadha, K., Cavallaro, J.R. (Nov. 2001). *A Reconfigurable Viterbi Decoder Architecture*. Signals, Systems and Computers. Conference Record of the Thirty-Fifth Asilomar Conference. Volume: 1 , 4-7 Pages:66 - 71 vol.1

[4] Proakis J. G. (2001). *Digital Communication*. 4th ed., New york, N.Y.: McGraw-Hill.

[5] Bhargava, Vijay K., Haccoun David, Matyas Robert, Nuspl Peter. (1981). *Digital Communication by satellite*. New York N.Y.: J. Wiley and Sons.

[6] Shu Lin, Daniel J., Costello Jr. (1983). *Error Control Coding : Fundamentals and Applications*. Englewood Cliffs, N.J.: Printice-Hall.

[7] Hagenaue,r J., Hoeher, P. (Nov. 1989). *A Viterbi algorithm with soft-decision outputs and its applications*. Global Telecommunications Conference, 1989, and Exhibition. 'Communications Technology for the 1990s and Beyond'. GLOBECOM '89. IEEE , 27-30 Page(s): 1680 -1686 vol.3.

[8] Berrou, C., Adde, P., Angui, E, Faudeil, S. (May 1993). *A low complexity soft-output Viterbi decoder architecture*. Communications, 1993. ICC 93. Geneva. Technical Program, Conference Record, IEEE International Conference. Volume: 2 , 23-26. pp: 737 -740 vol.2.

[9] Joeressen, O.J., Meyr, H.(July 1995). *A 40 Mb/s soft-output Viterbi decoder*. Solid-State Circuits, IEEE Journal. Volume: 30 Issue: 7, Page(s): 812 –818.

[10] Garrett, D., Stan, M. (Aug. 1998). *Low power architecture of the soft-output Viterbi algorithm*. Low Power Electronics and Design, 1998. Proceedings. International Symposium, 10-12 Page(s): 262 –267.0

[11] W.J., Gaudet, V.C., Gulak, P.G. (Oct. 2001). *Difference metric soft-output detection: architecture and implementation Gross*. Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions. Volume: 48 Issue: 10 , Page(s): 904 –911.

[12] Frenger, P., Orten, P., Ottosson, T. (Nov. 1999).*Convolutional codes with optimum distance spectrum*. Communications Letters, IEEE. Volume: 3 Issue: 11, pp: 317 –319.

[13] Viterbi, Andrew J., Omura, Jim K. (1979). *Principles of Digital Communication and coding* New york, N.Y.: McGraw-Hill.

[14] Viterbi, Andrew J. (1995). *CDMA: Principles of Spread Spectrum Communication*. Reading, Mass.: Addison-Wesley.

[15] Clark, George Cyril, Cain, J. Bibb. (1981). *Error-Correction Coding for Digital Communications*. New York, N.Y. : Plenum Press.

[16] Black, P.J., Meng, T.H.. (Dec. 1992). *A 140-Mb/s, 32-state, radix-4 Viterbi decoder*. Solid-State Circuits, IEEE Journal. Volume: 27, Issue: 12, pp:1877 – 1885

[17] Ta Lee, w.T., Ho chen, Gee Chen, L. (June 1995). *VLSI Architecture for Radix-2^k Viterbi Cecoding with Transpose Algorithm*. VLSI Technology, Systems, and Applications. Proceedings of Technical Papers. 1995 International Symposium. Pages:219 - 223

[18] Black, P., Meng, T. (June 1997). *A 1-Gb/s, four-state, sliding block Viterbi decoder*. Solid-State Circuits, IEEE Journal. Volume: 32 , Issue: 6 , pp:797 – 805

[19] Hekstra, A. P. (Nov. 1989). *An alternative to metric rescaling in Viterbi decoders*. IEEE Trans. Commun., volume 37, pp.1220-1222.

[20] Erik Paaske, Jacob Dahl Andersen. (June 1998). *High Speed Viterbi Decoder Architecture*. Fisrt ESA workship on tracking telemetry and command systems ESTEC.

[21] Cavallaro, J.R., Vaya, M. (April 2003). *Viturbo: a reconfigurable architecture for Viterbi and turbo decoding*. Acoustics, Speech, and Signal Processing, Proceedings.(ICASSP '03). 2003 IEEE International Conference. Volume: 2 6-10. pp:II - 497-500 vol.2

[22] Ranpara, S., Dong Sam, Ha. (1999). *A Low-power Viterbi Decoder design for wireless communications applications*. ASIC/SOC Conference, 1999. Proceedings. Twelfth Annual IEEE International , 15-18 Sept. 1999 pp:377 - 381

[23] Oliver Collins, M. (December 1992). *The Subtleties and Intricacies of building a Constraint Length.* 15 Convolutional Decoder", IEEE Transaction on communication Vol. 40, No. 12,

[24] Fettweis, G. Dawid, H., Meyr H. (Oct.1990). *high-Rate Viterbi Processor : A Systolic Array Solution.* Selected Areas in Communications, IEEE Journal. Volume: 8 , Issue: 8 , pp:1520 - 1534

[25] Fettweis, G., Dawid, H., Meyr, H. (1996). *A CMOS IC for Gb/s Viterbi Decoding : System Design and VLSI Implementation.* Very Large Scale Integration (VLSI) Systems, IEEE Transactions. Volume: 4 , Issue: 1 , March 1996 Pages:17 - 31

[26] Fettweis, G., Meyr, H. (1988). *Parallel Viterbi Decoding by breaking the compare-select feedback bottleneck.* Communications, 1988. ICC 88. Digital Technology-Spanning the Universe. Conference Record. IEEE International Conference. 12-15 June 1988 Pages:719 - 723 vol.2

[27] Xilinx. (dec 2000). *Virtex-II Platform FPGA Handbook.* (V1.0)

[28] Xilinx. *Virtex-II Platform FPGA's: Complet Data Sheet.* http://www.xilinx.com

[29] Xilinx. (June 2000). Xilinx Application Notes : *XAPP215.* http://www.xilinx.com

[30] Heller, J., Jacobs, I.(Oct 1971). *Viterbi Decoding for Satellite and Space Communication.* Communications, IEEE Transactions on [legacy, pre - 1988] Volume: 19 , Issue: 5 , Pages:835 – 848.

[31] Synplicity. Syndicated. Volume 1, issue 4. http://www.synplicity.com

[32] Synplicity. Synplicity Application Notes: *Using Retiming* http://www.synplicity.com.

[33] Xilinx. (Aug 2003). *Xilinx Viterbi Decoder IP Core V3.* http://www.xinlinx.com

[34] Shung, C.B., Siegel, P.H., Ungerboeck, G., Thapar, H.K., (Apr 1990). *VLSI Architectures for Metric Normalization in the Viterbi Algorithm.* Communications, 1990. ICC 90, Including Supercomm Technical Sessions. SUPERCOMM/ICC '90. Conference Record, IEEE International Conference on , 16-19, Pages:1723 - 1728 vol.4