REPUBLIQUE ALGERIENNE DEMOCRATIQUE & POPULAIRE
Ministère de l'enseignement Supérieur et de la Recherche Scientifique



*Université Dr. YAHIA FARES de Médéa*
*Faculté des Sciences et de la Technologie*
*Département du Génie Electrique et de l'Informatique*

**Projet de fin d'étude pour l'obtention du diplôme de mastère en génie électrique.**
**Option : Systèmes Intelligents et Communicants**

# Implémentation d'un décodeur déconvolutionnel de Viterbi sur DSP C5410

*Réalisé par :*

*Melle : ALLAOUI Fatima Zohra*

*Melle : HERICHE Fatima Zohra*

*Proposés et dirigés par :*

*promoteur Mr : BIRANE_A*

*Co-promoteur Dr : BATTEL. N*

*ANNEE UNIVERSITAIRE*

*2012/2013*

## *Remerciement*

*Nous tenons avant tout à remercier Dieu tout puissant de nous avoir donnée la force et la volonté pour achever ce modeste travail.*

*Nous tenons à remercier particulièrement nos parent ; notre succès demeure de loin le fuit de leur longue années de sacrifices et d'éducation.*

*Nos vits remerciement, à notre promoteur Dr-Batel Noureddine pour nous avoir proposé ce sujet et dirigé dans notre travail.*

*Nous remercions également les enseignants de l'université de YAHYA FARECE de formation en informatique et électronique pour tout le savoir qu'ils ont su nous transmettre durant ces cinq dernières années.*

*Nous ne pouvons nous empêcher d'avoir une pensée pour ceux et celles qui ont répondu présents et nous ont offert leur soutien morale dans les moments difficile et étaient à nos cotés pour partager avec nous les moments de joie.*

# Dédicaces

*Je dédie ce modeste travail aux être qu'ils me sont les plus chers au mode, ma mère et mon père et mon à qui je dois mon existence et succès que le Dieu le tout puissant les protèges.*

*A mes chère sœurs SABRINA, SOUAD, DJOUHAR et mes cher frères MOHAMED-AMINE, ABDE-ELKADER et sa femme NADJIYA sans oubliant mon oncle ABDE-REZZAKE et sa familles tout à son mon et mes tantes et touts les membres de familles ALLAOUI et REZZOUG.*

*A mon binôme FATIMA-ZOHRA et sa famille.*

*A tous mes amie KHADIDJA, NESRIN, GHANIA et touts les gent qui m'aiment et qui m'ont aidé.*

*A mon fiancé Hessen et toute sa famille.*

*«-- ALLAOUI FATIMA-ZOHRA-- »*



*Dédicaces*

*Je dédie ce modeste travail à*

*Mes très chères parents, pour leurs soutient moral et financier et d'être la lumière de ma vie, qui m'ont toujours encouragé pour terminer mes études, en leur espérant une longue vie et que Dieu les gardes.*

❖ *A mes frères et mes sœurs.*

❖ *A tous mes amis de près ou de loin sans exception.*

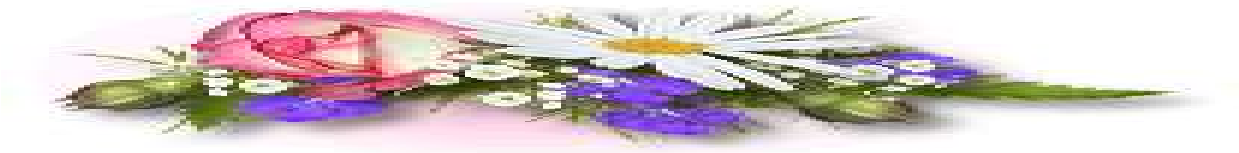
❖ *A mes chers amies de l'université de Médéa et surtout * KARIMA* et *SALIHA*NACERA*HAKIMA* ANISSA* THOURAYA* * FATIHA* et mon binôme *FATIMA ZOHRA * et sa famille.*

❖ *A mon promoteurs *Dr. ***BATTEL* * **

❖ *A tous les enseignants de faculté des sciences et de la technologie.*

❖ *A toute la promotion 2ème années Master sans exception*

***Heriche Fatima-Zohra***



## الملخص

الغرض من هذه الأطروحة هو تصحيح الأخطاء الذي يعد نظام قوي وناجح لتقوية أداء نظام الإرسال عبر قناة فوضوية. نظام التلافيف (convolutionnel) هو واحد من أهم الأنظمة الموجودة حاليا. المساحة المعمولة بها هي الشعرية (Treillis) و تنفيذه عبر فك الرمز يتم عموما عن طريق خوارزمية Viterbi باستعمال DSP من عائلة C4510.



## Abstract

The purpose of this study is the error correction which is a powerful method for increasing the performance of system transmission through a noisy channel. The convolutional coding is one of the best existing methods in practice. The state space is then the trellis, and his decoding is generally entrusted to the Viterbi algorithm. Its implementation is performed on a fixed-point DSP from TI family, C5410.



## Résumé

Le but de ce travail est la correction d'erreur qui est une méthode puissante pour augmenter la performance d'un système de transmission à travers un canal bruité. Le codage convolutionnel figure parmi les meilleures méthodes existantes en pratique. L'espace d'état est alors un treillis, et son décodage est confié généralement à l'algorithme de Viterbi. Son implémentation est à effectuer sur un DSP à virgule fixe de la famille TI, le C5410.

# SOMMAIRE

Table des matières:


RESUME ..... 5

INTRODUCTION ..... 11

I.PRESENTATION DES CODES CONVOLUTIONNEL..... 12

I.1L'ENCODAGE CONVOLUTIONNEL: ..... 13
I.2 VECTEUR DE CONNEXION : ..... 14
I.3 REPONSE IMPULSIONNELLE: ..... 16
I.4 REPRESENTATION POLYNOMIALE : ..... 17
I.5 LE DIAGRAMME D'ETAT: ..... 18
I.6 LE DIAGRAMME EN TREILLIS: ..... 20

II.ALGORITHME DE DECODAGE CONVOLUTIONNEL ..... 22

II. METHODES DU DECODAGE CONVOLUTIONNEL : ..... 23
II.1 DECODAGE PAR MAXIMUM DE VRAISEMBLANCE : ..... 23
II.1.2 Modèles de récepteurs à décisions soft et hard: ..... 25
II.1.3 L'algorithme du décodage convolutionnel de Viterbi: ..... 27
II.1.4 Exemple du décodage convolutionnel de Viterbi : ..... 28
II.1.5 Implémentation de décodeur : ..... 33
II.1.6 Calcul par addition-comparaison-sélection (ACS): ..... 34
II.1.7 ACS mené sur le treillis: ..... 35
II.1.8 L'historique de chemin : ..... 36
II.2 LES PROPRIETES DES CODES CONVOLUTIONNELS : ..... 37
II.2.1 Propriété de la distance : ..... 37
II.2.2 la capacité de correction d'erreur du code convolutionnel : ..... 41
II.2.3 Gain de codage ..... 42
II.3 LE TAUX DE RETRAÇAGE D'UN CODE CONVOLUTIONNEL : ..... 45
II.3.1 Comparaison et limitation de Viterbi : ..... 45

III.TECHNIQUES DE DEODAGE DE VITERBI SUR UN DSP ..... 46

III.1 CODAGE CONVOLUTIONNEL : ..... 48
III .1.1 : Processus de codage : ..... 48
III.1.2 Diagrammes d'état et en treillis de l'exemple GSM ..... 49
III.1.3 Codage convolutionnel sur le TMS320C54 : ..... 52
III.1.3.1 Edition de lien ..... 52
III.1.3.2 Programme en assembleur de l'encodeur : ..... 54
III.2. IMPLEMENTATION SUR DSP DE L'ALGORITHME DE VITERBI : ..... 57
III.2.1. Mise à jour métrique : ..... 58
III.2.2. Rétro-traçage : ..... 59
III.3. COMPARAISON ENTRES LES DECISIONS SOFT ET HARD : ..... 60

# SOMMAIRE

III.4. CALCUL DE LA DISTANCE LOCALE : ..... 60
III.5. CODE TMS320C54X DE DECODAGE DE VITERBI ..... 62
III.5.1 Initialisation : ..... 62
III.5.2 La mise à jour métrique des états du treillis : ..... 65
III.5.3 La fonction de rétro-traçage : ..... 71
CONCLUSION ..... 76
BIBLIOGRAPHIE : ..... 78
ABREVIATION : ..... 79
ANNEXE ..... 80

## Liste des figures :

Figure I. 1 : Encodeur de convolution de longueur k et de taux k/n. ..... 13
Figure I. 2: Encodeur de convolution de longueur k=3 et de taux ½. ..... 14
Figure I. 3: Décodage convolutionnel a un message de taux ½, k=3. ..... 15
Figure I. 4: Diagramme d'état du codeur (taux ½, k = 3). ..... 19
Figure I. 5: encodeur diagramme en treillis (taux ½ k = 3) ..... 20
Figure II. 1: la décision de décodage hard et soft. ..... 26
Figure II. 2: Le diagramme de décodage de treillis (taux ½, k=3). ..... 28
Figure II. 3 : métriques de chemin pour deux voies. ..... 30
Figure II. 4 : Sélection des chemins survivants ..... 32
Figure II. 5 : Exemple de décodeur des cellules. ..... 33
Figure II. 6: l'unité logique que rend effectif de les fonctions l'ajout-comparer-sélectionner correspond à la cellule 1. ..... 34
Figure II. 7 : ajouter-comparer-sélectionner les comptes dans le décodage de Viterbi. ..... 36
Figure II. 8: diagramme d'état étiqueté conformément à la distance du chemin tous-zéros. ..... 38
Figure II. 9 : Diagramme d'état selon la distance, longueur, et nombre des uns en entrée. ..... 41
Figure II. 10 : Diagramme de treillis étiqueté avec la distance du chemin de tous-zéros. ..... 41
Figure II. 11 : Probabilité d'erreur binaire en fonction de $E_b / N_0$. ..... 44
Figure II. 12: Performance d'erreur par bit pour différents schémas de décodage de Viterbi. ..... 45
Figure III. 1: Longueur de contrainte 5, codeur à convolution de débit½. ..... 48
Figure III. 2: Diagramme d'état ..... 51
Figure III. 3: Diagramme en treillis. ..... 52
Figure III. 4: Configuration de la mémoire interne du C5410 ..... 53
Figure III. 5: Pseudo-Code pour l'algorithme de Viterbi ..... 58
Figure III. 6: quatre buffers pour la mise à jour métrique des états du treillis ..... 63
Figure III. 7: Cellule en papillon du treillis propre au codeur convolutionnel GSM ..... 65

# Liste des tables

Listes des tables :

Table I. 1 : sortie du registre en fonction de déplacement d'un bit "1" d'entrée. ..... 16
Table I. 2 : superposition des réponses impulsionnelle suite aux bits"1" d'entrée. ..... 17
Table II. 1: Limite supérieure de gain de codage pour quelques codes convolutionnels. ..... 43
Table II. 2 : Gain de décodage de base (dB) pour décodage de Viterbi a décision soft. ..... 44
Table III. 1: Etats du codeur en relation avec ses E/S ..... 50
Table III. 2: Sorties codées après un premier traitement ..... 55
Table III. 3: Valeurs à décision soft. ..... 60
Table III. 4: Les opérations de mise à jour métrique du décodage de Viterbi en GSM. ..... 67
Table III. 5: Ordre des états dans la donnée de transition pour un intervalle de symbole ..... 71

# Introduction générale

## Introduction

Le codage de canal est un moyen très utilisé pour améliorer les performances des systèmes de communication en présence de bruit et distorsion de canal. Les techniques de correction d'erreurs sont les plus populaires dans les systèmes de communication numériques.

L'un des codages le plus utilisé dans ce cas, est le codage convolutionnel qui est souvent choisi dans la communication numérique moderne, comme codage interne. Accompagné, du codage de Reed-Solomon comme codage externe. Car il est insuffisant d'utiliser un seul codage correcteur d'erreur dans des systèmes complexes de transmission numérique, tels que GSM, LTE, Wimax, etc.

Notre but est d'implémenter le codage convolutionnel au niveau du transmetteur, et le décodage convolutionnel, en l'occurrence celui basé sur l'algorithme de Viterbi, dans le récepteur réciproque. L'implémentation est à assurer sur un processeur dédié au traitement du signal, qui est un DSP de la famille TM32054X.

Notre mémoire est composé de trois parties essentielles. Au premier chapitre, nous allons étudier les codes correcteurs d'erreurs convolutionnels avec leurs présentations, sous forme de polynômes générateurs, de réponse impulsionnelle, de diagramme d'état, et de diagramme en treillis.

Dans le deuxième chapitre, nous allons voir les différentes formulations du décodage convolutionnel, telles que l'algorithme de maximum de vraisemblance, et l'algorithme de Viterbi. Les propriétés inhérentes aux codeurs comme la fonction de transfert du codeur, la distance libre, le gain de codage, seront présentées

Le troisième et dernier chapitre, décrit l'implémentation du codeur et décodeur convolutionnel sur un DSP de la famille TMS320C54X, en considérant un exemple de données aléatoires à être codées puis décodées

# Chapitre I

# Présentation des codes convolutionnels

## Introduction

Soit l'entier *k*, représentant le nombre de bits de données qui forment le bloc d'entrée à l'encodeur *n*. un entier représentant le nombre total de bits qui forme le mot de code de sortie associé. Le rapport k / n est appelé le débit du code, qui est une mesure de la redondance ajoutée. Un code convolutionnel est décrit aussi par la longueur de contrainte k, qui représente le nombre de décalage des blocs d'entrée dans le registre à décalage.

## I.1L'encodage convolutionnel:

Ainsi, un encodeur convolutionnel général, figure I.1, est réaliser à l'aide d'un registre à décalage à Kk étages et *n* additionneur modulo 2 (on ports XOR).

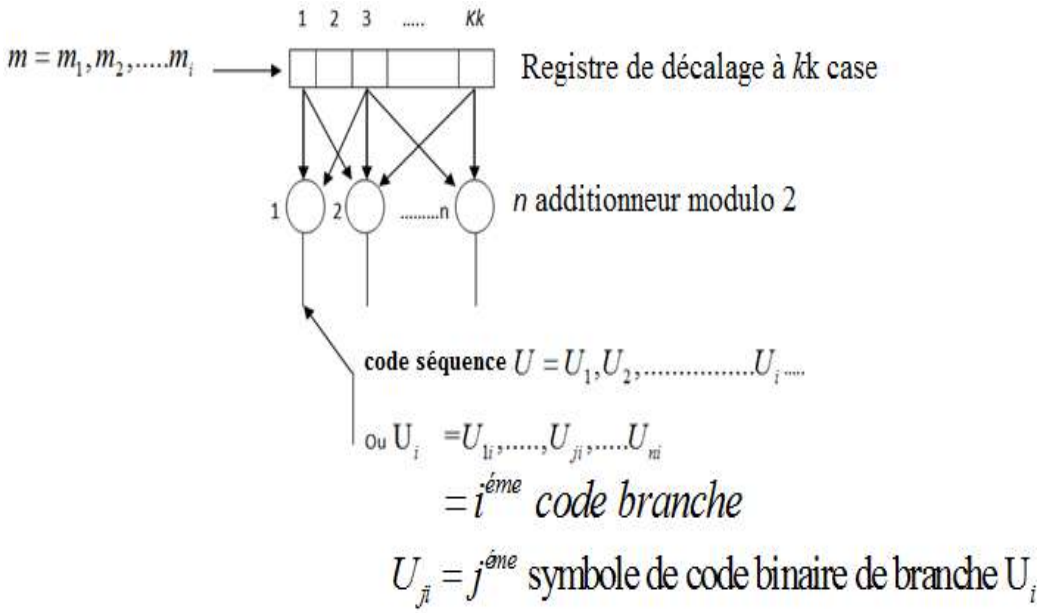



**Figure I. 1 : Encodeur de convolution de longueur k et de taux k/n.**

Par exemple, un décalage successif par 3 bits à la fois de mot d'entée constitués de k=5 bits chacun, dans un registre à décalage à 15 cellules implique une longueur de contrainte de K=15/3=5. A l'entrée de chaque *k* bits au la totalité des bits dans le registre subissent un décalage de k étages à droite, les sorties des *n* XOR collectées séquentiellement constituent les bits de code.

Comme on *n* bits de code pour chaque groupe d'entrée de *k* bits du message, le débit de code est *k* / *n* (informations par bits codé) ou k<n.

Nous constituant les codes convolutionnels usuellement utilisés pours les quels *k* = 1. C'est-à-dire, ou les bits du message sont décalés à l'intérieure de l'encodeur, un bit à la fois.

On a donc n bits codés pour chaque bits du message, soit un débit de code 1/n. [1]

## I.2 vecteur de connexion :

Un exemple de codeur convolutionnel est montré en figure I.2 comme modèle de codeur au débit 1/n. il s'agit d'un codeur avec longueur de contrainte *k* = 3(3 cellules) à n=2 pour XOR. Donc, le débit de code est k/n=1/2. A l'instant d'entrée de chaque bits, les bits de registre sont décalées d'une position à la droite. La sortie de l'encodeur commute entre $u_1$ puis $u_2$ qui sont les sorties des 2 ports XOR. [1]

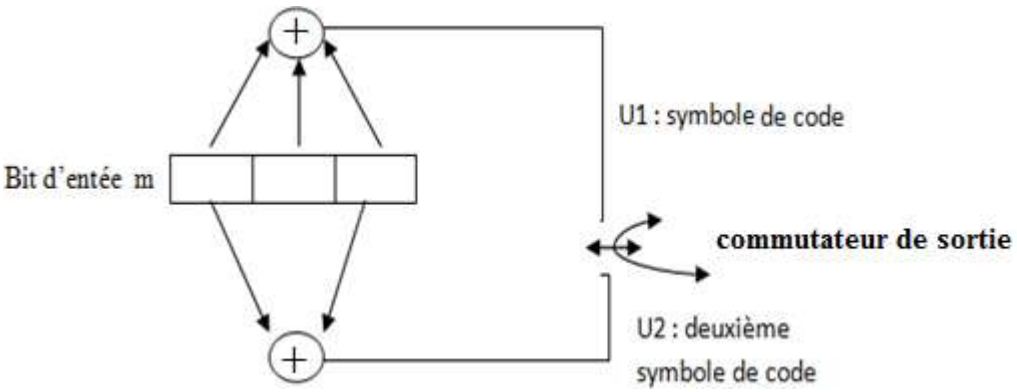



**Figure I. 2: Encodeur de convolution de longueur k=3 et de taux ½.**

Ceci forme un code à une paire de bits, associé au bit juste entré dans le registre à décalage. Le choix de connexion entre le port XOR et les étages du registre fixe les caractéristiques du code. Le problème de choix des connexions aboutissant à de meilleures propriétés de distance, est résolut par ordinateur pour toute les longueurs de contrainte n'excédant pas 20[3,5]. [1]

Une méthode pour représenter un encodeur consiste à spécifier de *n* vecteurs de connexion, liés aux *n* port XOR. Chaque vecteur est de dimension K et décrit la connexion du registre à décalage à l'un des ports XOR. Un "1" dans un vecteur, indique l'étage correspondant du registre qui est connecter au port XOR, et un "0" indique l'absence de la connexion entre

l'étage correspondant est le port XOR. Pour l'exemple de la figure I.2, les deux vecteurs de connexion $g_1$ par le port XOR du haut, et $g_2$ pour le port XOR de bas, sont :

$$g_1 = 1\ 1\ 1$$
$$g_2 = 1\ 0\ 1$$

Considérant un vecteur de message m= 101, codé convolutionnellement avec le codeur de la figure I.2. Les trois bits du message entrés par un à la fois aux temps $t_1, t_2$ et $t_3$ comme le montrer la figure I.3. [1]

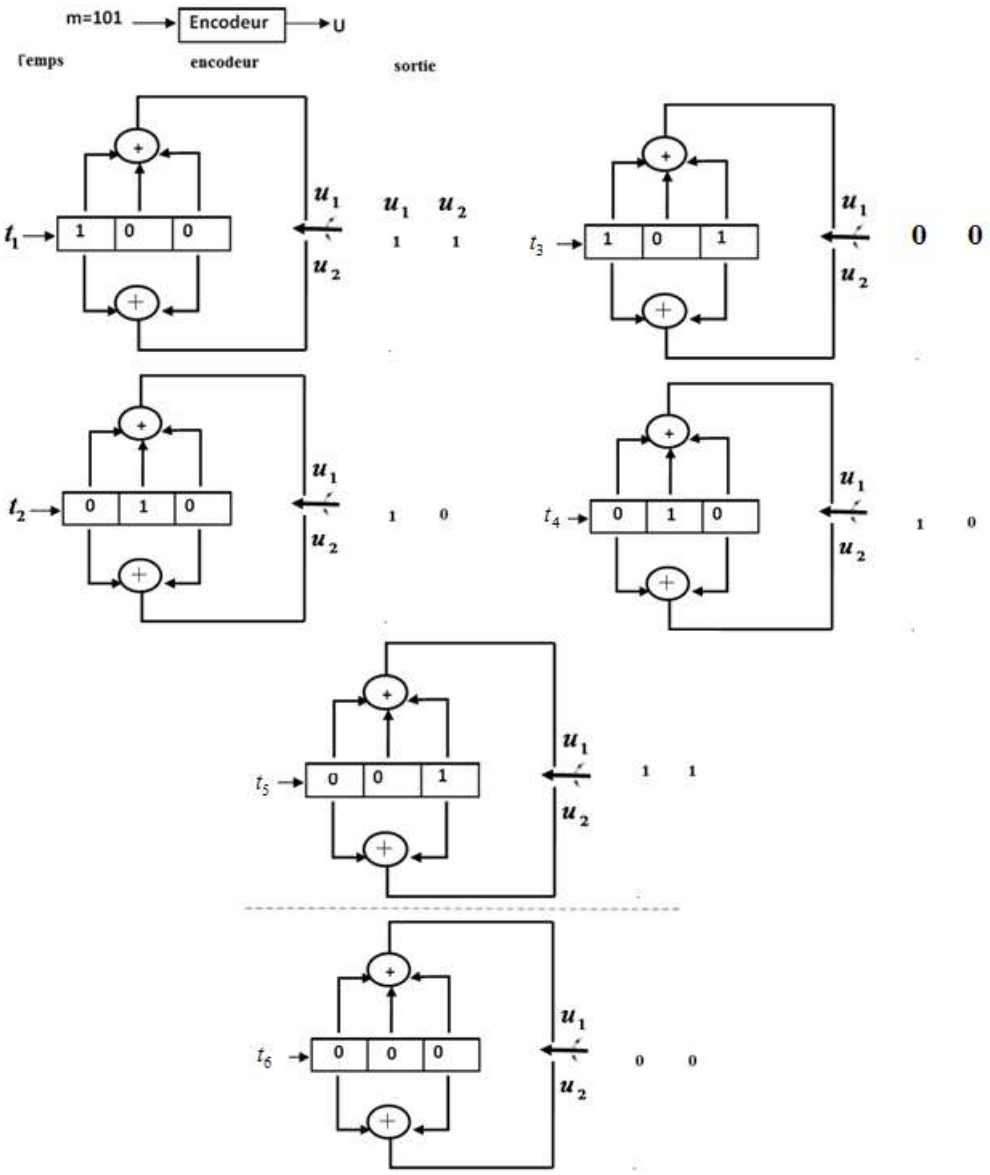



**Figure I. 3: Décodage convolutionnel a un message de taux ½, k=3.**

Par conséquent, k-1=2 zéros sont entrés aux temps $t_4$ et $t_5$ pour vider le registre et assurer que le dernier bit du message ait parcouru l'entière longueur du registre. La sortie séquentielle est donc 1110001011, ou le bit le plus à gauche représente le premier bit transmis. La séquence entière de la sortie, incluant les bits de code résultant du vidage est nécessaire au décodage du message. En plus des k-1 bits de vidage, un autre bit zéro d'entrée à l'instant $t_6$ est ajouter pour montrer que le vidage est bien terminer au temps $t_5$ . En d'autre terme un nouveau message peut être codé au temps $t_6$ . [1]

## I.3 Réponse impulsionnelle:

Une approche de l'encodeur en termes de sa réponse impulsionnelle, peut être menée. C'est la réponse à un seul bit "1"; qui se décale à travers le registre. Considérant les contenus du registre dans la figure I.2, lorsqu'un "1" se déplace à son intérieur, table I.1:

| conteneu de registre | Mots de sortie | |
|---|---|---|
| | U1 | u2 |
| 1 0 0 | 1 | 1 |
| 0 1 0 | 1 | 0 |
| 0 0 1 | 1 | 1 |

**Table I. 1 : sortie du registre en fonction de déplacement d'un bit "1" d'entrée.**

Pour une séquence d'entrée: 1 0 0. La séquence de sortie: 11 10 11.

La séquence de sortie pour une entrée d'un bit "1", est appelée la réponse impulsionnelle de l'encodeur. Ainsi, pour la séquence d'entrée de l'exemple avec m = 1 0 1, la sortie peut être exprimée par la s*uperposition* ou *l'addition linéaire* des impulsions d'entrée décalées, table I.2. [1].

| Entrée m | sortie |
|---|---|
| 1 | 11 00 11 |
| 0 | 00 00 00 |
| 1 | 11 10 11 |
| Somme modulo-2 : (ou XOR) | 11 00 00 10 |

**Table I. 2 : superposition des réponses impulsionnelle suite aux bits"1" d'entrée.**

C'est la même sortie que celle obtenue en figure I.3, démontrant que *les codes convolutionnels sont linéaires.* C'est à partir de cette propriété de production de la sortie par l'addition linéaire d' impulsions décalées dans le temps, ou la convolution de la séquence d'entrée avec la réponse impulsionnelle de l'encodeur qu'est dérivée le nom *codeur à convolutionnel*. [1]

## I.4 Représentation polynomiale :

Dés fois, les connexions de l'encodeur sont caractérisées par des générateurs polynômiaux, similaires à ceux décrivant l'implémentation des branches de retour dans un registre à décalage prévu pour des codes cyclique [sklar]. On a donc un générateur polynomial pour chaque port XOR, de degré k-1 ou moins, qui décrit la connexion du registre à décalage vers cette porte XOR, de la même manière qu'a fait un vecteur de connexion. Le coefficient de chaque terme dans le polynôme de degré k-1, est soit 1 ou 0, dépendant de l'existence ou l'absence de connexion entre le registre à décalage et le porte XOR en question. [1]

Pour l'exemple de l'encodeur de la figure I.2, on peut écrire le générateur polynômial $g_1(X)$ pour les connexions du haut, et $g_2(X)$ pour les connexions du bas comme :

$$g_1(X) = 1 + X + X^2$$

$$g_2(X) = 1 + X^2$$

Ou le terme d'ordre inferieure dans le polynôme correspondant à l'étage d'entrée du registre. La séquence de sortie est retrouvée par entrelacement, comme:

$U(X)=m(X)g_1(X)$ entrelacée avec $m(X)g_2(X)$

Tout d'abord, on exprime le vecteur du message $m=101$ comme un polynôme $m(X)=1+X^2$ . Ensuite, le polynôme de sortie $U(X)$, ou la séquence de sortie $U$ , du codeur de la figure I.2 peut être retrouvée comme suit :

$$m(X)g_1(X)=(1+X^2)(1+X+X^2)=1+X+X^3+X^4$$
$$m(X)g_2(X)=(1+X^2)(1+X^2)=1+X^4$$
$$m(X)g_1(X)=1+X+0X^2+X^3+X^4$$
$$m(X)g_2(X)=1+0X+0X^2+0X^3+X^4$$
$$U(X)=(1,1)+(1,0)X+(0,0)X^2+(1,0)X^3+(1,1)X^4$$
$$U=\ 1\,1\quad 1\ 0\quad 0\,0\quad 1\,0\quad 1\,1$$

## I.5 Le diagramme d'état:

Le codeur convolutionnel appartient à une classe de dispositifs connus comme machine à états finis qui est le nom général donné aux machines ayant une mémoire des signaux passé. Le terme fini réfère au fait qu'il n'y a qu'un nombre fini d'uniques états que la machine peut prendre. Un état consiste à la plus petite information, qui ensemble avec l'entrée courante de la machine, peut prédire sa sortie. Pour un codeur convolutionnel de débit 1/n, l'état est représenté par les contenus des k-1étages situé à droite. [1]

Soit l'état du codeur à l'instant $t_i$ défini par : $\mathrm{X}_i=m_{i-1},m_{i-2},.....,m_{i-k+1}$. La $i^{eme}$ branche $U_i$ du mot de code est complètement déterminée par l'état $\mathrm{X}_i$ et le présent bit d'entée $m_i$. L'état $\mathrm{X}_i$ représente donc l'historique de l'encodeur dans la détermination de sa sortie. L'état du codeur est dit être *Markovieur*, dans le sens par la probabilité d'être à l'état $X_{i+1}$ ne dépend que la connaissance de l'état juste passé $\mathrm{X}_i$, soit une probabilité $P(X_{i+1}|X_i)$. Une alternative pour représenter simplement un encodeur est avec un diagramme d'état.

Une telle représentation que l'encodeur des figures I.2 et I.3 et montrer en figure I.4.

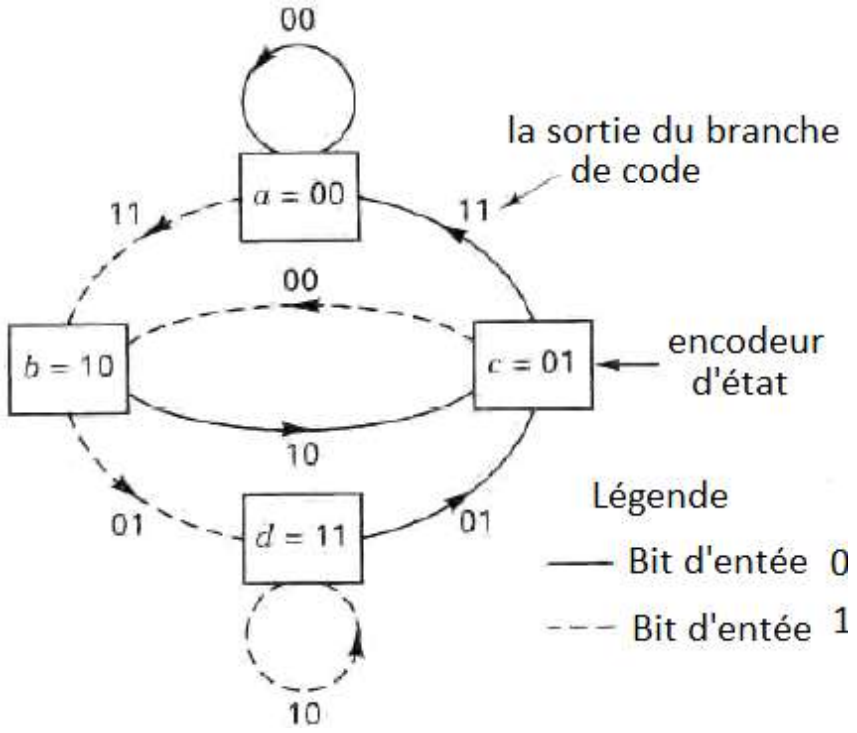



**Figure I. 4: Diagramme d'état du codeur (taux ½, k = 3).**

Les états encadrés du diagramme représentent les contenus possibles des (k-1) étages des droites du registre, et les chemins entre les états représentent les mots de sortie résultant de telles transitions entre états. Les états de registre sont désignes par *a=00, b=10, c=01*et *d=11*. Le diagramme d'état de la figure I.4 illustre toutes les transitions des états possibles de l'encodeur de la figure I.2. Il n'y a que deux transitions émanant de chaque état, correspondant aux deux bits d'entée possibles. Après de chaque branche entre états, est mentionné le mot de sortie associer à la transition d'état. En traçant la branche, on utilise la convention qu'une ligne continue signifier une branche associé avec un bit "0" d'entrée, et une ligne discontinue signifie une branche associé un bit "1"d'entée.

En conséquence au décalage d'un bit d'entrée à la fin, il n'y a peu deux transition d'état possibles par le registre peut prendre à chaque temps de bit. Par exemple, si l'état présent du code est 00, les seules possibles pour l'état au prochain décalage, sont 00 ou 10.[1]

## I.6 Le diagramme en treillis:

Même par le diagramme d'état caractérise complètement l'encodeur, il n'est pas facile de l'utiliser pour suivre ses transitions comme une fonction du temps puisque le diagramme d'état ne peut représenter l'historique temporel. Le diagramme en treillis ajointe la dimension du temps en diagramme d'état. Le diagramme en treillis du codeur convolutionnel de la figure I.2 est montré en figure I.5.

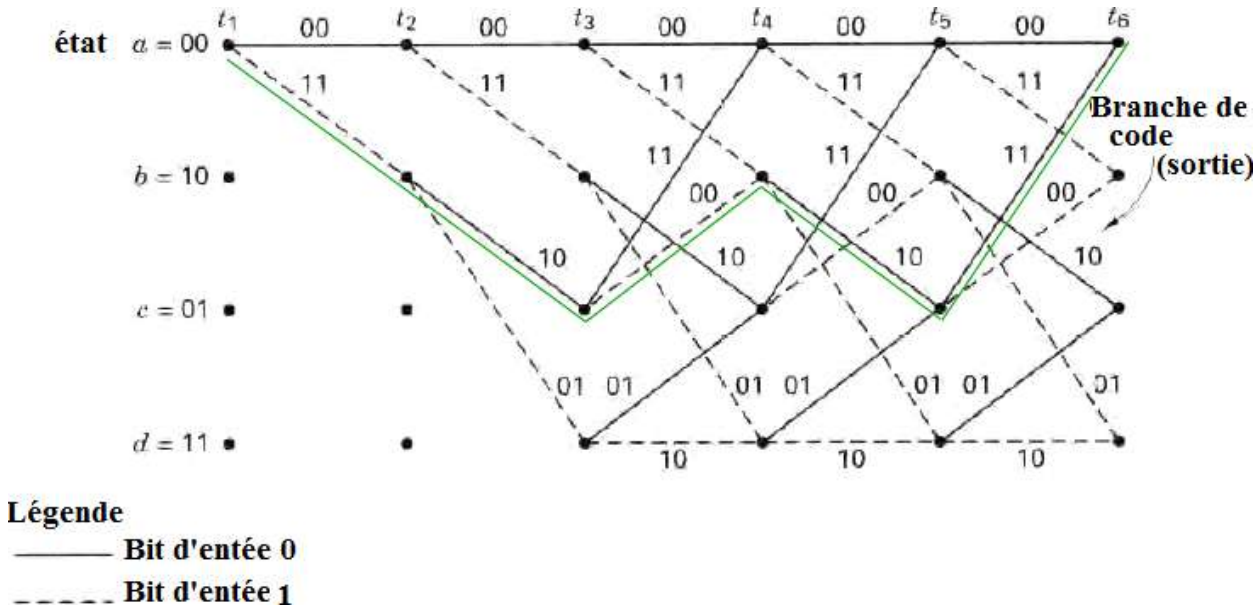



**Figure I. 5: encodeur diagramme en treillis (taux ½ k = 3) [basé sur l'arbre de la figure I.4] (En vers: le chemin propre a l'entrée 10100 de la figure I.3).**

Dans le tracé du diagramme en treillis, on utilise la même convention introduise avec le diagramme d'état. Un tracé continue dénote une sortie générée par un bit d'entrée, zéro, et un tracé discontinue dénote une sortie générée par un bit d'entrée, un.

Les nœuds du treillis caractérisent les états du codeur, ou la première ligne de nœuds correspond à l'état $a$ = 00, la seconde et les lignes suivantes correspondent aux états $b$ = 10, $c$ = 01, $d$ = 11. A chaque unité de temps le treillis nécessite $2^{k-1}$ nœuds pour présenter les $2^{k-1}$ états possibles de l'encodeur. Le treillis de notre exemple présent une structure périodique fixe après avoir atteint une profondeur de 3 dans le treillis. (Au temps $t_4$).

Dans le cas général la structure fixée apparait après qu'une profondeur k soit atteinte. A partir de ce point, chacun des états peut être accéder à partir de l'un des deux états

précédents. De même, chaque état peut transiter à l'une des deux états, avec une branche correspondante au bit d'entrée zéro et l'autre correspondant au bit d'entrée un.

Sur la figure I.5 les mots de sortie correspondant aux transitions d'état apparaissent comme étiquettes sur les branches du treillis. Une séquence de bit codé est caractériser par N branches (représentant N bits de données) occupant N intervalles de temps et associées avec un état particulier à chacun des N+1 instants (du début vers la fin). L'instant de départs est indiqué par $t_1$, tandis que le temps final de la dernière transition est indiqué par $t_{N+1}$. [1]

## Conclusion

Dans ce chapitre la manière de présenter un code convolutionnel est décrit, en citant les différentes méthodes de présentation en vecteurs de connexion, polynôme générateur, diagramme d'état, et diagramme de treillis. Dont la plus riche représentation est celle du treillis qui, on verra, sera utile pour mener à bien l'algorithme de Viterbi.

# Chapitre II

# Algorithmes de décodage Convolutionnel

## Introduction

Ce chapitre présente la partie essentielle de notre travail, car on cite maintenant les méthodes de décodage convolutionnel, en l'occurrence le décodage convolutionnel par l'algorithme de Viterbi.

## II. méthodes du décodage convolutionnel :

### II.1 décodage par maximum de vraisemblance :

Dans le cas, ou toutes les séquences du message d'entrée sont équiprobable, un décodeur qui achève la probabilité d'erreur minimale est celui qui compare les probabilités conditionnelles, aussi appelés la fonction de vraisemblance $P\left(Z\left|U^{(m)}\right.\right)$ où $Z$ est la séquence reçue et $U^{(m)}$ est l'une des séquences transmise possibles, puis choisit le maximum. Le décodeur choisit $U^{(m')}$ si :

$$P\left(Z\left|U^{(m')}\right.\right)=\max P\left(Z\left|U^{(m)}\right.\right)$$
$$\text{pour tout } U^{(m)} \qquad \text{(I.1)}$$

Comme le codeur convolutionnel présente une mémoire (la séquence reçue représente la superposition des bits courants aux bits passés), l'application du maximum de vraisemblance dans le décodage convolutionnel s'effectue dans contexte du choix de la séquence la plus probable, selon l'équation I.1. [1]

Il est en typiquement une multitude de séquences de code possibles qui peuvent avoir été transmises. Pour être spécifique, pour un code binaire, une séquence de $l$ mots est un membre parmi $2^l$ séquences possibles. De ce fais, dans le contexte du maximum de vraisemblance, le décodeur choisit un particulier $U^{(m')}$ comme séquence transmise si la vraisemblance $P\left(Z\left|U^{(m')}\right.\right)$ est plus grande que la vraisemblance de toutes les autres séquences transmise possible. Ce décodeur optimal qui minimise la probabilité d'erreur est connu comme le décodeur à maximum de vraisemblance.

En supposerons que le bruit est gaussien blanc additif à moyenne nulle, et qu'il affecte chaque code indépendamment des autres codes, le maximum de vraisemblance pour un code convolution de débit1/n est exprimé par:

$$P\left(Z\left|U^{(m')}\right.\right)=\prod_{i=1}^{\infty}P\left(Z_i\left|U_i^{(m)}\right.\right)=\prod_{i=1}^{\infty}\prod_{j=1}^{n}P\left(z_{ji}\left|u_{ji}^{(m)}\right.\right) \qquad \text{(I.2)}$$

Où $Z_i$ est la $i^{éme}$ branche de la séquence reçue $Z$, $U_i^{(m)}$ est la $i^{éme}$ branche d'une séquence particulière du code $U^{(m)}$, $z_{ji}$ est le $j^{éme}$ symbole de code de $Z_i$ et $U_i^{(m)}$ est le $j^{éme}$ symbole de code de $U_i^{(m)}$ et chaque branche comprend *n* symboles de code. Le décodage consiste à choisir le chemin à travers le treillis de la figure I.5 (ou chaque chemin possible définit une séquence de code) tel que (I.2) soit maximisée.

Généralement, il est préférable dans le calcul d'introduire le logarithme de la fonction de vraisemblance puisque cela permet une sommation au lieu d'une multiplication des termes.

$$y_U(\mathrm{m})=\log P\left(Z\left|U^{(m)}\right.\right)=\sum_{i=1}^{\infty}P\left(Z_i\left|U_i{}^{m}\right.\right)=\sum_{i=1}^{\infty}\sum_{j=1}^{n}\log P\left(Z_{ji}\left|U_{ji}{}^{m}\right.\right) \qquad \text{(I.3)}$$

Le décodeur cherchera alors à maximiser $y_U(\mathrm{m})$.

Comme pour un code binaire, le nombre possible de séquences composés le $l$ est $2^l$, le décodage a maximum de vraisemblance d'une telle séquence reçue utilisant le diagramme en treillis.

Réclame une comparaison exhaustive de $2^l$ de log-vraisemblance accumulés représentant toutes les différentes séquences possible qui peuvent avoir été transmises.

Ce moyen de calcul étant peu pratique, l'algorithme de Viterbi permet quant à lui de configurer le décodeur à fin qu'il écarte les chemins du treillis que ne peuvent être des candidats possibles pour la séquence à maximum de vraisemblance. Le chemin décodé est choisi parmi un groupe réduit de chemins survivants. Le décodeur reste ainsi optimal donc le sens ou le chemin décodé est le même que ce lui obtenu par le biais du maximum de l'équation I.3, mais la pré-rejection des chemins peu probable réduit la complexité du décodage. Cela n'implique pas que l'algorithme de Viterbi est meilleure pour chaque avec une représentation arborescente. Il est montré qu'il est possible de configurer un décodeur

qui peut jeter les chemins qui ne pouvait pas être candidats à la séquence par maximum de vraisemblance. Le chemin de décodeur est choisi parmi un certain nombre réduit de chemin survivant. Un tel décodeur est toujours optimale en ce sens que le chemin de décodage est le même que le chemin de décodage obtenu à partir de décodeur à «force brute » de vraisemblance maximale mais le rejet précoce des chemins improbables réduit la complexité de décodage. Il existe plusieurs algorithmes qui donnent des solutions approchées au problème du maximum de vraisemblance de décodage y compris la séquence [9, 10] et le seuil. Chacun de ces algorithmes est adapté à certaines applications particulières mais c'est tout sous-optimal. En contraste, l'algorithme de décodage de Viterbi effectue un décodage à maximum de vraisemblance est donc optimal. Cela ne signifie pas que l'algorithme de Viterbi est le mieux pour chaque application il ya des contraintes sévères impliquer par la complexité du matériel. [1]

### II.1.2 Modèles de récepteurs à décisions soft et hard:

Avant d'examiner un tel algorithme de décodage, nous allons décrire la partie avant décodage. La séquence des mots codés $U^{(m)}$, formée de branches comprenant chaque *n* symboles de code, peut être considérée être reçue à partir d'un démodulateur ou d'un égaliseur de canal. En général, de tels dispositifs de communication (D.C) combinent $l$ symboles à un temps donné, dans une forme d'onde de signal $s_i(t)$ avec i=1, 2,..., $M = 2^l$.

Lorsque, le D.C combine chaque symbole de code dans une forme d'onde binaire. Le signal étant corrompu par le bruit du canal, s'écrit $s_i(t) + n(t)$ où $s_1(t)$ est le signal binaire transmis durant l'intervalle de symbole T et représentant le bit "1", $s_2(t)$ par le bit"0", et $n(t)$ est le bit gaussien de moyenne nulle. [7]

Une fois échantillonné i $t = T$ , le signal à l'entré du décodeur est réduit à un nombre $z(T) = a_i + n_0$ ou $a_i$ est la composant du signal utile et $n_0$ est la composante du bruit. $z(T)$ est donc une variable aléatoire gaussienne de moyenne de $a_1$ ou $a_2$, dépendant du bit envoyé, 0 ou 1. La détection opère une décision pour connaitre le signal transmis, à partir de

la comparaison de $z(T)$ avec un seuil. Les probabilités conditionnelles de $z(T)$, $P(z|s_1)$ et $P(z|s_2)$ sont montrées dans la figure II.1.

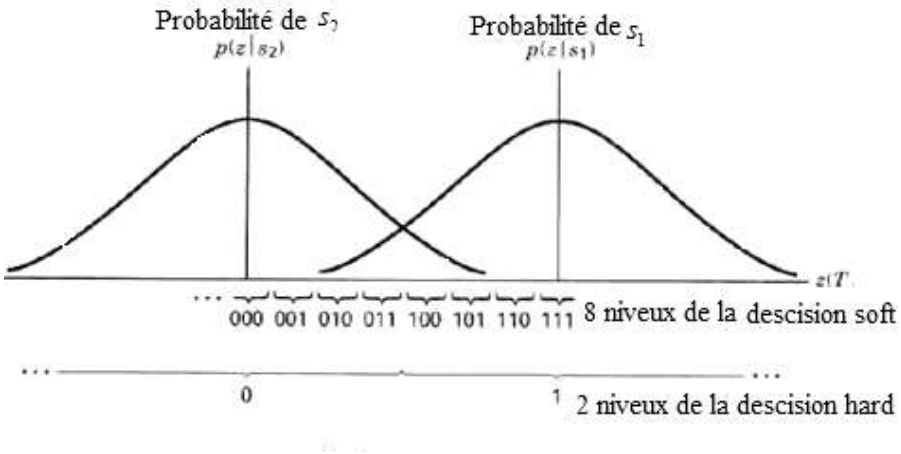



**Figure II. 1: la décision de décodage hard et soft.**

Le D.C effectue une décision hard pour dire si $z(T)$ représente un zéro ou un. La sortie est donc quantifier en deux niveaux, zéro et un, alimentant le décodeur.

Comme le décodeur opère sur des décisions hard menées par le D.C, le décodage est décrit comme décodage à décision hard.

La sortie du D.C peut envoyer au décodeur, une valeur de $z(T)$ quantifiée sur plus de deux niveaux. Lorsque le niveau de quantification du signal à l'entée du décodeur est supérieur à deux, le décodage est appelé décodage à décision soft. Une telle implémentation fournit au décodeur plus que celle du cas de la décision hard.

Huit niveaux (3 bits) de quantification sont illustrés à l'abscisse de la figure II.1.

Lorsque le D.C envoie une décision hard au décodeur, il envoie un unique symbole binaire. Lorsque le D.C envoie une décision soft au décodeur, quantifiée à huit niveaux, il envoie un mot à 3 bits décrivant un intervalle le long de $z(T)$. L'envoie d'un mot à 3 bits au lieu d'un seul bit est équivalent à l'envoie d'une mesure de confidence au décodeur accompagnant la décision elle-même sur le code.[8]

Se référent à la figure II.1, si le décodeur reçoit 111, c'est une déclaration que le bit du code est un "1" avec une très haute confidence, tandis que la réception d'un 100 est une déclaration que le bit du code est "1" avec une très faible confidence.

La décision soft procure au décodeur plus d'information, que ce dernier utilise pour retrouver la séquence avec une meilleure performance en erreur que dans le cas d'une décision hard. En figure II.1, la métrique à 8 niveaux de décision soft est souvent choisie comme -7, -5, -3, -1, 1, 3, 5, 7. Une telle décision mène à une simple interprétation de la décision-soft : le signal de la métrique représente une décision (choisir $s_1$ si positif, choisir $s_2$ si négatif), et l'amplitude de la métrique représente le niveau de confidence de cette décision. La quantification à plus de huit niveaux améliore faiblement les performances [2]. Cependant, la décision soft à huit niveaux combinent au décodage de 3 bits pour chaque symbole de code, il faut donc 3 fois plus de termes au processus de décodage. Le pris à payant pour le décodage à décision-soft est aussi une augmentation requise en taille mémoire (et possible encore en vitesse d'exécution). Dans notre simulation en assembleur, on se contentera d'une décision-hard afin de réduire la complexité de la programmation. [8]

### II.1.3 L'algorithme du décodage convolutionnel de Viterbi:

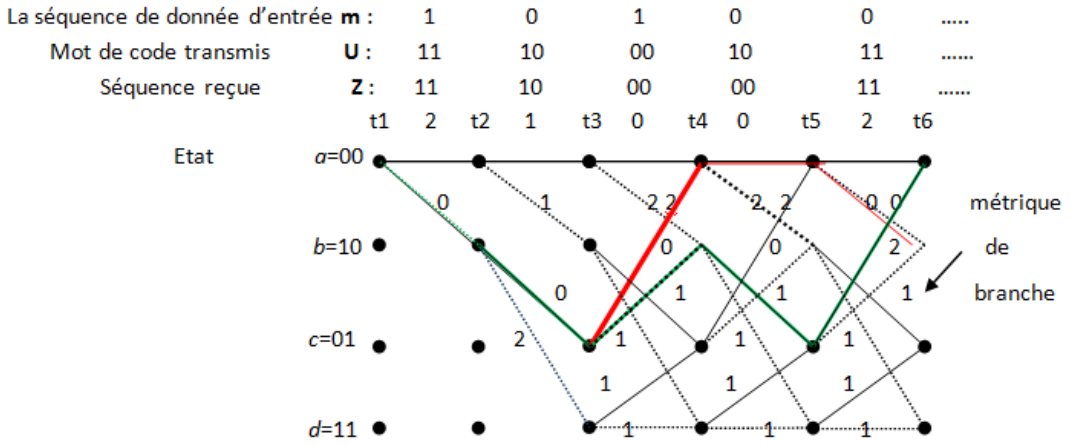
L'algorithme de décodage de Viterbi assure essentiellement le décodage de maximum de vraisemblance. Cependant, il réduit le calcul en tenant couples de la structure spéciale dans le treillis de code. L'avantage de décodage de Viterbi comparait au décodage brut, est que la complexité du décodeur de Viterbi n'est pas fonction du nombre de symboles dans la séquence de code. L'algorithme mesure de similitude en distance entre le signal reçu au temps $t_i$, et tous les chemins du treillis entrant à dans chaque état au temps $t_i$ .

L'algorithme de Viterbi supprime de considération les chemins du treillis puis ne sont pas de possible candidat par le choix du maximum de vraisemblance. Lorsque deux chemins aboutissent à un même état, celui ayant la meilleure métrique est choisie, ce chemin est appelé le chemin *survivant*. Cette sélection de chemins *survivants* est appliquée à tous les états. Le décodeur continue à avancer profondément dans le treillis, réalisant des décisions par l'élimination des chemins les moins probables.

La réjection progressive des chemins peux probables, réduit la complexité du décodage, toute en choisissant les mots de code avec une métrique de maximum de vraisemblance en de façon équivalente, une métrique à minimum de distance.

### II.1.4 Exemple du décodage convolutionnel de Viterbi :

L'encodeur de l'exemple est montré en figure I.2, et son diagramme en treillis, en figure I.5. Un treillis similaire peut être utilisé afin de représenter le décodeur, figure II.2. En début au temps $t_1$ à l'état 00. La pleine structure de treillis est atteinte après le temps $t_3$. L'idée de base de la procédure de décodage est mieux saisie en examinant les deux treillis, de l'encodeur en figure II.1 et du décodeur en figure II .2.




Au t6 2 chemins survivants (comme explique la figure II.4 avec d'autre exemple), 00200 (branche vert-rouge) avec métrique 2 : pas conservées 00010 (tous branches verts) avec métrique 1 : conservé alors avec la branche pointillés comme " 1" et la branche gras "0" dans le treillis le message sera 10100.

**Figure II. 2: Le diagramme de décodage de treillis (taux ½, k=3).**

Pour le treillis du décodeur, il convienne à chaque intervalle de temps, à étiqueter chaque branche avec la distance de Hamming entre les symboles du code reçus et le mot de la branche en correspondance sur le treillis du l'encodeur. L'exemple sur la figure II.2 montre une séquence de message **m,** la séquence de mots de code correspondant **U** et une séquence reçu corrompu par le bruit $Z$ . Les mots de branche du l'encodeur sont les symboles de code normalement émanent de la sortie du codeur comme résultat de chaque transition d'état. Les étiquettes sur les branches de treillis du décodeur sont accumulées par

le décodeur "au vol". C'est-à-dire qu'à fur et amuseur que les symboles de code sont reçues, chaque branche de treillis de décodage est étiquetée avec une métrique de similitude (distance de Hamming) entre les symboles de code reçus et chaque mot de branche à cet intervalle de temps. A partir de la séquence reçue $Z$, la figure II.2, la séquence reçue au temps $t_1$ est 11.

Afin d'étiqueter les branches du décodeur au temps de départ $t_1$ avec le métrique approprié en terme de distance de Hamming, nous examinant le treillis du l'encodeur, figure I.5.

Ici on voit pour la transition d'Etat 00→00 donne un mot de branche de sortie égale 00. Mais on a reçue 11.

Ainsi sur le treillis du décodeur, on étiquète la transition d'état 00→00 avec la distance de Hamming entre elle soit 2 (11-00=11= nombre de '1'est 2). Nous examinant encore le treillis du codeur, on voit que la transition d'état 00→10 donne un mot de branche de sortie égal à 11, qui correspond exactement avec les symboles du code reçues au temps $t_1$. Ainsi, le treillis de décodage se mit étiqueté la transition d'état 00→10 au la distance de Hamming de 0. En résume, la métrique entré sur une branche de treillis du décodeur représentant la différence (distance) entre ce qui a été reçu et ce qui devrait être reçu a partir de la branche correspondante sur le treillis de l'encodeur. Cette métrique décrit au fait une mesure de corrélation entre un mot de code de branche reçu et chaque mot de branche candidat. On continue à étiquètes les branche du treillis du décodeur de cette manière, tan que symboles sont reçues à chaque temps $t_i$. L'algorithme de décodage utilise ces métriques en distance minimum pour trouver le chemin le plus probable (distance minimum) à travers le treillis.

La base de décodage de Viterbi est que si deux branches dans le treillis accèdent à un même état, l'un des d'eux peut toujours être éliminé dans la recherche du chemin optimal. La figure II.3 montre un exemple de deux chemins aboutissant au temps $t_5$ à l'état 00. On définissant la métrique de chemin de Hamming cumulative pour un chemin donné au temps $t_i$ comme la somme des métriques en distance de Hamming des branches long de ce chemin jusqu'au temps $t_i$ .

On figure II.3, le chemin du haut à une métrique de 4, le chemin de bas a une métrique de 1. Le chemin de haut ne peut être une position du chemin optimum à cause du chemin de base entrant le même état avec une plus faible métrique.

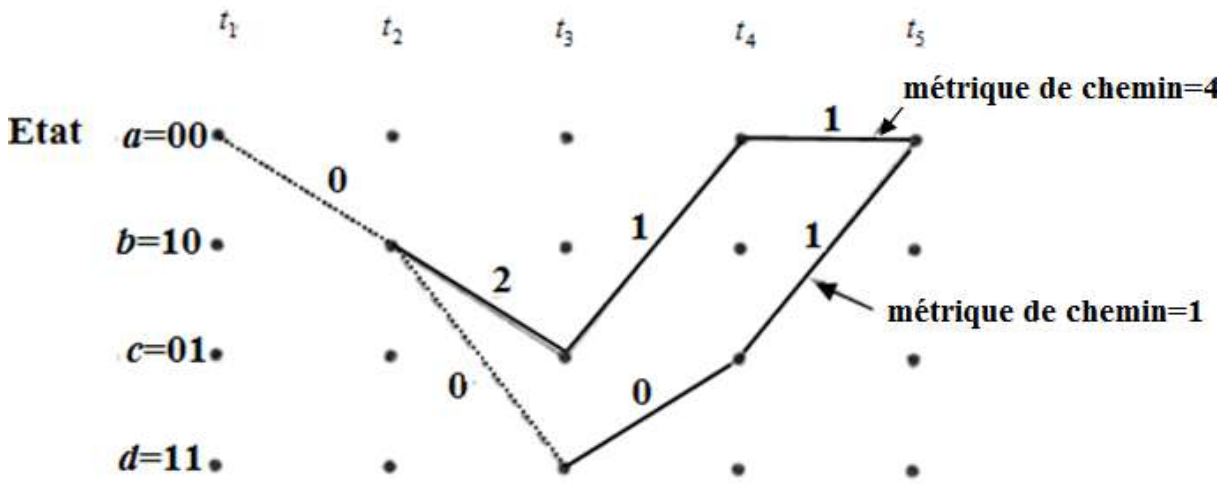



**Figure II. 3 : métriques de chemin pour deux voies.**

A chaque instant $t_i$ il ya $2^{k-1}$ états dans le treillis, où K est la longueur de contrainte, et chaque état peuvent être d'accès au moins de deux chemins. Le décodage de Viterbi consiste à traves la mesure métrique à éliminer l'un d'eux. Cette méthode est effectué pour chacun des $2^{k-1}$ états au nœud au moment $t_i$ puis le décodeur déplace à temps $t_{i+1}$ et répète le processus. A un moment donné, la métrique du chemin gagnant pour chaque état est désignée comme *l'état métrique* de cet état, à ce temps, nous allons évoquée quelle que pas sur un exemple en figure II.4. Soit la séquence de données d'entrée **m,** le mot de code **U,** et la séquence reçu **Z,** comme montrer en figure II.2. [9]

Au temps $t_1$ les symboles de code reçues soit 11. À partir de l'état 00 les seules transitions possibles sont vers l'état 00 ou état 11, comme montrer la figure II.4a. La transition d'état 00→00 a une métrique de branche 2, transition d'état 00→10 a une métrique de branche 0. Au temps $t_2$ il y a deux branches possibles quittant chaque état, figure II.4b. Les métriques cumulatives de ces branches sont étiquetés $\Gamma_{a,}\Gamma_{b,}\Gamma_c$ et $\Gamma_d$ correspondant à l'état d'arrivée. A l'instant $t_3$ en figure II.4c il y a encore deux branches divergeant de chaque état.

En résultat, il y a deux chemins entrant chaque état au temps $t_4$. Un chemin entrant chaque état peut être éliminé, celui ayant le la plus grand métrique cumulatif de trajet. Dans le cas d'égalité des métriques de deux chemins entrants, l'un d'eux est éliminé arbitrairement.

Le chemin survivant à chaque état est montrée en figure II.4d. À ce point de décodage, il y a qu'un seulement un seul chemin survivant entre les temps $t_1$ et $t_2$. Ainsi, le décodeur peut maintenant décider que la transition d'état qui si produite entre $t_1$ et $t_2$ est 00→10. Comme cette transition a eu lieu par l'entrée d'un bit '1', le décodeur fait sortir un '1' comme premier bit décodé. On note pour le premier bit n'a été décodé qu'après que le calcul du métrique de trajet s'est pour suivi assez profondément dans le treillis.

Pour une implémentation typique de décodeur ce retard de décodage est évalué à cinq fois la longueur de contrainte en bits. La figure II.4e montre la prochaine étape dans le processus de décodage. Au temps $t_5$ il ya encore deux chemins entrent chaque état et l'un de chaque paire être éliminées. La figure II.4f montre le survivant au temps $t_5$. On Note dans notre exemple qu'on ne peut encore mètre une décision sur le second bit d'entrer de donnée, car il reste deux branches quittant l'état 10 au temps $t_2$ . Au temps $t_6$ en figure II.4g on se trouve de nouveaux avec les chemins survivants vers les nouveaux états, et en figure II.4h on voit les survivants au temps $t_6$.

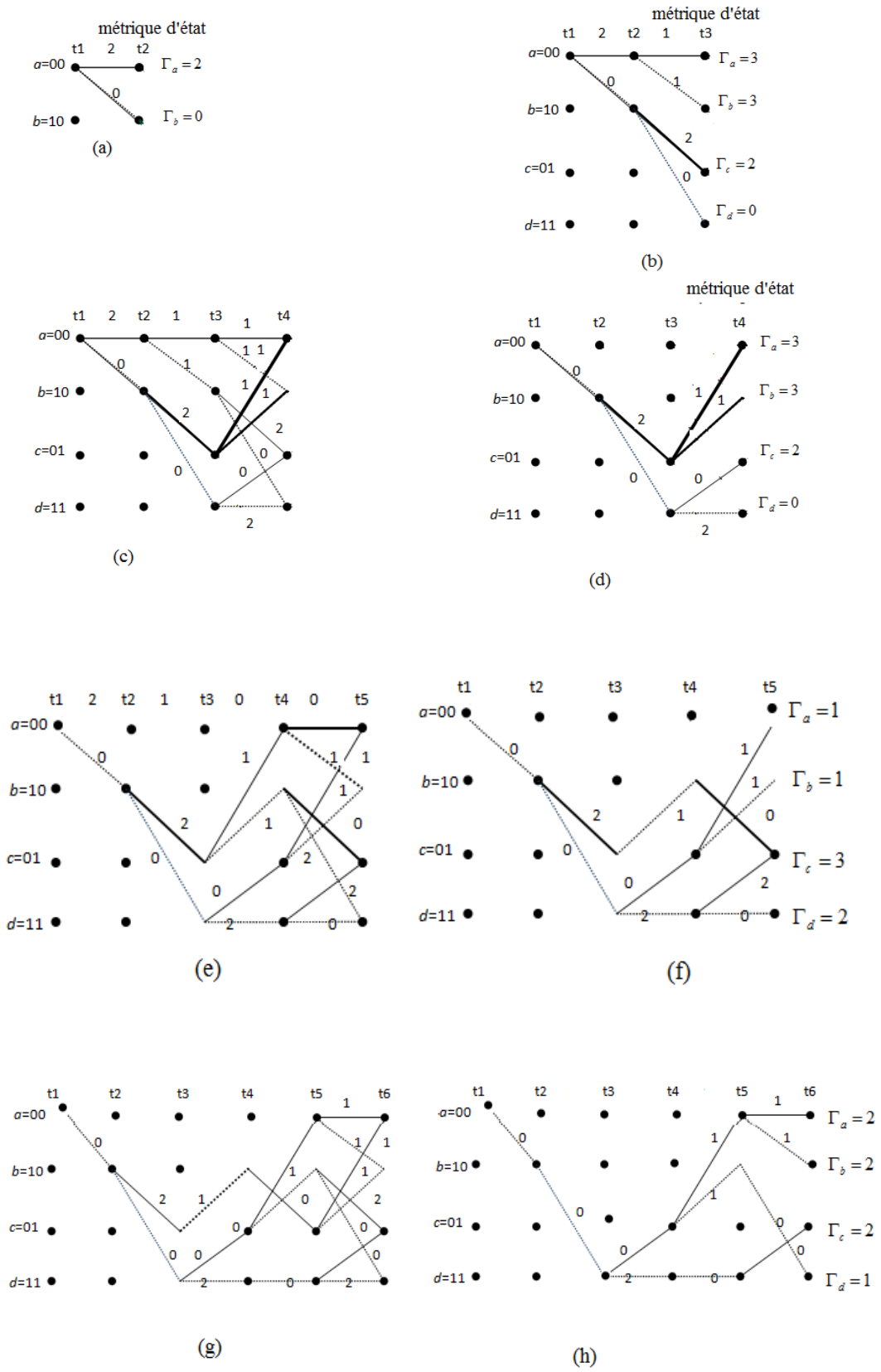



**Figure II. 4 : Sélection de chemins survivants (a) survivant à $t_2$ . (b) survivant à $t_3$ . (c) des comparaisons métriques à $t_4$ . (d) de survivant à $t_4$ . (e) de comparaisons métriques à $t_5$ . (f) survivant à $t_5$ . (g) des comparaisons métriques à $t_6$ . (h) survivant à $t_6$ .**

Aussi en figure II.4(h) le décodeur fournit un '1' comme le second bit décodé correspondant au l'unique chemin survivant entre $t_2$ et $t_3$.

Le décodeur continu ainsi a avancé profondément dans le treillis et à mettre des décisions sur bit d'entrée de donnée par élimination de toutes les branches sauf un.

Cette démarche garantit qu'il n'y a jamais pus de branche que d'état. En vérifie par exemple en figure II.4.b. d. f et h qu'il ya seulement 4 chemins. Par contre le calcul brut du maximum de vraisemblance doit analyser $2^l$ séquence possibilité par une séquence de code binaire ayant une longueur de $l$ mots de branche. [8]

### II.1.5 Implémentation de décodeur :

Dans le contexte du diagramme de treillis de la figure II.2, la formation des cellules de treillis s'accomplit après un temps $t = K$. Les transitions durant n'importe qu'elle intervalle de temps par la suite peut être regroupées dans $2^{K-2}$ cellules disjointes au chaque cellules supporte quatre transmissions possible. Pour le k = 3 et par exemple on a $2^{K-2} = 2$ cellules qui sont montrées en figure II.5 où *a*, *b*, *c* et *d* se réfèrent aux états au temps $t_i$ et *a'*, *b' c'* et *d'* se référent aux états au temps $t_{i+1}$. Chaque branche porte la métrique de branche $\delta_{xy}$ où l'indice indique que la métrique correspond à la transition de l'état $x$ vers l'état $y$. Ces cellules et unités logiques associés assurant la mise à jour *des métrique d'état* [$\Gamma_x$] où $x$ désigne un état particulier représentent les blocs de base du décodeur. [7]

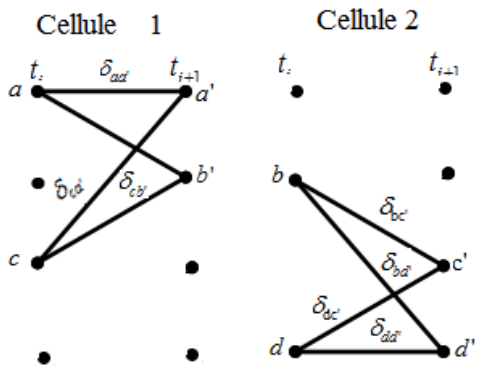



**Figure II. 5 : Exemple de décodeur des cellules.**

### II.1.6 Calcul par addition-comparaison-sélection (ACS):

En continuant avec l'exemple k = 3, 2 cellules la figure II.6 illustre l'unité logique correspond à la cellule n°1. La logique exécute les opérations d'additions-comparaison-sélection (ACS). La métrique d'état $\Gamma_a$ a la métrique de branche $\delta_{aa'}$ ,et l'ancien métrique d'état de l'état C, $\Gamma_c$ à la métrique de branche $\delta_{ca'}$. Ceci résulte en deux possibles métriques de branche comme candidats au nouveau métrique d'état $\Gamma_{a'}$.Les deux candidates sont comparées dans l'unité logique de la figure II.6. Le plus grand maximum de vraisemblance (distance minimale) des deux métriques de branche, est stocké, le nouveau historique de chemin $\hat{m}_{a'}$ pour l'état a, ou $\hat{m}_{a'}$ est l'historique du chemin de message de l'état augmenté de la donnée du chemin sélectionné.

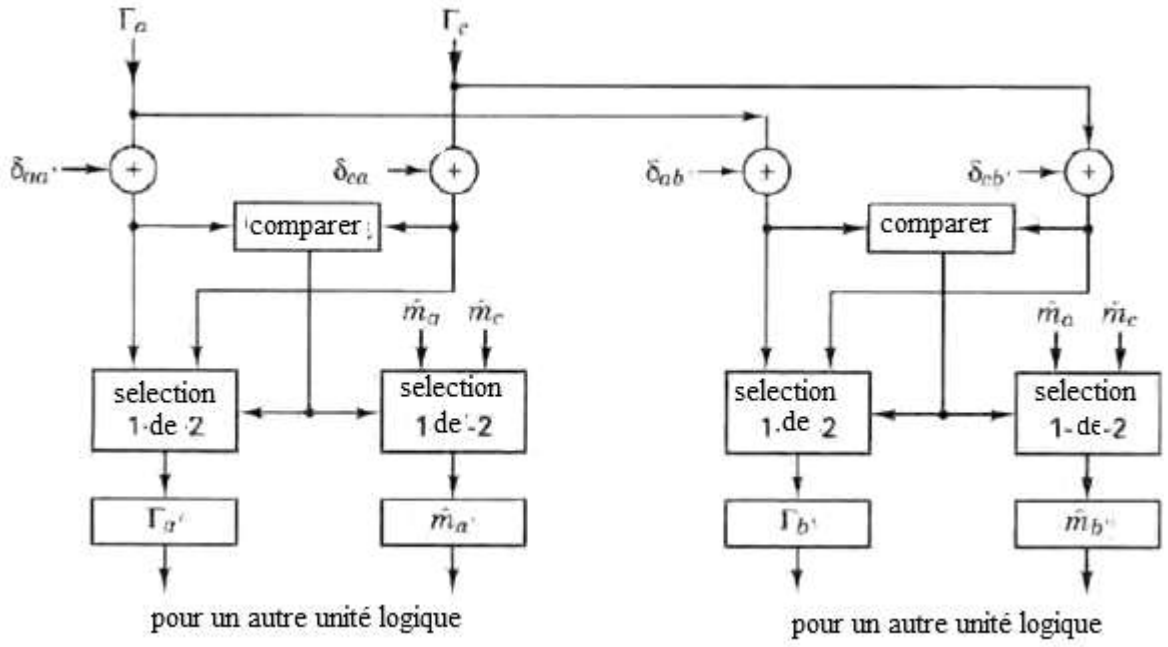



**Figure II. 6: l'unité logique que rend effectif de les fonctions l'ajout-comparer-sélectionner correspond à la cellule 1.**

En figure II.6, est montré aussi la logique (ACS) de la cellule numéro 1 qui a aboutie la nouvelle métrique d'état $\Gamma_{b'}$ et le nouvel historique de chemin $\hat{m}_{b'}$. Les anciens bits sur le chemin avec le plus faible métrique d'état, forment la sortie de décodeur. [8]

### II.1.7 ACS mené sur le treillis:

Considérant le même exemple qui utilisé dans la description du décodage de Viterbi, figure II.4. La séquence du message était **m**=11011, la séquence de mot de code était **U**=1101010001, et la séquence reçue était Z=1101011001. La figue II.7 retrace un diagramme en treillis similaire à celui de la figure II.2. La métrique du branche affichée sur chaque branche est la distance de Hamming entre les symboles le code reçus et le mot de branche correspondant sur le treillis du l'encodeur. En plus, la figure II.7 indique une valeur à chaque état x, et pour chaque temps de puis le temps $t_2$ à $t_6$ est la métrique d'état $\Gamma_x$. L'opération d'ajout -comparaison-sélection (ACS) est effectué lorsque deux transitions entrent dans un état, comme les temps $t_4, t_5$ …..

Par exemple au temps $t_4$, la valeur du métrique d'état pour l'état *a* est obtenu par une incrémentation du métrique d'état $\Gamma_3$ = 3 au temps $t_3$ avec la métrique de branche $\delta_{aa'} = 1$, donnant un candidat de valeur 4. Simultanément, la métrique d'état $\Gamma_c = 2$ au temps $t_3$ est incrémenté avec la métrique de branche $\Gamma_{ca'} = 1$donne un candidat de valeur 3. L'opération ACS sélectionne le plus grand maximum de vraisemblance (distance minimale) de métriques de branche comme nouveaux métrique d'état. Ainsi, pour l'état *a* au temps $t_4$, le nouveau métrique d'état est $\Gamma_{a'} = 3$. Le chemin sélectionné est montré avec un tracé en gras, et les chemins éliminé par un tracé fin. On observe sur le treillis de la figure II.7, allons des métriques d'état de la gauche vers la droite, et chaque temps que la valeur de chaque métrique d'état est obtenue par incrémentation de métrique d'état connectée a partir de temps passé le long du chemin sélectionné (tracé en gras) avec le métrique de branche entre eux. [7]

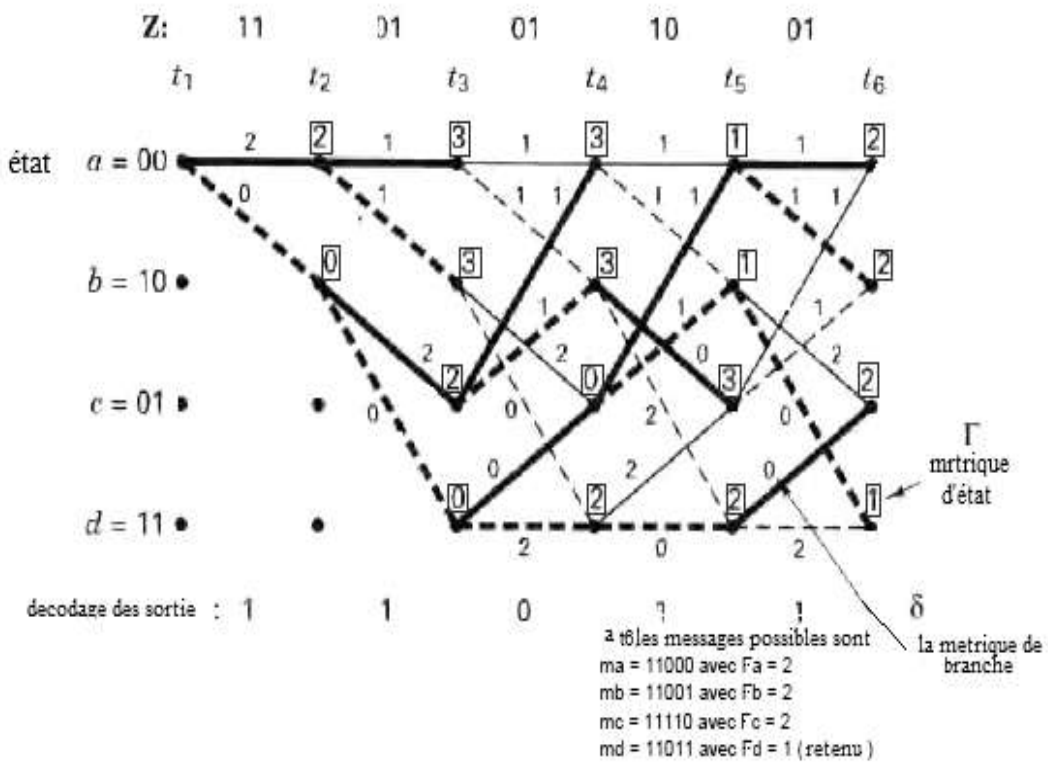



**Figure II. 7 : ajouter-comparer-sélectionner les comptes dans le décodage de Viterbi.**

À certains points dans le treillis (après un intervalle de temps de 4 ou 5 fois la longueur de contrainte), le ancien bit peuvent être décodé. Comme exemple, au temps $t_6$ en II.7, on voit que la métrique d'état à distance minimale à une d'état a une valeur de 1. A partir de l'état *d*, le chemin gagnant peut être tracé en inverse jusqu'au temps $t_1$ et on vérifie que le message décoder est le même comme le message original, on utilisant la convention qu'une ligne pleine représente le bit '0' et une ligne discontinue représente le bit '1'. [1]

### II.1.8 L'historique de chemin :

Les exigences de stockage du décodeur de Viterbi augmentent de façon exponentielle avec la longueur de contrainte **k**. Pour un code de débit espace1/N, le décodeur retient un ensemble de $2^{k-1}$ chemins après chaque étape de décodage. Une simple implémentation du décodeur contient une quantité fixe d'historique de chemin est fait sortir t le bit le plus ancien sur un chemin arbitraire, chaque fois qu'il avance d'un niveau plus profond dans le treillis. La quantité requise d'historique du chemin à stocker est :

$$u = h2^{k-1} \qquad \text{(I.4)}$$

Il a été démontré que la longueur en bit de l'historique de chemin par état, est de valeur *h* égal 4 ou 5 fois la longueur de contrainte du code qui est suffisante pour les performances entièrement optimales du décodeur. L'espace requis *u* est la limitation de base de l'implémentation des décodeurs de Viterbi. Les décodeurs commerciaux sont limités à une longueur de contrainte d'environ K = 10. Les efforts à augmenter le gain de codage par l'augmentation des contraintes sont proportionnels à une augmentation exponentielle du besoin en mémoire (et en complexité). [2]

## II.2 Les propriétés des codes convolutionnels :

### II.2.1 Propriété de la distance :

Considérons les propriétés de la distance des codes convolutionnels dans le contexte du simple encodeur en figure I.2, et son diagramme en treillis en figure I.5. On veut évaluer la distance entre toutes les paires possibles de séquences de code, et on s'intéresse à la distance minimale entre toutes les paires de ces séquences de code, car la distance minimale est liée à la capacité de correction des erreurs du code. En plus, un code convolutionnel est un groupe ou un code linéaire. En général, il n'y a aucune perte dans la recherche de la distance minimum entre chaque séquence de code et la séquence des zéros. En supposant, la séquence d'entrée tous-zéros qui a été transmis, les chemins d'intérêt sont ceux qui commencent et se terminent à l'état 00. Ainsi, l'erreur qui nous intéresse est associée au chemin survivant qui diverge de et converge vers le chemin tous-zéros.

On veut quantifier la capacité du décodeur en termes d'erreurs, et on veut aussi connaitre la manière "simple" du décodeur à commettre des erreurs. La distance minimale pour faire telle erreur peut être trouvée en examinant exhaustivement chaque chemin de l'état 00 à l'état 00.

On commence par le diagramme d'état de la figure I.4. Tout d'abord, on désigne les branches du diagramme d'état par $D^0 = 1, D^1$ ou bien $D^2$, comme illustré en figure II.8,

où l'exposant de $D$ désigne la distance de Hamming du mot de la branche vers la branche tous-zéros.

La rétro--boucle au nœud *a* peut être éliminée car elle ne contribue pas aux propriétés de la distance de la séquence de code par rapport à la séquence de tous-zéros. En outre, le nœud *a* peut être divisé en deux nœuds (noté *a* et *e*), l'une représente l'entrée et l'autre la sortie du diagramme d'état.

Tous les chemins d'origine *a*= 00 et se terminant à *e* = 00 peuvent être tracés sur le diagramme d'état modifié de la figure II.8. On peut calculer la fonction de transfert du chemin *a, b, c, e* (commence et se termine à l'état 00) en termes $D$ ,comme $D^2\ D\ D^2 = D^5$. L'exposant de $D$ représente le comptage cumulatif du nombre des uns dans le chemin, et par conséquent la distance de Hamming par rapport au chemin tous-zéros. [1]

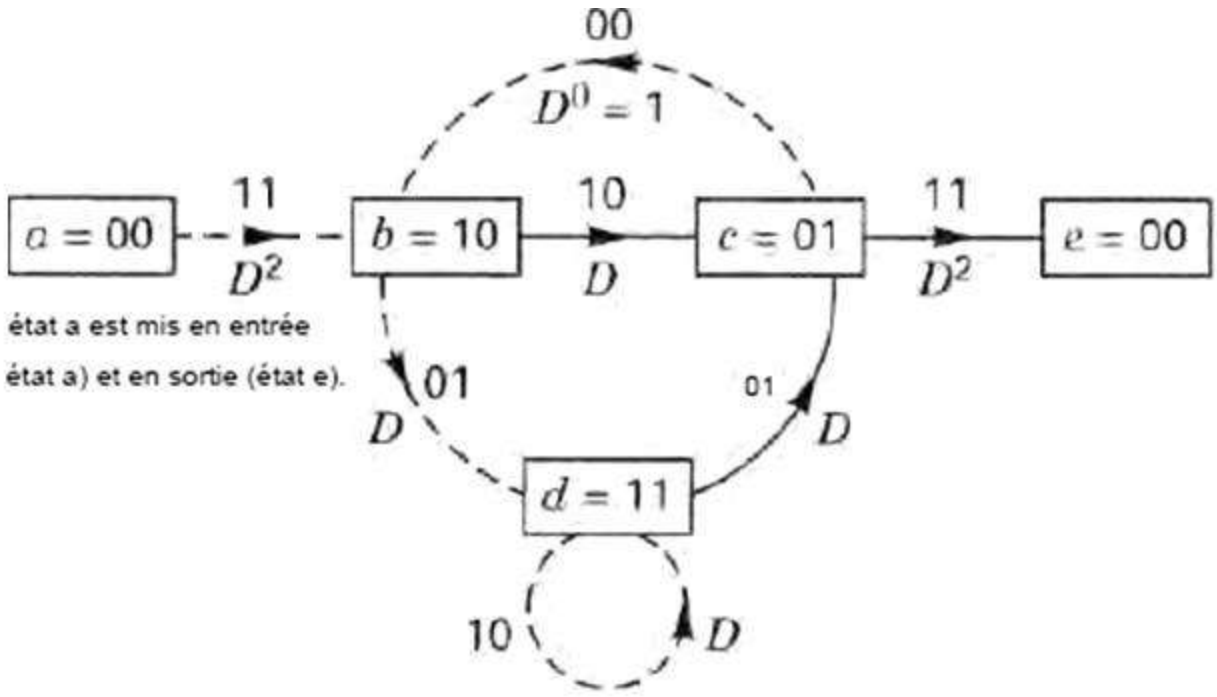



**Figure II. 8: diagramme d'état étiqueté conformément à la distance du chemin tous-zéros.**

De même, les chemins *a, b, c, d, e* et *a, b, c, e,* a chacun une fonction de transfert $D^6$ et ainsi, une distance de Hamming de 6 relativement au chemin tous-zéros. Les équations d'état sont donc :

$$X_b = D^2 X_a + X_c$$
$$X_c = DX_b + DX_d = DXb/(1-\mathrm{D}) = \mathrm{Xd} \quad d'ou \ \mathrm{Xb}=\mathrm{D}^2(1\text{-}\mathrm{D})\mathrm{X}_a/(1-2D)$$
$$X_d = DX_b + DX_d$$
$$X_e = D^2 X_c$$

(1.5)

Où chaque état (sauf l'entrée) est exprimé en fonction de ses entrées pondérées de leurs distances.

La fonction de transfert $T(\mathrm{D})$ est parfois appelée la fonction génératrice du code qui peut être exprimée comme : $T(\mathrm{D}) = X_e / X_a$. Pour résoudre les équations d'état montrées dans l'équation (I.5), on obtient:

$$T(\mathrm{D}) = \frac{D^5}{1-2D}$$ En effectuant la division, on obtient l'expansion (I.6)

$$\mathrm{T(D)}=D^5 + 2D^6 + 4D^7 + .....2^l D^{l+5} + .....$$

La fonction de transfert pour ce code indique qu'il y a un seul chemin de distance 5 au chemin tous-zéros, deux de distance 6, quatre de distance 7 et en général il y a $2^l$ chemins de distance $l+5$ au chemin tout- zéros où $l = 0,1,2,,.....$

La distance libre $d_f$ du code est le poids de Hamming du terme d'ordre plus faible dans l'expansion de $T(\mathrm{D})$, dans cet exemple $d_f = 5$. Dans l'évaluation des propriétés de la distance, la fonction de transfert $T(\mathrm{D})$ ne peut pas être utilisée pour de grandes longueur de contrainte, car la complexité de $T(\mathrm{D})$ augmente de façon exponentielle avec celle-ci. La fonction de transfert peut être utilisée pour fournir des informations plus détaillées que la simple distance des différents chemins. On introduit le facteur *L* pour chaque branche du diagramme d'état telle que l'exposant *L* puisse servir le compteur indiquant le nombre de branches dans tous les chemins de l'état *a* = 00 à l'état *e* = 00.

En outre, on peut introduire un facteur *N* pour tout transition da la branche causé par le bit d'entrée'1'. Ainsi, pour chaque branche traversée, l'exposant cumulée sur N augmente de un, seulement si cette transition de branche est due à un bit d'entrée'1'. Pour les caractéristiques du code convolutionnel dans l'exemple en figure I.2, les facteurs L et N sont indiqués sur le diagramme d'état modifié de la figure II.10. Les équations (I.5) peuvent être modifiées comme suit:

$$\begin{aligned} X_b &= D^2LNX_a + LNX_c \\ X_c &= DLX_b + DX_d = DL(\mathrm{Xb} + Xb) = \mathrm{DLXb}/(1-\mathrm{DLN}) \\ X_d &= DLNX_b + DLNX_d = DLNXb/(1-DLN) \\ X_e &= D^2LX_c \quad d'ou \quad Xb = D^2LN(1-DLN)\,\mathrm{Xa}/\,\mathrm{DL}[1-DL(1+\mathrm{L})\,\mathrm{N}] \end{aligned} \tag{I.7}$$

La fonction de transfert de ce diagramme d'état par division du numérateur sur dénominateur est :

$$\begin{aligned} T(\mathrm{D},\mathrm{L},\mathrm{N}) &= \frac{D^5L^3N}{1-DL(1+L)\,\mathrm{N}} \\ &= D^5L^3N + D^6L^4(1+L)N^2 + D^7L^5(1+L)^2N^3 \\ &+ \ldots\ldots + D^{l+5}L^{l+3}(1+\mathrm{L})^l\,N^{l+1} + \ldots\ldots \end{aligned} \tag{I.8}$$

Ainsi, on peut vérifier certaines propriétés du chemin affiché dans la figure II.10 dérivée de la figure I.5. Il y a un chemin de distance 5, de longueur 3, qui diffère par un bit d'entrée du chemin tous-zéros. Il y a deux chemins de distance 6, l'un de longueur 4 et l'autre de longueur 5, et les deux diffèrent par deux bits d'entrée du chemin tous-zéros. Ainsi, les chemins de distance 7, dont l'un est de longueur 5, deux sont de longueur 6, et l'autre de longueur 7, les quatre chemins correspondent à la séquence d'entrée qui différent en trois bits d'entrée du chemin tous-zéros.

Ainsi, si le chemin tous-zéros est le chemin correct et le bruit nous amène à choisir l'un des chemins incorrects de distance 7, trois bits erronés seront effectués. [1]

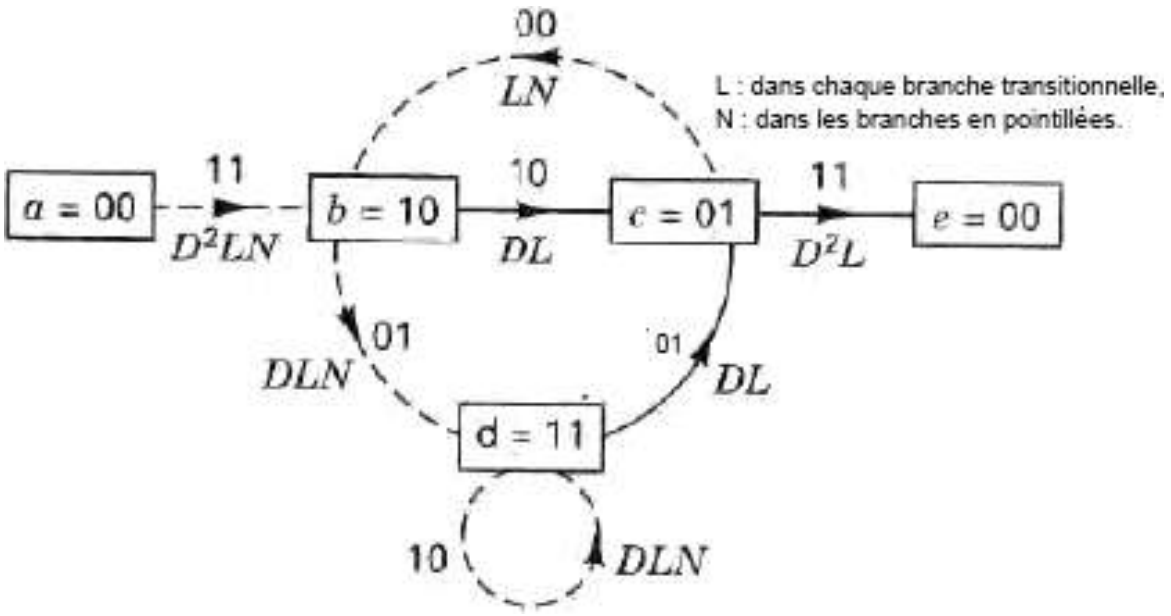



**Figure II. 9 : Diagramme d'état selon la distance, longueur, et nombre des uns en entrée.**

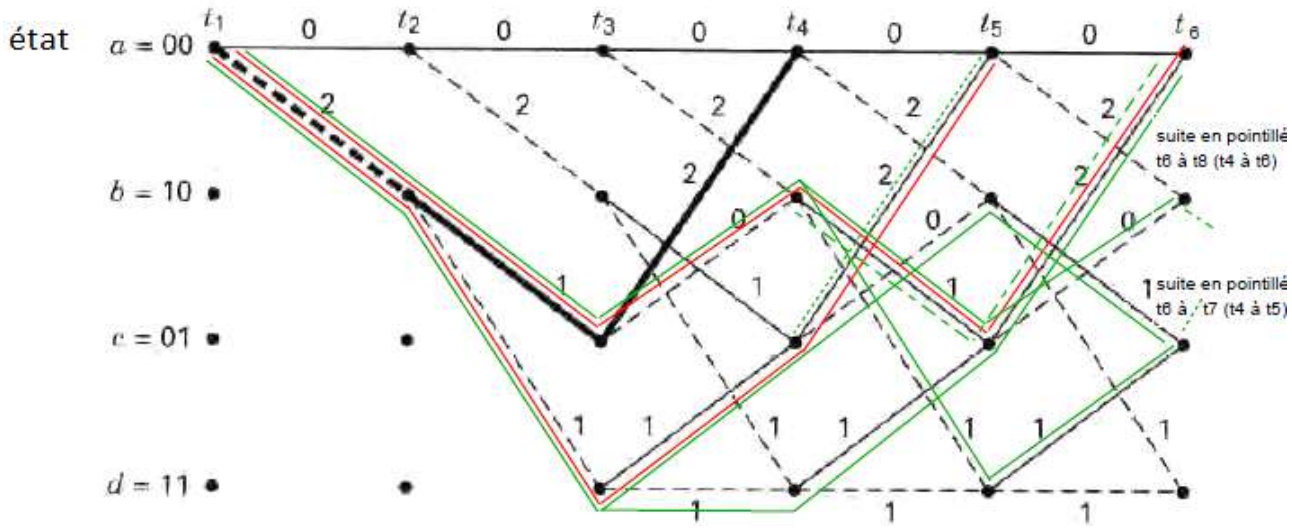



**Figure II. 10 : Diagramme de treillis étiqueté avec la distance du chemin de tous-zéros.**

A partir de l'équation (I.8), on déduit, le chemin (en gras) de distance minimale $(D^5) : D^5L^3N$, deux chemins (en rouge) de distance $D^6$ : $D^6L^4N^2$ et $D^6L^5N^2$. Quatre chemins (en vert) de distance $D^7 : D^7L^5N^3$, 2 chemins $D^7L^6N^3$ D, et $D^7L^7N^3$, etc.….

## II.2.2 la capacité de correction d'erreur du code convolutionnel :

Il y a un seul chemin de distance 5 du chemin tous-zéros. Ce chemin s'écarte du chemin tous-zéros au temps t1 et fusionne avec elle au moment t4. De même, il ya deux chemins de distance 6, l'un part au temps t1 et fusionne au temps t5, et l'autre part au temps t1 et fusionne au temps t6, et ainsi de suite. On peut également voir que les lignes pointillées

et les lignes continues du diagramme, pour les quelles les bits d'entrée pour le chemin de distance 5, sont 100, ils se diffèrent en un seul bit d'entrée à partir de la séquence d'entrée tous-zéros. De même, les bits d'entrée des chemins de distance 6 sont 1100 et10100, chacun diffère en deux positions du chemin tous-zéros. La distance minimale de l'ensemble des chemins de longueur arbitraire qui divergent et puis fusionnent est appelée la distance libre minimal, ou simplement la distance libre, qu'est considérée être 5 dans l'exemple, comme montré par les lignes en gras de la figure II.10, la capacité de correction d'erreurs du code qui est définie comme le rapport du nombre maximum garantis d'erreurs corrigibles par le mot de code est:

$$t = \left[ \frac{d_f - 1}{2} \right] \qquad \text{(I.9)}$$

Fixant $d_f = 5$, on voit que le code caractérisé par la figure I.2, réalise un encodeur pouvant corriger n'importe qu'elles deux erreurs de canal. [2]

### II.2.3 Gain de codage

Pour une probabilité d'erreur d'un bit binaire donnée, le gain de codage est défini comme la réduction en $E_b / N_0$ qui peut être réalisé par l'utilisation du code. Gain de codage G est généralement exprimé en dB, comme :

$$G(\text{dB}) = \left( \frac{E_b}{N_0} \right)_a (dB) - \left( \frac{E_b}{N_0} \right)_c (dB) \qquad \text{(I.10)}$$

Où $(E_b / N_0)_u$ et $(E_b / N_0)_c$ représentent le $E_b / N_0$ requis non codé et codé respectivement.

Le gain de codage, comme le présente l'équation (I.10), est défini comme la réduction sauvent exprimé en dB dans le $E_b / N_0$ requis pour accomplir la probabilité d'erreur spécifiée du système codé relativement au system non codé avec la même modulation et les caractéristiques de canal. La table (II.1) liste une limite supérieure aux gains de codage

comparés au BPSK cohérent, le mot codé pour plusieurs distances libres maximales de codes convolutionnels avec des longueurs de contrainte variant de 3 à 9 sur un canal Gaussien minimal d'un décodeur à décision-hard. La table illustre qu'il est possible d'obtenir un gain de codage significatif même avec un code convolutionnel simple. Le gain de codage varie avec la probabilité d'erreur par bit requise. La table (II.1) présent le gain de codage comparé avec BPSk cohérent, obtenu par une l'implémentation hardware ou bien par une simulation non codé sur un canal de Gaussien à décodage avec décision-soft. $E_b / N_0$ non codé est donné dans la colonne d'extrême gauche. De la table II.1, on peut voir que le gain de codage augmente lorsque la probabilité d'erreur de bit diminue. Cependant, le gain de codage ne peut pas augmenter indéfiniment, il a une limite supérieure comme le montre la table. Cette limite en décibels est:

$$gain - codage \leq 10\log_{10}\left(rd_f\right) \qquad \text{(I.11)}$$

Ou $r$ est le débit de code, et $d_f$ est la distance libre. L'examen de la table II.1 révèle également qu'à $P_B = 10^{-7}$, pour les débits de code ½ et 3/2, les codes les plus faibles tendent à être plus proche de la limite supérieure que les code puissants. Généralement, le décodage de Viterbi est utilisé avec entrées binaires et sorties hard ou sorties soft à 3 bits quantifiés. La longueur de contrainte varie entre 3 à 9. Le débit de codage est rarement inférieur à 1/3 et l'historique de chemin est généralement de quelques longueurs de contraintes. [2]

| Codes de taux 1/2 | | | Codes de taux 1/2 | | |
|---|---|---|---|---|---|
| K | $d_f$ | Limite supérieure (dB) | K | $d_f$ | Limite supérieure (dB) |
| 3 | 5 | 3.97 | 3 | 8 | 4.26 |
| 4 | 6 | 4.76 | 4 | 10 | 5.23 |
| 5 | 7 | 5.43 | 5 | 12 | 6.02 |
| 6 | 8 | 6.00 | 6 | 13 | 6.37 |
| 7 | 10 | 6.99 | 7 | 15 | 6.99 |
| 8 | 10 | 6.99 | 8 | 16 | 7.27 |
| 9 | 12 | 7.78 | 9 | 18 | 7.78 |

**Table II. 1: Limite supérieure de gain de codage pour quelques codes convolutionnels.**

Typiquement, les résultats de simulation de performance d'erreur sont présentés dans la figure II.11 pour un décodage de Viterbi à décision hard. Notons que chaque incrémentation

de longueur de contrainte améliore le $E_b / N_0$ requis par un facteur environ 0.5 dB à $p_B = 10^{-5}$. [2]

| $E_B/N_0$(dB) Non codé | TAUX DE CODE | 1/3 | | ½ | | | 2/3 | | 3/4 | |
|---|---|---|---|---|---|---|---|---|---|---|
| $P_B$ | K | 7 | 8 | 5 | 6 | 7 | 6 | 8 | 6 | 9 |
| 6.8 | $10^{-3}$ | 4.2 | 4.4 | 3.3 | 3.5 | 3.8 | 2.9 | 3.1 | 2.6 | 2.6 |
| 9.6 | $10^{-5}$ | 5.7 | 5.9 | 4.3 | 4.6 | 5.1 | 4.2 | 4.6 | 3.6 | 4.2 |
| 11.3 | $10^{-7}$ | 6.2 | 6.5 | 4.9 | 5.3 | 5.8 | 4.7 | 5.2 | 3.9 | 4.8 |
| Limite supérieure | | 7.0 | 7.3 | 5.4 | 6.0 | 7.0 | 5.2 | 6.7 | 4.8 | 5.7 |

**Table II. 2 : Gain de décodage de base (dB) pour décodage de Viterbi a décision soft.**

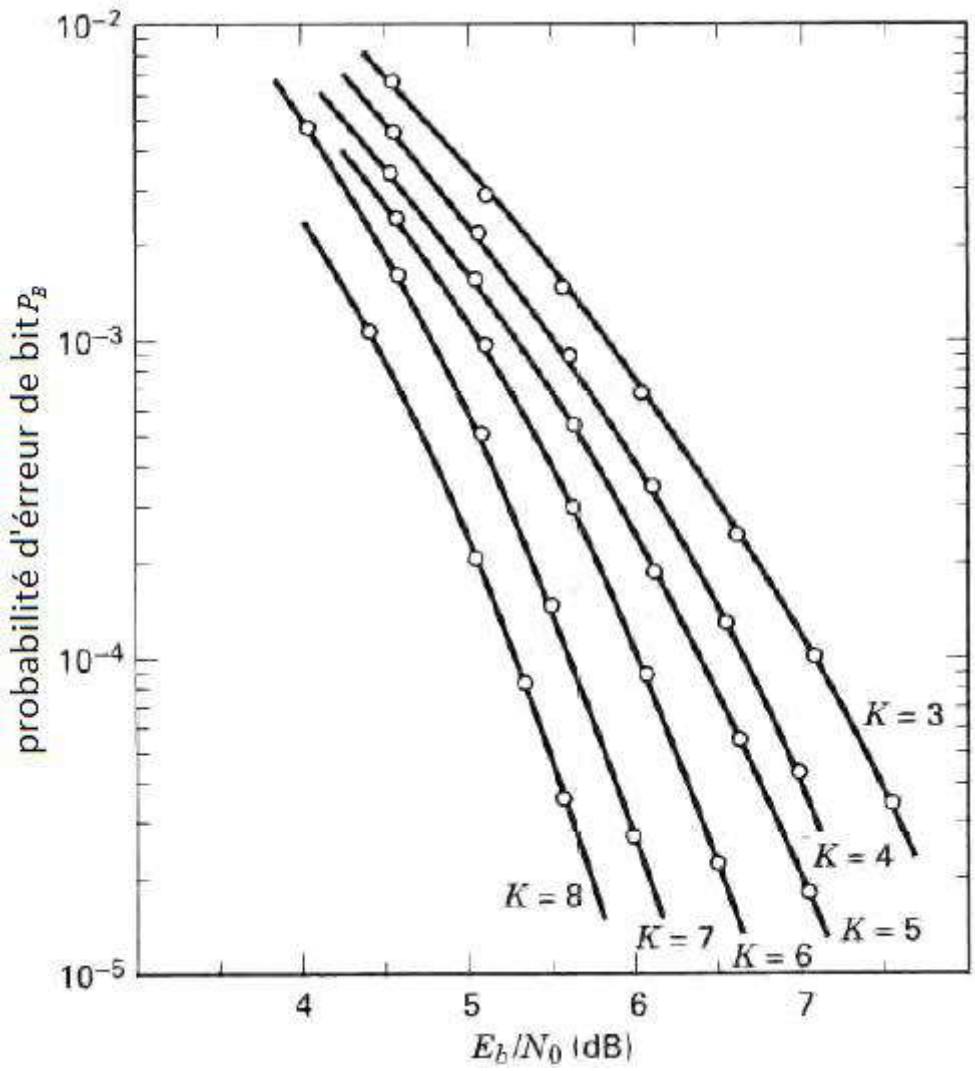



**Figure II. 11 : Probabilité d'erreur binaire en fonction de $E_b / N_0$ pour des codes de débit ½ en utilisant une PBSK cohérente.**

## II.3 Le taux de retraçage d'un code convolutionnel :

### II.3.1 Comparaison et limitation de Viterbi :

L'inconvénient majeur de l'algorithme de Viterbi est lorsque la probabilité d'erreur décroît exponentiellement avec la longueur de contrainte, le nombre des états de code et par conséquent la complexité du décodeur, croît exponentiellement avec la longueur de contrainte. D'autre part, la complexité de calcul de l'algorithme de Viterbi est indépendante des caractéristiques du canal. [1]

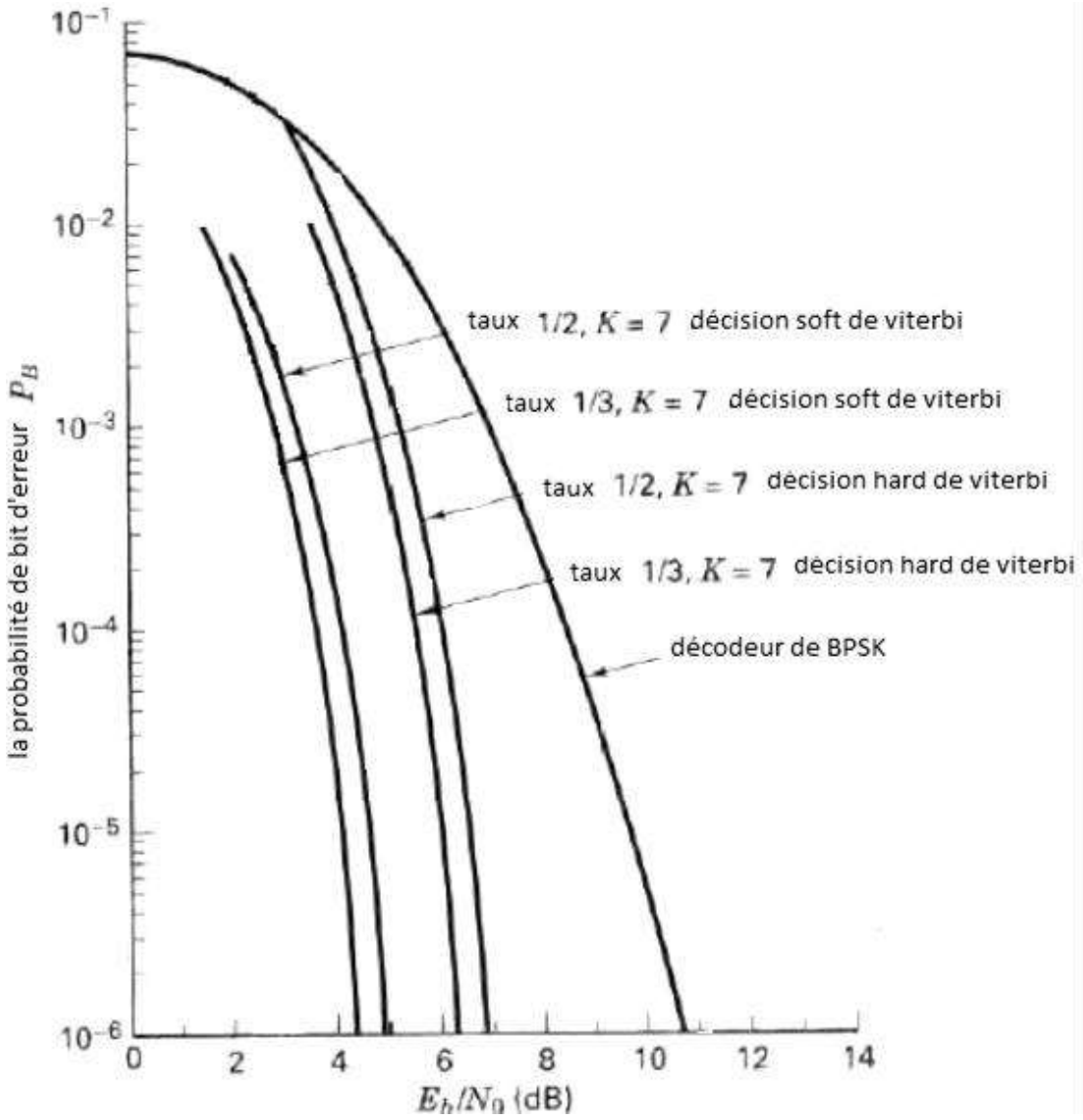



**Figure II. 12: Performance d'erreur par bit pour différents schémas de décodage de Viterbi utilisant une BPSK cohérente à terme un canal gaussien.**

En figure II.12, quelques courbes typiques de $P_B$ en fonction de $E_b / N_0$ sont présentées. Elles illustrent les performances comparatives en utilisant une BPSK cohérente à canal AWGN. Les courbes comparent le décodage de Viterbi (débit 1/2, et 1/3 décision soft, K=7), par rapport au décodage de Viterbi (débit 1/2, et 1/3 décision hard, K=7).

## Conclusion

Dans la dernière décennie, le développement du décodage a été dans le domaine des codes convolutionnel puisque dans presque toutes les applications des codes convolutionnel surpassent par les codes en bloc à la même complexité d'implémentation du décodeur et du l'en codeur. Pour les communications satellite, la technique de la correction directe de l'erreur peut facilement réduire le rapport signal sur bruit nécessaire pour une performance d'erreur spécifiée par 5 à 6 dB, ce gain en codage peut se traduire directement à une réduction équivalente de la puissance de radiation effective (EIRP) du satellite et par conséquent le poids et le coût d'un satellite.

Le fait que le débit 1/n du codes convolutionnel a une mémoire de période (K-1) bits, où K est la longueur de contrainte du codeur, avec une telle mémoire, le codage de chaque bit de donnée d'entrée, ne dépend pas seulement de la valeur de ce bit mais des valeurs de (K-1) bits d'entrée qui le précèdent, on a présenté le problème de décodage dans un contexte d'un algorithme de maximum de vraisemblance, examinons toutes les séquences des code candidates qui peuvent être crées par l'encodeur, et on sélectionne un qui apparait statistiquement le plus probable, la décision est basée sur la métrique du distance pour les symboles de code reçus.

# Chapitre III

# Les techniques de décodage de Viterbi sur un DSP

## Introduction

L'implémentation de l'algorithme de décodage de Viterbi sur DSP, va être concrétisée sur l'exemple du codage convolutionnel utilisé dans la norme GSM. La famille C54X est choisie pour son implémentation hardware de l'unité ACS.

## III.1 Codage convolutionnel :

### III .1.1 : Processus de codage :

La correction des erreurs dépend du nombre d'échantillons passés qui forment les symboles de code. Le nombre de bits d'entrée utilisées dans le processus de codage est la longueur de contrainte qui est calculée comme étant le nombre des retards unitaires plus un.

Dans la figure III.1, il y a quatre retards. La longueur de contrainte est cinq. La longueur de contrainte représente la durée totale des valeurs utilisées et est déterminée indépendamment du nombre de dérivations utilisées pour former les mots de code. La longueur de contrainte implique de nombreuses propriétés du système, et elle est fixée à cinq dans la norme GSM, figure III.1.

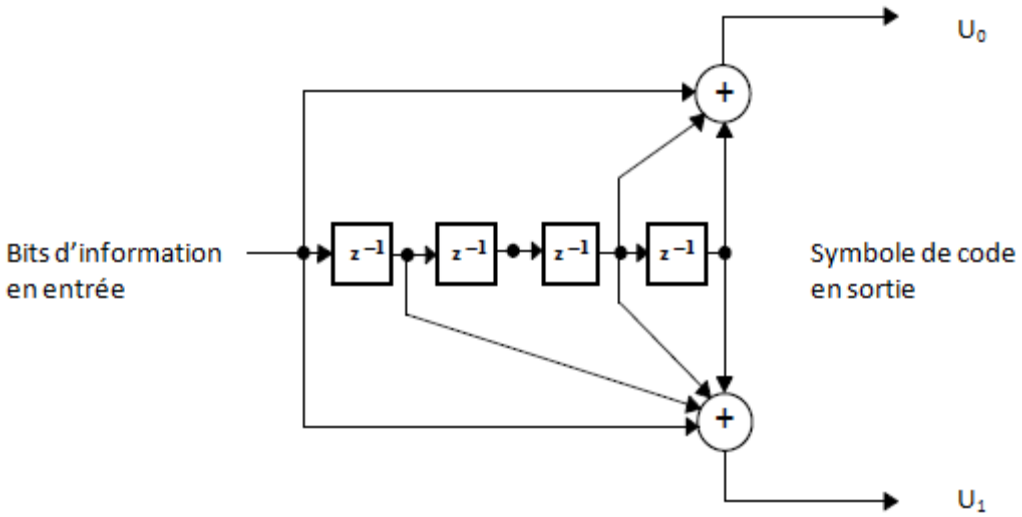



**Figure III. 1: Longueur de contrainte 5, codeur à convolution de débit ½.**

Le débit ou taux de codage est un autre facteur important qui influe sur la correction d'erreur, est représente le rapport entre les bits des données d'entrée et les bits transmis. Dans la figure III.1, deux bits sont transmis pour chaque bit d'entrée, d'où un taux de codage

de 1/2. Il est montrée que le débit des systèmes 1/n sont les plus utilisés en raison de l'efficacité du processus de décodage.

La sélection polynomiale est importante parce que chaque polynôme générateur possède différentes propriétés de correction d'erreur. La sélection des polynômes qui fournissent le plus haut degré d'orthogonalité, maximise la probabilité de trouver la séquence correcte [3]. Dans le cas de l'encodeur de la figure III.1, la combinaison des bits de sortie est décrite par les polynômes générateurs :

$$G_0(X) = 1 + X^3 + X^4 \quad \text{(III.1)}$$

$$G_1(X) = 1 + X + X^3 + X^4 \quad \text{(III.2)}$$

Ou bien, $G_0$=25 et $G_1$=27. [4]

### III.1.2 Diagrammes d'état et en treillis de l'exemple GSM

La table III.1 représente les 16 états de l'encodeur en fonction de ses sorties $U_1U_2$. Les états représentent les 4 bits retardés dans les 4 cellules du registre à décalage de l'encodeur, figure III.1.

Le nombre de retard des états dans le diagramme en treillis est déterminé par la longueur de contrainte. Dans l'exemple du codeur GSM, le nombre d'états possibles est $2^{K-1} = 16$, table III.1.

| Etats | Sorties U1U2 | |
|---|---|---|
| | 1er bit d'entrée = 0 | 1er bit d'entrée = 1 |
| 0 | 00 | 11 |
| 1 | 11 | 00 |
| 2 | 11 | 00 |
| 3 | 00 | 11 |
| 4 | 00 | 11 |
| 5 | 11 | 00 |
| 6 | 11 | 00 |
| 7 | 00 | 11 |
| 8 | 01 | 10 |
| 9 | 10 | 01 |
| A | 10 | 01 |
| B | 01 | 10 |
| C | 01 | 10 |
| D | 10 | 01 |
| E | 10 | 01 |
| F | 01 | 10 |

**Table III. 1: Etats du codeur en relation avec ses E/S**

Le nombre de bits représentant les chemins des états est la fonction du taux de codage. Dans cet exemple, deux bits de sortie sont générés pour chaque bit d'entrée, résultant en chemin d'état à 2 bits, en figure III.2. Comme les chemins des états représentent les valeurs transmises actuelles, ils correspondent à des points de constellation, dont les valeurs d'amplitude et de phase sont utilisées par le modulateur.

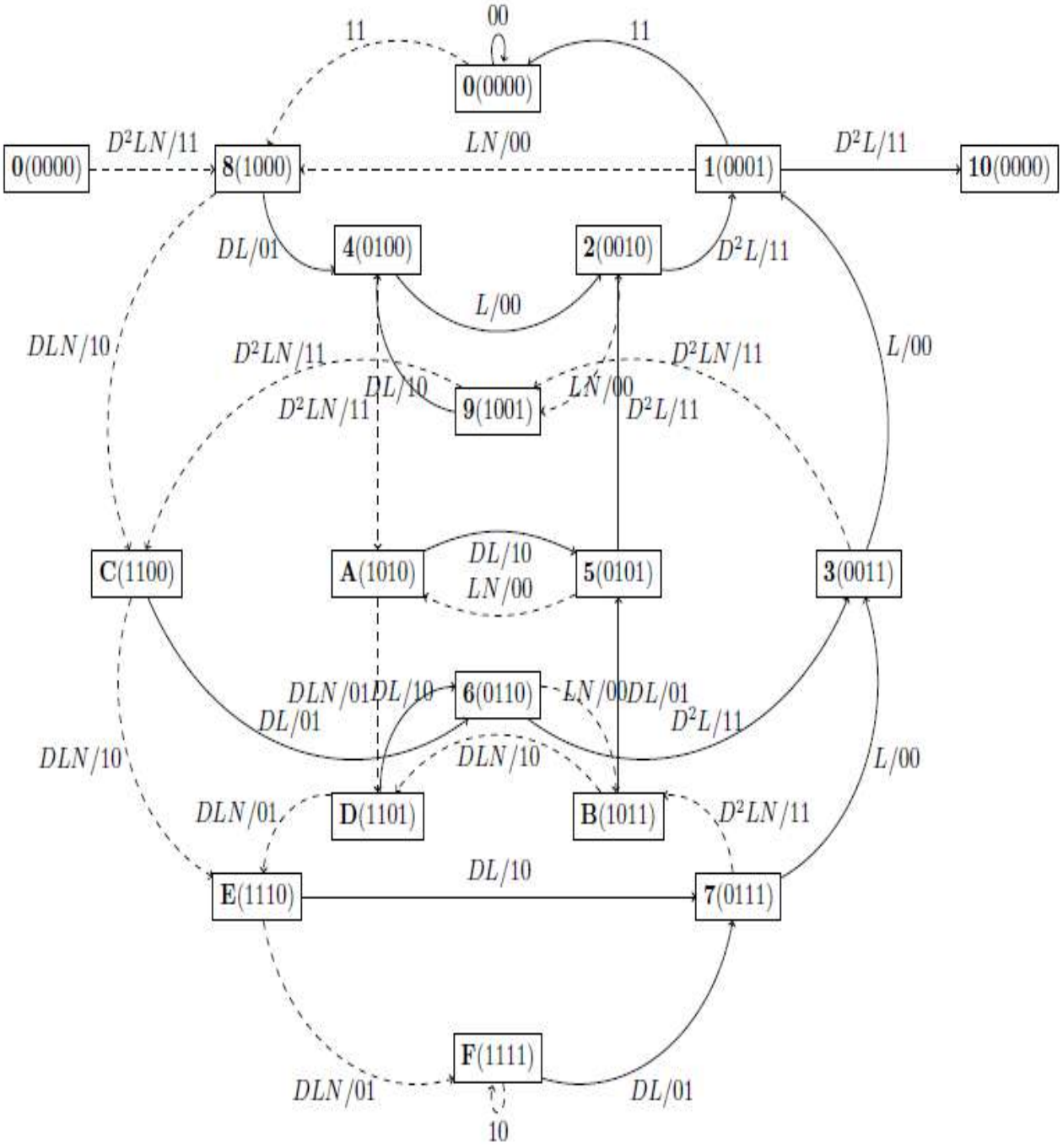



**Figure III. 2: Diagramme d'état**

Le diagramme en treillis qui en découle, est donné en figure III.3.

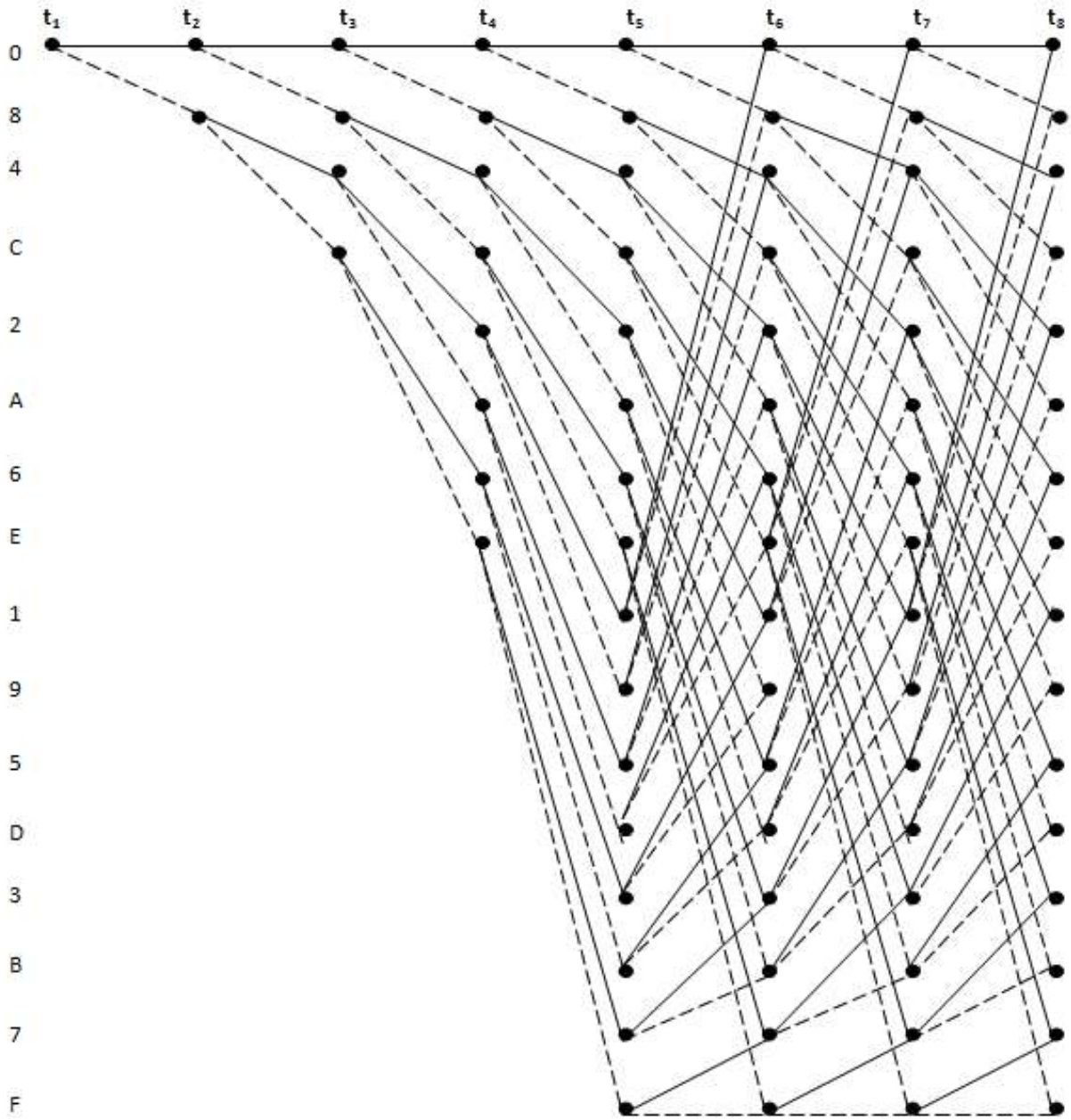



**Figure III. 3: Diagramme en treillis.**

### III.1.3 Codage convolutionnel sur le TMS320C54 :

Le codage convolutionnel des données est efficacement accompli sur l'architecture du DSP de la famille C54x. C'est dû à son implémentation hardware de l'unité d'opérations ACS et ses possibilités de traitement en double-mot.

#### III.1.3.1 Edition de lien :

La mémoire interne du DSP C5410 constituée d'un ensemble de cellules à 16 bits chacune (word), se compose de : ROM = 16 kW, DARAM (Dual Access) = 8 kW, SARAM

(Single Access) = 56 kW. La configuration par défaut de la mémoire interne est donnée en figure III.4 [4]

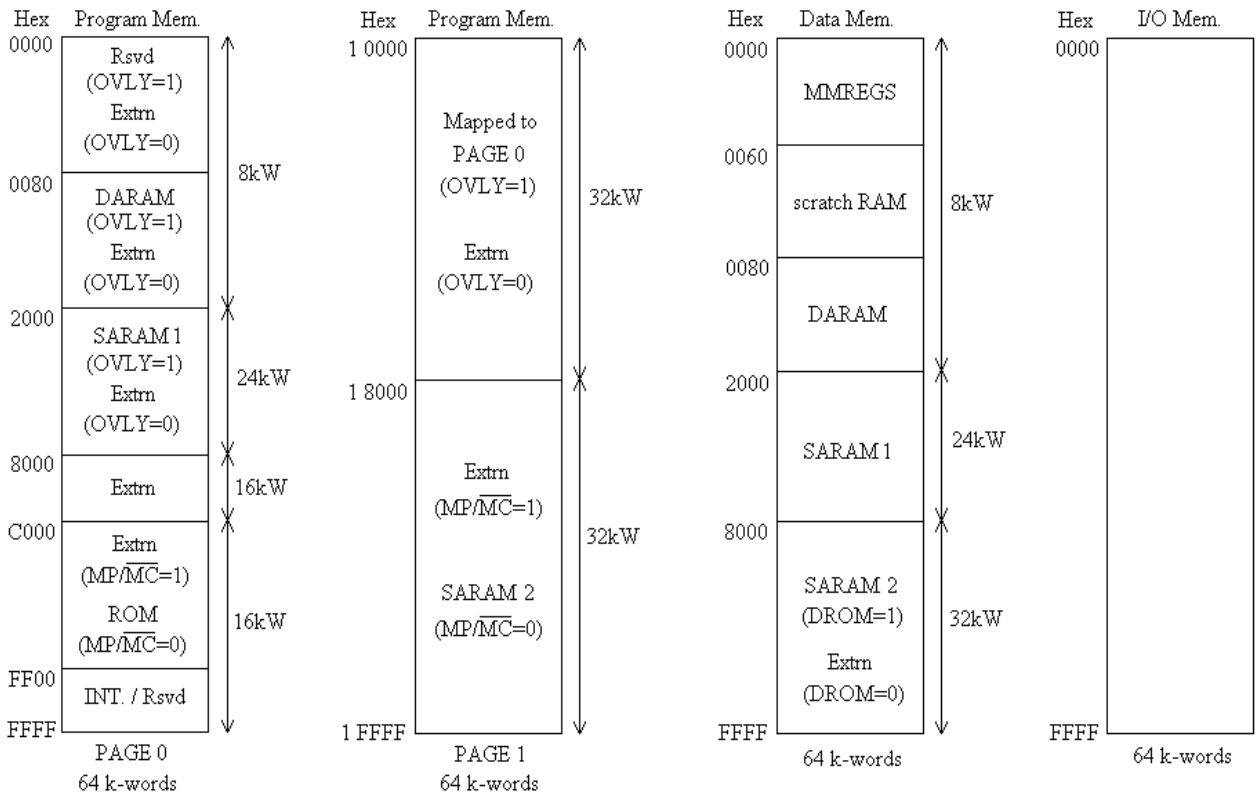



**Figure III. 4: Configuration de la mémoire interne du C5410**

L'éditeur de lien profite des ressources internes voire externes du DSP, en espace mémoire disponible. Il attribue la zone d'adresses [8000h à FF00h] à la mémoire-programme en ROM, et la zone d'adresses [80h à 7FFFh] à la mémoire-données en DARAM et SARAM1.

```
MEMORY

{

   PAGE 0:       ROM     (RXI):   origin = 0x8000,  len = 0x7F00

   PAGE 1:       BUFFER (RWI):  origin = 0x0080,  len = 0x8

                        RAM     (RWI):  origin = 0x0088,  len = 0x7F70

}

SECTIONS

{

   .text   :          {} > ROM          PAGE 0
```

```
        .bss   :    {} > BUFFER    PAGE 1

        .data  :    {} > RAM       PAGE 1

}
```

### III.1.3.2 Programme en assembleur de l'encodeur :

La programme principal en assembleur génère deux codes de sortie à 16 bits chacun, pour chaque mot d'entrée. Les mots d'entrées ont été choisis comme étant 1, 2, 3, 4. Pour le codage convolutionnel des données d'entrée, le programme appelle deux sous-programmes successivement, codeur et alternance.

```
                .include "Encodeur.asm"
                .global _main
                .data                   ; section renfermant les données (à 16 bits) à coder
data_in         .word 0                 ; 1ère donnée additionnelle toujours nulle
                .word 1                 ; ici commence la 1ère donnée
                .word 2
                .word 3
                .word 4
shift           .word 16                ; sauvegarde du décalage pour le registre T
                .text
_main:          call codeur             ; appel du sous-programme de codage
                call alternance         ; sous-programme d'alternance G0G1 des bits codés
                nop
                nop
                .end
```

Le programme réserve de l'espace mémoire ou 'buffer' aux données d'entrée, dont la taille est 4×32bits. De même, il réserve deux fois cet espace pour les données codées de sortie suite à son débit ½.

```
                .mmregs
                .global  data_in                ; données d'entrée à coder, définis en programme principal
Nwords          .set 4                          ; taille du buffer d'entrée à 32-bits :
                                                ; (0,1),(1,2),(2,3),(3,4) = nbre_data = 4 (x32)
data_code       .bss     dataCode, 2*Nwords     ; table des données codées en sortie : 4*(U0,U1)
                .asg     AR2, PointeurIN
                .asg     AR3, PointeurOUT
                .txt
```

Le sous-programme de codage après configuration de la mémoire interne, commence le codage convolutionnel en effectuant les opérations XOR appropriées sur les données d'entrée rassemblées en paquets de 32 bits. Il fournit en sortie, les mots de codes $U_0$ et $U_1$, table III.2. Ces mots de code sont arrangés de façon consécutive de 16 bits, en prenant la sortie $U_0$ comme mot MSB.

| U0 | U1 |
|---|---|
| 0001 | B003 |
| 6003 | 6002 |
| 5004 | D006 |

**Table III. 2: Sorties codées après un premier traitement**

```
;-------------------------------------------------------------------------------
; sous-programme du codage convolutionnel (cas GSM)
```

```
;-------------------------------------------------------------------------------

codeur:     ldm     PMST, A             ; chargement du contenu du registre PMST
            and     #0FFD7h, A          ; OVLY=0 : in-RAM appartient à la mémoire-données
            stlm    A, PMST             ; DROM=0 : in-ROM appartient à la mémoire-programme
            stm     #data_in, PointeurIN    ; PointeurIN -> données d'entrée à coder
            stm     #data_code, PointeurOUT ; PointeurOUT -> données codées en sortie
            stm     #Nwords-1, BRC      ; compteur de boucle sur Nwords-1
            rptb    endloop-1           ; répétition pour chaque entrée à 16-bit
            ld      *PointeurIN+, 16, A ; charge A_haut par la donnée data(0) à 16-bit
            or      *PointeurIN, A      ; charge A_bas par la donnée suivante data(1) à 16-bit
            ld      A, B                ; B = A
            xor     B, -3, A            ; A = X(n) XOR X(n-3)
            xor     B, -4, A            ; A = X(n) XOR X(n-3) XOR X(n-4) = U0
            stl     A, *PointeurOUT+    ; sauvegarde U0, PointeurOUT sur la prochaine sortie
            xor     B, -1, A            ; A = X(n) XOR X(n-1) XOR X(n-3) XOR X(n-4) = U1
            stl     A, *PointeurOUT+    ; sauvegarde U1, PointeurOUT sur la prochaine sortie
endloop     ret
```

Le sous-programme alternance, remet en forme les données codés $U_0$ et $U_1$ de manière à assurer leur alternance au niveau du bit. L'entrelacement binaire est effectué en décalant bit par bit dans l'accumulateur A, les mots $U_0$ et $U_1$ situés successivement dans l'accumulateur B. Les codes ainsi traités constituent les sorties $G_0G_1$ à transmettre.

```
;-------------------------------------------------------------------------------

; sous-programme de la mise en forme alternée (U0,U1) des bits codés

;-------------------------------------------------------------------------------

            .asg    AR2, Compteur       ; compteur de boucle externe
```

```
alternance:   stm    #data_code, PointeurOUT ; PointeurOUT -> données codées en sortie
              stm    #Nwords-1, Compteur     ; compteur de boucle sur Nwords
loopA         stm    #15, BRC                ; compteur de boucle (16 fois)
              rptb   loopB-1                 ; répétition pour chaque entrée à 16-bit
              ld     *(shift), T             ; T = shift (par adressage absolue)
              ld     *PointeurOUT+, TS, B    ; B_haut = U0
              sftl   B, 1                    ; C = MSB_B
              rol    A                       ; LSB_A = C
              ld     *PointeurOUT-, TS, B    ; B_haut = U1
              sftl   B, 1                    ; C = MSB_B
              rol    A                       ; LSB_A = C
              addm   #1, *(shift)            ; incrémentation du décalage (par adressage absolue)
loopB         dst    A, *PointeurOUT+        ; sauvegarde de G0G1 dans l'ancien emplacement
              st     #16, *(shift)           ; "st" est sur plus de 2 mots
              banz   loopA, *Compteur-       ; delayed impossible avec "st" après branchement
              ret
```

## III.2. Implémentation sur DSP de l'algorithme de Viterbi :

L'algorithme de Viterbi (VA) fournit une méthode pour minimiser le nombre des séquences de symboles de données (chemins de treillis). Comme un décodeur à maximum de vraisemblance, le VA identifie la séquence des codes avec la plus haute probabilité, correspondant à la séquence transmise sur la base de la séquence reçue.

Le VA est composé d'une mise à jour métrique et d'une routine de rétro-traçage. Dans la mise à jour métrique, les probabilités sont accumulées pour tous les états en fonction du symbole d'entrée courant. La routine de retraçage reconstruit les données, une fois un chemin à travers le treillis est identifié. Un bref pseudo code des principales étapes de la VA est illustré en figure III.5 [4].

```
pour chaque trame :
  {
   initialiser les métriques.
   pour chaque symbole:
       {
        mise à jour métrique ou Ajout-Compare-Sélectionne (ACS)
        pour chaque état de retard :
             {
             Calculer la distance locale d'entrée pour chaque chemin possible
             Accumuler la distance locale pour chaque chemin
             Sélectionner et stocker la distance minimale
             Sauvegarder l'indication du chemin pris
             }
        }
   rétrotraçage
   pour chaque bit dans une trame :
       {
       Calculer la position dans la transition de données de l'état actuel
       Lire le bit sélectionné correspondant à l'état
       Mise à jour de la valeur d'état avec un nouveau bit
        }
   inverser l'ordre des bits de sortie
   }
```

**Figure III. 5: Pseudo-Code pour l'algorithme de Viterbi**

### III.2.1. Mise à jour métrique :

Bien qu'un état est entré pour chaque symbole transmis, le VA doit calculer l'état précédent le plus probable pour tous les états possibles, ceci est dû à ce que l'état actuel de l'encodeur n'est pas connu jusqu'à ce qu'un nombre de symboles est reçu.

Chaque état de retard est lié aux états précédents par un ensemble de tous les chemins possibles. Pour des encodeurs à taux 1/n, il n'y a que deux chemins à partir de chaque état de retard. Ceci limite considérablement les calculs.

Le chemin d'état est estimé en combinant la valeur d'entrée courante et les métriques cumulées des états précédents. Étant donné que chaque chemin a un symbole associé (ou point de constellation) [3], la distance locale de symbole à partir de l'entrée courante est calculée. Pour une meilleure estimation de la validité des données, la distance locale est ajoutée aux distances cumulées de l'état à qui le chemin pointe.

Comme chaque état dispose de deux ou de plusieurs chemins d'entrée possibles, la distance accumulée est calculée pour chaque chemin d'entrée. Le chemin à la distance minimale accumulée est sélectionné comme le chemin survivant. Cette sélection de la séquence la plus probable est la clé de l'efficacité de VA. En rejetant la plupart des chemins, le nombre de chemins possibles est réduit au minimum.

Une indication du chemin et de l'état de retard précédent est stockée afin de permettre la reconstruction de la séquence d'état à partir du dernier. La distance minimale accumulée est stockée pour utilisation dans le prochain intervalle de symbole. Il s'agit de la mise à jour métrique qui se répète pour chaque état. La mise à jour métrique est aussi appelée opération d'ajout-comparaison-sélection (ACS): **A**ccumulation de la distance des données, **C**omparaison des chemins d'entrée, et **S**élection du trajet à maximum de vraisemblance.[5]

### III.2.2. Rétro-traçage :

Après un certain nombre de symboles, égal environ à quatre ou cinq fois la longueur de contrainte, une petite précision est obtenue avec des entrées supplémentaires.

Après quatre ou cinq fois la langueur de contrainte, l'état dont la distance accumulé minimale, peut être utilisé pour lancer le rétro-traçage. Une méthode plus précise consiste à attendre une trame entière de donnée reçue avant le début de rétro-traçage. Dans ce cas, des bits de queue sont ajoutés pour forcer le treillis à l'état zéro, fournissant un point connu pour commencer le rétro-traçage.

Une valeur de 1 en position de bit indique que l'état précédent est via le trajet inférieur, et un 0 indique que l'état précédent est via le chemin supérieur. Chaque état antérieur est construit en déplaçant la valeur de transition dans le LSB de l'état [3]. Cette opération est répétée pour chaque intervalle de symbole jusqu'à ce que la séquence des états soit reconstruite. Comme ces états de retard représentent directement les sorties actuelles, c'est une opération simple de reconstruire les données originales à partir de la séquence des états. Dans la plupart des cas, les bits de sortie sont dans un ordre inversé, étant donné que le rétro-traçage fonctionne de la fin jusqu'au début.[6]

## III.3. Comparaison entres les décisions soft et hard :

La méthode utilisée pour calculer les distances locales dépend de la représentation des données reçues. Si les données sont représentées par un seul bit, ce sont des données à décision hard et les mesures de la distance de Hamming sont utilisées. Lorsque les données sont représentées par plusieurs bits, ce sont alors des données à décision soft, et les mesures de la distance Euclidienne sont utilisées.

| Valeur | Signification |
|---|---|
| 011 | Valeur la plus confidente |
| 010 | |
| 001 | |
| 000 | Valeur positive la moins confidente |
| —— | Valeur Nulle |
| 111 | Valeur la moins confidente |
| 110 | |
| 101 | |
| 100 | Valeur négative la plus confidente |

**Table III. 3: Valeurs à décision soft.**

Ces valeurs de décision soft viennent généralement d'un égaliseur, qui réduit l'interférence entre symboles. Ceci produit des valeurs de confidence basées sur les différences entre les données reçues et les données attendues [3].

## III.4. Calcul de la distance locale :

Avec des entrées HD de décision hard, la distance locale utilisée est la distance de Hamming. Elle est calculée en additionnant les différences de bits individuels entre les données reçues et prévues. Avec les entrées de décision soft, la distance Euclidienne est généralement utilisée. Elle est définie, pour le débit 1/*c*, par:

$$\text{distance_local(j)} = \sum_{n=0}^{c-1} \left[ SD_n - U_n(\text{j}) \right]^2 \qquad \text{(III.3)}$$

Où $SD_n$ sont les entrées de décision soft, $u_n$(j) sont les entrées attendues pour chaque état du chemin, *j* est un indicateur de la trajectoire, et *c* est l'inverse du taux de codage. Cette

mesure de distance est le carré de la longueur du vecteur, de dimension *c*, des données reçues aux données attendues [3].

L'expression III.3 s'écrit encore :

$$\text{distance_local(j)} = \sum_{n=0}^{c-1}\left[ SD^2{}_n - 2SD_nU_n(\text{j}) + U_n{}^2(\text{j}) \right] \qquad \text{(III.4)}$$

Afin de minimiser la distance accumulée, nous sommes préoccupés par les parties de l'équation qui sont différentes pour chaque trajet. Les termes $\sum_{n=0}^{c-1} SD^2{}_n$ et $\sum_{n=0}^{c-1} U_n{}^2(\text{j})$ sont les même pour tous les chemins et ils peuvent être éliminés, ce qui réduit l'équation à:

$$\text{distance_local(j)} = -2\sum_{n=0}^{c-1} SD_nU_n(\text{j}) \qquad \text{(III.5)}$$

Ce qui veut dire que le max de (III.5) assure le min de (III.3).

Ainsi, comme la distance locale est une valeur négative, la valeur minimale se produit lorsque la distance locale est maximale. Le premier scalaire -2 est supprimé, et les maximums sont recherchés dans la procédure de mise à jour métrique. Cette équation est la somme des produits des valeurs reçues et prévu, bit par bit. Pour le taux de codage ½, on a [3] : $\text{distance_locale(j)} = SD_0U_0(\text{j}) + SD_1U_1(\text{j})$ (III.6)

La dépendance de $U_n$ sur les chemins du treillis est déterminée par les polynômes du codeur. Inversement, les valeurs de $SD_n$ représentent les données reçues qui n'ont aucune dépendance avec l'état actuel. Le calcul de la distance locale diffère en fonction du nouvel état en cours d'évaluation.

Les $U_n$(j) sont codés comme des valeurs signés antipodales, ce qui signifie que 0 correspond à +1 et 1 correspond à -1. Cette représentation permet aux équations d'être encore réduites à une simple somme et différence des données reçues. Pour un taux de système 1/n, il y a seulement $2^n$ distances locales uniques à chaque intervalle de symbole. Avec, la moitié de

ces distances locales qui sont simplement l'inverse de l'autre moitié. Ainsi, uniquement $2^{n-1}$ valeurs doivent être calculées ou stockés [3].

## III.5. Code TMS320C54x de décodage de Viterbi

Le code sur TMS320C54x pour le décodage de Viterbi peut être divisé en trois parties: initialisation, mis à jour métrique et rétrotraçage. Ces segments de code, avec modifications légères, sont utilisés sur des systèmes avec différentes longueurs de contrainte, de taille des trames et de taux de codage.

### III.5.1 Initialisation :

Avant le commencement du décodage de Viterbi, une initialisation est nécessaire.

Le mode de traitement est configuré avec:

• mode extension de signe (SXM = 1)

• mode accumulateur dual 16 bits (C16 = 1), afin de permettre simultanée la mis à jour métrique de deux chemins du treillis.

Les besoins en tampons et pointeurs, sont définies par :

• Tampon d'entrée

• Tampon de sortie

• Table de transition

• Stockage métrique (des tampons circulaires doivent être utilisés).

Les valeurs métriques sont initialisées.

Le compteur de répétition de bloc est chargé avec un nombre de bits de sortie (-1) pour la mise à jour métrique.

Le tampon de données d'entrée est le double des données à décoder (à cause du débit ½). Le tampon de sortie contient des valeurs à un seul bit pour chaque période de symbole. Ces bits

sont concaténés, de sorte qu'ils nécessitent un tampon linéaire de taille moitié de celui d'entrée.

La taille de la table de transition est déterminée par la longueur de contrainte, $2^{K-1}/16$ mots de 16 bits. Soit, dans notre cas, un seul mot de 16 bits.

Le stockage métrique nécessite deux tampons, chacun des deux avec une taille égale au nombre d'états ($2^{K-1}$=16 mots). Pour minimiser la manipulation de pointeur, ces tampons sont généralement configurés comme un seul tampon circulaire [3]. Mais dans notre cas, et en tenant compte de certaines restriction du DSP employé, nous sommes amené à utiliser quatre tampons circulaires égaux, de taille 8 mots de 16 bits chacun, figure III.6.

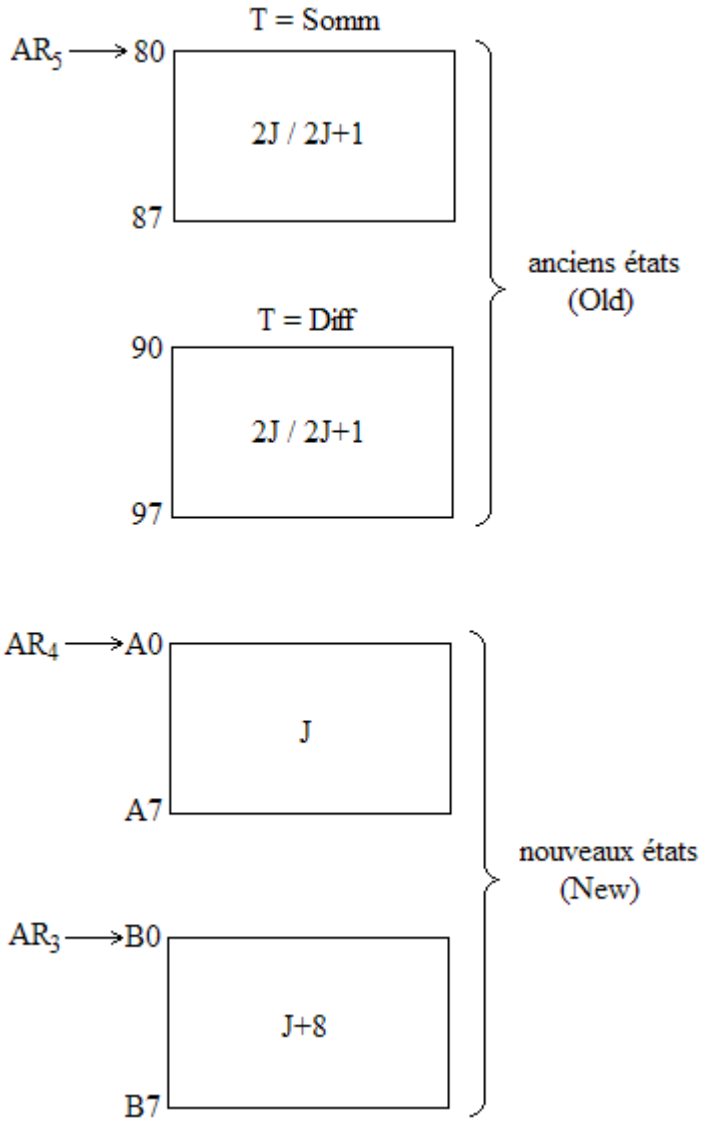



**Figure III. 6: quatre buffers pour la mise à jour métrique des états du treillis**

L'éditeur de lien configure le programme en ROM et les données en RAM. C'est dans cette dernière partie de la mémoire, que sont logés les quatre buffers des anciens et nouveaux états du treillis.[8]

```
MEMORY

{

  PAGE 0:    ROM   (RXI): origin = 0x8000,  len = 0x7F00
   PAGE 1:    table1  (RWI):          origin = 0x80,    len = 8
              table2  (RWI):          origin = 0x90,    len = 8
              table3  (RWI):          origin = 0xA0,    len = 8
              table4  (RWI):          origin = 0xB0,    len = 8
              RAM1 (RWI):             origin = 0xB8,    len = 0xA
              RAM2          (RWI):origin = 0xC2,    len = 0x7F50
}
SECTIONS
{
 .text          : {} > ROM    PAGE 0
        "buff1"         : {} > table1   PAGE 1
        "buff2"         : {} > table2   PAGE 1
        "buff3"         : {} > table3   PAGE 1
        "buff4"         : {} > table4   PAGE 1
        .data   : {} > RAM1   PAGE 1
        .bss    : {} > RAM2   PAGE 1
}
```

Cela implique aussi la configuration du registre de répétition de bloc (BK) à huit, du registre d'index (AR0) à deux pour la manipulation de données en dual-memory (2×16 bits). Pour ce type d'accès mémoire, les registres indirects $AR_5$, $AR_4$, $AR_3$, doivent être utilisés comme pointeurs afin de permettre leur repositionnement automatique pour l'intervalle de symbole suivant dans des buffers circulaires. Les quatre tampons sont utilisés dans la mise à jour métrique, avec deux tampons pour l'ancienne métrique accumulée, pointés par AR5, et un tampon pour la nouvelle métrique, pointé par AR4, et un autre pour la nouvelle métrique, pointé par AR3, figure III.6.

Tous les états, sauf l'état 0, sont mis à la même valeur métrique initiale. L'état 0 est l'état de démarrage et exige une valeur initiale différente [3]. L'état 0 est généralement mis à une valeur 0, tandis que tous les autres états sont définis pour la valeur la plus négative (que nous avons fixée à 0xC000), offrant une marge de croissance pour les mises à jour métriques.

### III.5.2 La mise à jour métrique des états du treillis :

La plupart du temps de calcul est consacré à la mise à jour métrique, puisque tous les états doivent être mis à jour à chaque intervalle de symbole. Par conséquent, beaucoup d'efforts ont été déployés pour minimiser le temps de calcul de la mise à jour métrique. L'unité compare-sélectionne-stocke (CSSU ou ACS) est incorporé dans l'architecture C54x pour simplifier cette procédure.

En examinant le treillis de la figure III.3, on constate qu'une fois sa périodicité atteinte, il est composé à chaque fois de $2^{K-2} = 8$ cellules en papillon, figure III.7. Ainsi, une certaine symétrie inhérente dans la structure de treillis peut être utilisée pour simplifier les calculs. Les chemins d'état partant des deux anciens états sont complémentaires. Les chemins d'état arrivant aux deux nouveaux états sont identiques. Cette symétrie est une fonction des polynômes du codeur.[5]

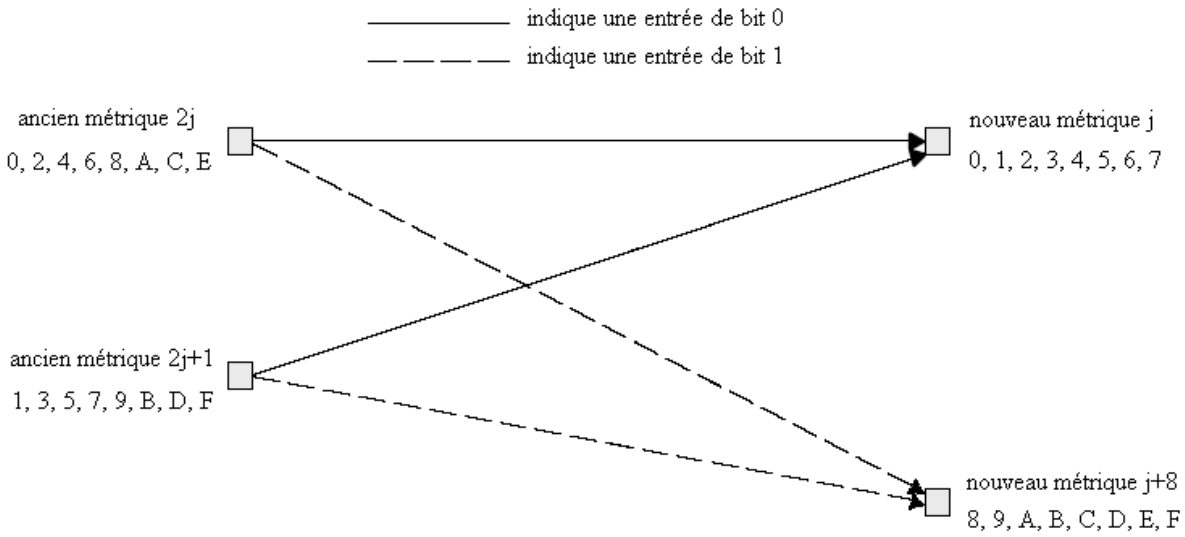



**Figure III. 7: Cellule en papillon du treillis propre au codeur convolutionnel GSM**

Ces symétries fournissent des méthodes pour simplifier la procédure de mise à jour métrique:

• Seulement une seule mesure de la distance locale est nécessaire pour chaque papillon, elle est alternativement ajoutée et soustraite pour chaque nouvel état

• Les métriques cumulées antérieures (anciennes valeurs métriques) sont les mêmes pour les mises à jour des deux nouveaux états, en minimisant les manipulations d'adresses.

Pour ces raisons et pour satisfaire les latences de pipeline, la mise à jour métrique est habituellement réalisée sur les papillons. Les huit cellules en papillon partagent les mêmes distances locales. Une distance locale est donc calculée et stockée avant la mise à jour métrique des 8 cellules (ou 16 états du treillis). Cette distance locale est mise dans le registre T qui n'a pas à être chargé pour chaque papillon.

Chaque tampon de la mise à jour métrique est sur 8 mots. Les anciennes métriques 2j et 2j+1, sont consultées dans l'ordre consécutif 0, 1, 2, 3,…, par le pointeur AR5. Les nouvelles métriques j et j+8, sont mises à jour dans l'ordre 0, 8, 1, 9,…, et utilisent les deux pointeurs $AR_4$ et $AR_3$ à cette fin. A la fin de la mise à jour métrique, ces tampons sont échangés, de telle sorte que les métriques récemment mises à jour deviennent les anciennes métriques pour l'intervalle de symbole suivant.

En plus des métriques de tampons, le registre de transition (TRN) doit être stocké. Un seul bit par état suffit pour indiquer le chemin survivant en court. Un mot de mémoire est requis pour chaque 16 états propres à un intervalle de symbole.

La table III.4 donne la procédure de mise à jour métrique pour le décodeur convolutionnel utilisé dans la norme global des systèmes de communications mobiles (GSM). Deux macros, BFLY_DIR et BFLY_REV, automatisent les calculs de papillon. La somme et la différence réfèrent à la distance locale. New et Old réfèrent à la métrique courante et précédente pour un état donné. Le registre TRN indique à l'aide de chacun de ses 16 bits, l'une des 2 branches retenue avec, TRN(0)=0 pour branche entrante supérieure ou TRN(0)=1 pour branche entrante inférieure, ce qui aboutit à une branche survivante parmi 2 d'un nouvel êta. [6]

| Opération | calcul |
|---|---|
| Calcule de la distance locale | Temps (0)=$HD_0$- $HD_1$=diff<br>Temps (1)=$HD_0$+ $HD_1$=sum |
| Chargement de registre T | T= Temp(1) |
| BFLY_DIR (ect) | New(0)=max[Old(0)+sum, Old(1)-sum] comparaison du<br>New(8)=max[Old(0)-sum, Old(1)+sum] cumul métrique |
| BFLY_REV (erse) | New(1)=max[Old(2)-sum, Old(3)+sum]<br>New(9)=max[Old(2)+sum, Old(3)-sum] |
| BFLY_DIR | New(2)=max[Old(4)+sum, Old(5)-sum]<br>New(10)=max[Old(4)-sum, Old(5)+sum] |
| BFLY_REV | New(3)=max[Old(6)+sum, Old(7)+sum]<br>New(11)=max[Old(6)+sum, Old(7)-sum] |
| Chargement de register T | T=Temp(0) |
| BFLY_DIR | New(4)=max[Old(8)+diff, Old(9)-diff]<br>New(12)=max[Old(8)-diff, Old(9)+ diff] |
| BFLY_REV | New(5)=max[Old(10)-diff, Old(11)+ diff]<br>New(13)=max[Old(10)+diff, Old(11)- diff] |
| BFLY_DIR | New(6)=max[Old(12)+diff, Old(13)-diff]<br>New(14)=max[Old(12)-diff, Old(13)+diff] |
| BFLY_REV | New(7)=max[Old(14)-diff, Old(15)+diff]<br>New(15)=max[Old(14)+diff, Old(15)-diff] |
| Stockage de<br>Registre de transition | Trans(i)=TRN Qui renferme les chemins survivants des 8 papillons (BFLY) |

**Table III. 4: Les opérations de mise à jour métrique du décodage de Viterbi en GSM.**

Après que les métriques dans un seul intervalle de symbole sont mises à jour, les pointeurs des tampons métriques sont mis à jour aussi pour l'itération suivante. Étant donné que les tampons métriques doivent être circulaires, ceci est accompli en utilisant la syntaxe de la mise à jour du pointeur, * ARn +0% [6].

Les calculs impliqués dans les quatre étapes de la mise à jour métrique du suivi d'un état, sont donc :

1. Calculer la distance locale de l'entrée à chaque chemin possible :

La distance locale peut être décrite comme une somme de produit, expression (III.6), à un taux de codage 1/2. Il s'agit d'une simple procédure d'addition-comparaison-sélection.

Comme seulement les $2^{n-1}$ distances locaux doivent être calculées pour un taux 1/n, à cause du fait que la moitié est l'inverse de l'autre moitié, les distances locales inverses sont accommodées par la soustraction dans l'accumulation totale de la distance.

```
                    .asg    AR5, Etats_2J_2J1   ; pointeur sur le buffer des anciens états (2j et 2j+1)
                    .asg    AR4, Etat_J         ; pointeur sur le buffer des nouveaux états (j)
                    .asg    AR3, Etat_J8        ; pointeur sur le buffer des nouveaux états (j+8)

                    .text

cellules            .macro  ; accumule dist. totale pour chaque chemin et sélectionne celles minimales
                    nop                         ; régle un problème de latence
                    cell_dir                    ; appel cellule avec j=0 ou j=4
                    cell_inv                    ; appel cellule avec j=1 ou j=5
                    cell_dir                    ; appel cellule avec j=2 ou j=6
                    cell_inv                    ; appel cellule avec j=3 ou j=7
                    ldm     Etats_2J_2J1, A     ; pointeur d'anciens états dans A
                    .endm

cell_dir  .macro    ; butterfly direct
                    DADST  *Etats_2J_2J1, A     ; A = OLD_Met(2*j)+T // OLD_(2*j+1)-T
                    DSADT  *Etats_2J_2J1+0%, B  ; B = OLD_Met(2*j)-T // OLD_(2*j+1)+T
                    CMPS    A, *Etat_J+%
                ;NEW_Met(j) = Max[Old_Met(2*j)+T, Old_Met(2*j+1)-T]
                ; TRN = TRN << 1                 on a donc TRN[0]=0: branche entrante sup.
                ; Si (Old_Met(2*j)+T =< Old_Met(2*j+1)-T) Alors TRN[0]=1: branche entrante inf.
                    CMPS   B, *Etat_J8+%
                 ;NEW_Met(j+8)=Max[Old_Met(2*j)-T, Old_Met(2*j+1)+T]
                ; TRN = TRN << 1                 on a donc TRN[0]=0: banche entrante sup.
                ; Si (Old_Met(2*j)-T =< Old_Met(2*j+1)+T) Alors TRN[0]=1: branche entrante inf.
                    .endm

cell_inv .macro     ; butterfly inverse
                    DSADT *Etats_2J_2J1, A      ; A = OLD_Met(2*j)-T // OLD_(2*j+1)+T
                    DADST  *Etats_2J_2J1+0%, B  ; B = OLD_Met(2*j)+T // OLD_(2*j+1)-T
                    CMPS    A, *Etat_J+%
                ;NEW_Met(j = Max[Old_Met(2*j)-T, Old_Met(2*j+1)+T]
                ; TRN = TRN << 1                 on a donc TRN[0]=0: branche entrante sup.
                ; Si (Old_Met(2*j)-T =< Old_Met(2*j+1)+T) Alors TRN[0]=1: branche entrante inf.
                    CMPS   B, *Etat_J8+%
                ;NEW_Met(j+8)=Max[Old_Met(2*j)+T, Old_Met(2*j+1)-T]
                ; TRN = TRN << 1                 on a donc TRN[0]=0: branche entrante sup.
                ; Si (Old_Met(2*j)+T =< Old_Met(2*j+1)-T) Alors TRN[0]=1: branche entrante inf.
                    .endm
```

2. Accumuler la distance totale pour chaque trajet :

Grâce à ses deux accumulateurs et instructions spécialisées, le C54x peut accumuler des métriques pour deux trajets en un seul cycle, si la distance locale est stockée dans le registre T. La double instruction d'addition-soustraction, DADST, ajoute le registre T à une valeur de la mémoire et enregistre le résultat dans la moitié inférieure de l'accumulateur, simultanément, soustrait le registre T de la valeur du prochain emplacement mémoire et stocke le résultat dans la moitié supérieure de l'accumulateur. DSADT effectue le calcul complémentaire.

3. Sélectionner et enregistrer la distance minimale

4. Enregistrer l'indication du chemin parcouru

Ces deux étapes sont effectuées en un seul cycle, à l'aide de l'instruction spécialisée du C54x, CMPS (ou compare-sélecte-stocke) qu'utilise le CSSU en assurant de :

• Comparer les deux valeurs de 16 bits signés dans les moitiés supérieure et inférieure de l'accumulateur.

• Conservez la valeur maximale dans la mémoire

• Indiquer la valeur maximale par le bit de test et contrôle (TC), tout en le décalant dans le registre de transition (TRN).

Ceci permet de sélectionner la métrique cumulée minimale, et indique le chemin d'accès associé à cette valeur. Les valeurs d'état précédentes ne sont pas stockées; elles sont reconstruites dans la routine de rétro-traçage à partir du registre de transition [3].

Ce qui suit est le code implémenté sur C5410 pour la mise à jour des états du treillis, suite au traitement de deux mots de code à l'entrée du décodeur.

```
; B/ Mise à jour Métrique par Add-Compare-Select ACS (calcul des états de chaque bit d'entrée de code)
;_____________________________________________________________________
; 1. Calcul de la distance locale de chaque chemin possible du treillis :
;          données reçues en hard-decision (HD0=U0, HD1=U1) = bits individuels issus d'un démodulateur
;          données reçues en soft-decision  (SD0, SD1) = bits multiples issus d'un égaliseur
;          notre cas considère des codes reçus en hard-decision avec débit de code 1/2
;          4 possibilités de distance locale avec G0(j)=G1(j)=+/-1 (où bit0=+1, bit1=-1),
;          soit  +/-(U0+U1) = +/-somm,  +/-(U0-U1) = +/-diff
```

```
                    .asg    AR0, indice             ; utilisé en adressage indéxé
                    .asg    AR1, versCode           ; utilisé en adressage indirect
                    .asg    AR2, offset             ; utilisé en adressage indirect
                    .asg    AR7, compteur           ; utilisé comme compteur
                    .asg    AR6, paquet             ; utilisé comme compteur

decodeur:           stm     #Nwords/2-1, compteur   ; compteur de boucle, moitié des codes d'entrée
                    ldm     PMST, A                 ; chargement du contenu du registre PMST

                    and     #0xFFD7, A      ; OVLY=0 : in-RAM appartient à la mémoire-données
                    stlm    A, PMS          ; DROM=0 : in-ROM appartient à la mémoire-programme
                    ssbx    SXM             ; validation de l'extension de signe
                    ssbx    C16             ; mode Dual 16-bit
                    stm     #0x7F00, SP     ; initialisation de la pile au fond de la RAM2
                    stm     #8, BK          ; pour 4 buffers circulaires (8 mots par buffer)
                    stm     #2, indice      ; pour adressage indexé
                    stm     #buffer1, Etats_2J_2J1  ; pointe sur le buffer des anciens états (2j et 2j+1)
                    stm     #buffer3, Etat_J        ; pointe sur le buffer des nouveaux états (j)
                    stm     #buffer4, Etat_J8       ; pointe sur le buffer des nouveaux états (j+8)
                    stm     #code, versCode         ; pointe sur les données d'entrée à décoder

; ici on évalue le métrique pour 2 mots de code: (2 x 16 = 32bits) > (5 x longueur_contrainte = 25bits)

loop1               stm     #1, paquet          ; compteur par paquets de 2 mots de codes (soit 32 bits)
loop2               stm     #transition, offset ; pointe sur la sauvegarde des contenus du registre TRN
                    stm     #7, BRC             ; compteur sur les 16/2 couples de bits (U0,U1) de code
                    rptb    loop3-1
                    pshm    offset              ; *transition -> SP
                    ld      *(decale), T        ; T = decale (par adressage absolue)
                    stm     #distance, offset   ; pointe sur les distances locales (somm et diff)
                    ld      *versCode, TS, B
                    ld      B, -1, A
                    and     #1, A               ; masque bit LSB:  A = bit(U0)
                    and     #1, B               ; masque bit LSB:  B = bit(U1)
                    stl     A, *(tempo)         ; tempo = A
                    sub     B, A                ; A = bit(U0) - bit(U1)
                    stl     A, *offset+         ; offset(0) = diff, pointeur sur offset(1)
                    add     *(tempo), B         ; B = bit(U0) + bit(U1)
                    stl     B, *offset          ; offset(1) = somm
                    ld      *offset-, T         ; T = somm, pointeur sur offset(0)
                    cellules                    ; mise à jour métrique de 4 cellules avec j=0,1,2,3
                    ld      *offset, T          ; T = diff, pointeur reste sur offset(0)
                    add     #16, A
                    stlm    A, Etats_2J_2J1     ; pointeur sur le prochain buffer des anciens états
                    popm    offset              ; récupération de l'adresse de transition
                    cellules                    ; mise à jour métrique de 4 cellules avec j=4,5,6,7
                    addm    #2, *(decale)       ; incrémentation du décalage (par adressage absolue)
                    sub     #16, A
                    stlm    A, Etats_2J_2J1     ; pointeur sur le buffer précédent des anciens états
                    st      TRN, *offset+       ; TRN pour un bit d'entrée (8 TRN par mot d'entrée)

; permutation de pointeurs des buffers d'état (les états récents devenant anciens):
                    ldm     Etats_2J_2J1, A     ; A = AR5
                    mvmm    Etat_J, Etats_2J_2J1    ; AR5 = AR4
                    stlm    A, Etat_J           ; AR4 = A
                    add     #16, A
                    stlm    A, Etat_J8          ; AR3 = A
loop3               st      #-14, *(decale)
                    ld      *versCode+, A       ; juste pour mise à jour du pointeur
                    pshm    Etat_J              ; *Etat_J -> SP
```

```
        pshm    Etat_J8              ; *Etat_J8 -> SP
        stm     #transition+8, Etat_J8   ; pointe sur la 2ème moitié d'espace de transition
        stm     #transition, Etat_J      ; pointe sur la 1ère moitié d'espace de transition
        rpt     #7
        mvdd    *Etat_J+, *Etat_J8+
        popm    Etat_J8              ; SP -> *Etat_J8
        popm    Etat_J               ; SP -> *Etat_J
        banz    loop2, *paquet-      ; puis sortir, après 2 mots de codes traités
```

### III.5.3 La fonction de rétro-traçage :

Le rétrotraçage exige moins de traitement que la mise à jour métrique, puisque seulement un bit par intervalle de symbole est généré à la sortie du décodeur de Viterbi. Les étapes de calcul sont :

1. Calculer la position dans la donnée de transition de l'état courant :

La mise à jour métrique stocke dans le registre de transition un bit par état de retard, indiquant le chemin survivant. En raison de l'ordre dans lequel les états sont actualisés, les données de transition stockées sont mélangées. La fonction principale de l'algorithme de rétrotraçage est d'extraire le bit correct à partir des données de transition, pour chaque intervalle de symbole. Si les papillons sont actualisés dans l'ordre consécutif, les données de transition pour un intervalle de symbole sont stockées comme montré dans la table III.8.

| | ***Nombre de bits dans le mot de transition TRN*** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| état correspondant | 0 | 8 | 1 | 9 | 2 | A | 3 | B |

| | ***Nombre de bits dans le mot de transition TRN*** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| état correspondant | 4 | C | 5 | D | 6 | E | 7 | F |

**Table III. 5: Ordre des états dans la donnée de transition pour un intervalle de symbole**

La recherche du numéro de bit correct dans le mot de transition exige la considération de la méthode d'extraction de bit par le C5410. L'instruction BITT (bit de test spécifié par le registre T) spécifie le numéro de bit d'une façon inverse; c'est-à-dire, l'adresse de bit correspond à 15–TREG. L'adresse de bit calculée doit correspondre à cette notation.

L'adresse de bit est trouvée en décalant le bit de l'état d'un pas à gauche et en posant le MSB comme LSB. Ceci peut être exprimé comme :

$$Num_Bit = 2 \times Etat + [Etat > (K-2)] \& 1 \qquad avec \ K=5 \qquad \text{(III.7)}$$

Ce numéro est alors chargé dans le registre T pour la prochaine étape. Bien que cette valeur peut être plus grande que 16, l'instruction BITT n'emploie que les quatre LSBs du TREG. Il n'y a aucun besoin de masquer les bits supérieurs.

2. Lecture du bit choisi correspondant à l'état :

L'instruction BITT copie le bit choisi dans le bit de TC.

3. Actualiser la valeur d'état avec le nouveau bit.

L'instruction ROLTC décale l'accumulateur de un bit vers la gauche et décale le bit de TC dans l'accumulateur comme LSB. Cette valeur devient le nouvel état, utilisé dans la prochaine itération [3]. L'algorithme de rétrotraçage extrait les bits de sortie dans une boucle de 16, permettant au bit unique de chaque itération d'être combiné dans un mot de 16 bits. Lorsque toutes les entrées dans la table de transition sont traitées, les bits dans chaque mot de sortie sont alors inversés dans l'ordre pour représenter les sorties du début à la fin [3].

Le code pour la routine de rétrotraçage et de réarrangement de bit est comme suit :

```
;_____________________________________________________
; C/ Fonction Rétro-Trace (décodage de 2 mots à la fois)
;_____________________________________________________
K                       .set    5                   ; longueur de contrainte du codeur convolutionnel
; sélection de l'état à plus faible métrique parmi les derniers états mis à jour
trace                   rsbx    TC                  ; indicateur de la boucle pour les étas j: TC=0
                        stm     #6, BRC             ; compteur sur les récents états j
                        st      #7, *(tempo)        ; tempo initialisé à la 1ère valeur : BRC + 1
                        ld      *Etats_2J_2J1+%, A  ; commencement par l'état j=0
                        rptb    loop4-1
                        sub     *Etats_2J_2J1+%, A, B   ; B = A - *Etats_2J_2J1
                        xc      2, BLT              ; exécute l'instruction suivante si B<0
                        mvmd    BRC, *(tempo)       ; sauvegarde du compteur courant
                        xc      2, BLT              ; exécute les 2 instructions suivantes si B<0
                        ld      *Etats_2J_2J1-%, A  ; juste pour décrémenter le pointeur d'état
                        ld      *Etats_2J_2J1+%, A  ; sauvegarde du plus grand état métrique
loop4                   ldm     Etats_2J_2J1, B     ; pointeur d'état dans B
```

```
                add     #16, B
                stlm    B, Etats_2J_2J1          ; pointeur au prochain buffer des anciens états
                stm     #7, BRC                  ; compteur sur les états j+8 les plus récents ..
                rptb    loop5-1                  ; .. devenus juste anciens
                sub     *Etats_2J_2J1+%, A, B    ; B = A - *Etats_2J_2J1
                xc      2, BLT                   ; exécute l'instruction suivante si B<0
                mvmd    BRC, *(tempo)            ; sauvegarde du compteur courant
                xc      2, BLT                   ; exécute les 2 instructions suivantes si B<0
                ld      *Etats_2J_2J1-%, A       ; juste pour décrémenter le pointeur d'état
                ld      *Etats_2J_2J1+%, A       ; sauvegarde du plus grand état métrique
                xc      1, BLT                   ; exécute l'instruction suivante si B<0
                ssbx    TC                       ; flag de boucle des étas j+8: TC=1
loop5           ldm     Etats_2J_2J1, B          ; pointeur d'état dans B
                sub     #16, B
                stlm    B, Etats_2J_2J1          ; pointeur sur sa position initiale
                xc      1, NTC
                ld      #7,      A               ; A=7 si TC=0
                xc      1, TC
                ld      #15, A                   ; A=15 si TC=1
                sub     *(tempo), A              ; A = A - BRC (n° d'état à plus faible métrique)
                rsbx    OVM                      ; reset du mode overflow (dépassement permis)
                stm     #decode, paquet          ; address pointer for output bits (end of buffer)
                stm     #transition, offset      ; table de transition (à 16 TRN)

; décodage à rebours des données à partir du chemin survivant (1mot de sortie pour 2 mots d'entrée)
                stm     #15, BRC                 ; nombre de bit à décoder dans un mot
                rptb    loop6-1                  ; boucle au nombre des TRN traités
                ld      *paquet, B
                roltc   B                        ; TC en LSB
                stl     B, *paquet               ; un bit décodé dans le paquet decode

; Calcul de la position du bit dans TRN (les 4 instr. suivantes exécutent l'équation (III.7)) :
                sftl    A, -(K-2), B             ; B = A>>(K-2)  décalage -3 positions (à droite)
                and     #1, B                    ; B = B&1 = MSB de l'état
                add     A, 1, B                  ; B = B+(A<<1) = 2*état + MSB de l'état
                stlm    B, T                     ; T = B (position bit)
                nop                              ; latence avant utilisation de T
                bitt    *offset+                 ; copie du bit correspondant du mot TRN dans TC
                sftl    A, 1                     ; A=2A
                and     #0xF, A                  ; A = état antérieur 2j
                xc      2, TC                    ; si TC=1; branche inférieure indiquant ..
                add     #1, A                    ;... A = état antérieur 2j+1

; si A=j implique bit=0, si A=j+8 implique bit=1
                sub     #7, A, B                 ; B = A - 7
                nop                              ; latence avant test de B
                xc      1, BLEQ                  ; si B <= 0 ..
                rsbx    TC                       ; .. bit décodé : TC = 0
                xc      1, BGT                   ; si B > 0 ..
                ssbx    TC                       ; .. bit décodé : TC = 1
loop6           banz    loop1, *compteur-        ; retour de boucle pour la lecture des mots codés
                nop
                nop
                .end
```

## Conclusion

La génération des DSPs TMS320C54x permet une implémentation avec performances élevées du décodage de Viterbi. Le papillon du treillis peut être calculé à un rythme d'environ trois cycles par papillon, par l'utilisation d'accès dual en mémoire et traitement dual des états du treillis tout en bénéficiant d'un traitement rapide par l'unité ACS, câblée hardware.

# Conclusion générale

## Conclusion

Dans une étude préliminaire, nous avons montré que les états du diagramme en treillis d'un codeur convolutionnel sont conçus simplement par les bits actuels dans le registre à décalage du codeur, tandis que les chemins entre états représentent les symboles qui sont émis par le codeur. Cependant, chaque colonne d'état de retard dans le treillis indique un intervalle de symbole.

Les données codées par convolution sont décodées grâce à la connaissance des transitions d'états possibles dans le treillis, générées à partir de la dépendance du symbole courant aux données passées. Le processus de décodage estime la séquence des états de retard, sur la base des symboles de données reçues, pour reconstituer un chemin à travers le treillis. Les états de retard représentent directement les données codées, puisque les états correspondent aux bits dans le registre à décalage du codeur. La correspondance entre les données et les états permet la reconstruction aisée des données, une fois que le chemin à travers le treillis soit unique et bien déterminé (chemin survivant).

Pour notre implémentation du codeur correcteur d'erreur, ce dernier est pris sur un exemple concret du système de communication mobile GSM, qui s'est vu implémenté sur le DSP TMS320C5410. L'implémentation a touché donc un encodeur de longueur de contrainte 5 et de débit de codage ½, ainsi que son décodeur basé sur l'algorithme de Viterbi. Le code assembleur correspondant a été amélioré et optimisé, en progression avec son développement et sa vérification continuelle.

Dans la réalisation de notre implémentation, plusieurs difficultés ont été rencontrées. La première est dans l'installation sur nos Pcs, du 'code composer studio' du DSP, qui est le logiciel de développement des codes en destinée du C5410. Après un temps perdu non négligeable, nous nous sommes résignés à travailler au labo d'informatique.

Certaines données sont prises comme exemple dans notre implémentation du codeur-décodeur convolutionnel qui une fois codées, n'ont pu être décodées correctement. Ceci est dû en partie au temps limité qui nous est imparti, à l'égard de la difficulté de programmation en assembleur sur le DSP cible. Nous pensons que cet échec peut être dû à l'une ou

plusieurs raisons suivantes, dont on cite : le mauvais choix des données d'essai (1,2,3,4) dont plusieurs états du treillis présentaient la même métrique faible (impossibilité d'avoir un unique chemin survivant), malgré qu'on a respecté la théorie qui dit que les données de treillis convergent sur un chemin optimal après approximativement cinq fois la longueur de contrainte. D'autres causes possibles ont été recensées et restent à vérifier pour dépanner le programme de décodage.

## Bibliographie :

[1] : B. SKLAR, « DIGITAL COMMUNICATIONS, Fundamentals and Applications », Prentice Hall PTR, Second Edition.

[2] : J. A. Heller, I. W. Jacob, "Viterbi decoding for satellite and space communication" IEEE Trans. Commun. Techno, vol.19, n°5, 1971.

[3] : H. Hendrix, "Viterbi Decoding Techniques for the TMS320C54x DSP Generation", Member Group Technical Staff, Application Report, 2002.

[4] : Texas Instruments, "TMS320C54x, Assembly Language Tools, User's Guide", SPRU 102E, 2001.

[5] : Texas Instruments, "TMS320C54x DSP, Reference Set", SPRU 172B, 1998.

[6] : Texas Instruments, "TMS320C54x DSP, CPU and Peripherals", SPRU 131F, 1999.

[7] : S. Haykin, « Communication Systems », Wiley & Sons, Inc., 2001.

[8] : J. G. Proakis, « Digital Communications », Fourth edition, McGraw Hill inc., 2001.

[9] : B. P. Lathi, " Modern Digital and Analog Communication Systems", Oxford University Press, 1998.

# Abréviations

## Abréviation :

DSP…… Digital Signal Processing – Traitement numérique des signaux.

ACS……Ajouter-Comparer-Sélectionner.

DC……Démodulateur codé.

VA……. Viterbi algorithm—Algorithme de Viterbi.

TMS..... Member, Group Technical Staff.

GSM.....Global System Mobile.

AUL……...Unité Arithmétique et Logique.

AWGN…….Additive White Gaussian Noise—Le bruit additive blanc de gaus

## Annexe

Le programme principal de l'encodeur convolutionnel :

```
compiler sous l'option "non auto-initialization"
; ouvrir la fenêtre des registres (mode flottant) et la fenêtre mémoire (sur data_code)
; lancer avec "Restart" en mode pas à pas: touche F11 (lent) ou F10 (rapide).
                    .include "Encodeur.asm"
                    .global _main
                    .data                   ; section renfermant les données (à 16 bits) à coder
data_in             .word 0                 ; 1ère donnée additionnelle toujours nulle
                    .word 1                 ; ici commence la vraie 1ère donnée
                    .word 2
                    .word 3
                    .word 4
shift               .word 16        ; sauvegarde du décalage pour le registre T

                    .text
_main:
                    call codeur             ; appel sous-programme de codage
                    call alternance ; sous-programme d'alternance G0G1 des bits codés
                    nop
                    nop
                    .end
```

Les sous-programmes de codage convolutionnel:

sous-programme du codage convolutionnel (cas GSM).

```
codeur:     ldm     PMST, A             ; chargement du contenu du registre PMST
            and     #0FFD7h, A      ; OVLY=0 : in-RAM appartient à la mémoire-données
```

```
                    stlm    A, PMST          ; DROM=0 : in-ROM appartient à la mémoire-programme
                    stm     #data_in, PointeurIN      ; PointeurIN -> données d'entrée à coder
                    stm     #data_code, PointeurOUT ; PointeurOUT -> données codées en sortie
                    stm     #Nwords-1, BRC                    ; compteur de boucle sur Nwords-1
                    rptb    endloop-1                         ; répétition pour chaque entrée à 16-bit
                    ld      *PointeurIN+, 16, A       ; charge A_haut par la donnée data(0) à 16-bit
                    or      *PointeurIN, A          ; charge A_bas par la donnée suivante data(1) à 16-bit
                    ld      A, B                              ; B = A
                    xor     B, -3, A                    ; A = X(n) XOR X(n-3)
                    xor     B, -4, A                    ; A = X(n) XOR X(n-3) XOR X(n-4) = U0
                    stl     A, *PointeurOUT+          ; sauvegarde U0, PointeurOUT sur la prochaine sortie
                    xor     B, -1, A                ; A = X(n) XOR X(n-1) XOR X(n-3) XOR X(n-4) = U1
                    stl     A, *PointeurOUT+        ; sauvegarde U1, PointeurOUT sur la prochaine sortie
endloop             ret
```

sous-programme de la mise en forme alternée (U0,U1) des bits codés

```
.asg    AR2, Compteur                           ; compteur de boucle externe

alternance:         stm     #data_code, PointeurOUT         ; PointeurOUT -> données codées en sortie
                    stm     #Nwords-1, Compteur              ; compteur de boucle sur Nwords
loopA               stm     #15, BRC                         ; compteur de boucle (16 fois)
                    rptb    loopB-1                       ; répétition pour chaque entrée à 16-bit
                    ld      *(shift), T                    ; T = shift (par adressage absolue)
                    ld      *PointeurOUT+, TS, B           ; B_haut = U0
                    sftl    B, 1                           ; C = MSB_B
                    rol     A                              ; LSB_A = C
                    ld      *PointeurOUT-, TS, B           ; B_haut = U1
                    sftl    B, 1                           ; C = MSB_B
                    rol     A                              ; LSB_A = C
                    addm    #1, *(shift)                   ; incrémentation du décalage (par adressage absolue)
loopB               dst     A,      *PointeurOUT+          ;sauvegarde de G0G1 dans l'ancien emplacement
                    st      #16, *(shift)                  ; "st" est sur plus de 2 mots
                    banz    loopA, *Compteur-              ; delayed impossible avec "st" après
branchement
                    ret
```

Le programme principal de décodage Viterbi :

```
.mmregs
                    .global code                ; code d'entrée à décoder, définis en programme principal

buffer1             .sect "buff1"               ; 1er buffer métrique
                    .word 0                     ; métrique de l'état "0" initialisé à 0
                    .loop 7                     ; les métriques des autres états du treillis sont...
                    .word 0xC000 ;              ; .. initialisés avec valeur à mi-intervalle négatif
                    .endloop                    ; (valeur assez négative avant saturation)

                    .asg 2 , x                  ; assigne à X la valeur 2 dans la boucle
                    .loop 3                     ; boucle d'assemblage pour 3 autres buffers
buffer:x:.sect "buff:x:"
                    .loop 8                             ; boucle d'assemblage d'initialisation d'un buffer
                    .word 0xC000
                    .endloop
                    .eval x+1, x                        ; x+1 dans x, incrémentation de x
                    .endloop

                    .data
Nwords              .set 8                              ; taille du buffer des codes d'entrée à 16-bits
```

```
decale          .word -14                       ; initialisation du décalage à droite de bit d'un mot de code
tempo           .byte 0                         ; pour la sauvegarde temporaire d'un bit

decode          .bss dataDec, Nwords/2          ; table des données décodées en sortie
distance .bss distLoc, 2                        ; pour la sauvegarde de somm et diff comme distance locale
transition      .bss trans, 16                  ; pour la sauvegarde des registres de transition de 2 mots
;-----------------------------------------------------------------------------------------------------------------------
A/ Initialisation
;-----------------------------------------------------------------------------------------------------------------------
.asg    AR5, Etats_2J_2J1       ; pointeur sur le buffer des anciens états (2j et 2j+1)
                        .asg    AR4, Etat_J                     ; pointeur sur le buffer des nouveaux états (j)
                        .asg    AR3, Etat_J8            ; pointeur sur le buffer des nouveaux états (j+8)

                        .text

cellules  .macro                                        ; accumule dist. totale pour chaque chemin et sélectionne celles minimales
                        nop                                                     ; régle un problème de latence
                        cell_dir                        ; appel cellule avec j=0 ou j=4
                        cell_inv                        ; appel cellule avec j=1 ou j=5
                        cell_dir                        ; appel cellule avec j=2 ou j=6
                        cell_inv                        ; appel cellule avec j=3 ou j=7
                        ldm             Etats_2J_2J1, A         ; pointeur d'anciens états dans A
                        .endm

cell_dir  .macro                                        ; butterfly direct
                        DADST  *Etats_2J_2J1, A         ; A = OLD_Met(2*j)+T // OLD_(2*j+1)-T
                        DSADT  *Etats_2J_2J1+0%, B      ; B = OLD_Met(2*j)-T // OLD_(2*j+1)+T
                        CMPS    A, *Etat_J+%    ;NEW_Met(j) = Max[Old_Met(2*j)+T, Old_Met(2*j+1)-T]
                                                                ; TRN = TRN << 1
                on a donc TRN[0]=0: branche entrante sup.
                        ; If (Old_Met(2*j)+T =< Old_Met(2*j+1)-T) Then TRN[0]=1: branche entrante inf.
                        CMPS    B, *Etat_J8+%           ;NEW_Met(j+8)=Max[Old_Met(2*j)-T, Old_Met(2*j+1)+T]
                                ; TRN = TRN << on a donc TRN[0]=0: branche entrante sup.
                                ; If (Old_Met(2*j)-T =< Old_Met(2*j+1)+T) Then TRN[0]=1: branche entrante inf.
                        .endm

cell_inv .macro                                         ; butterfly inverse
                        DSADT *Etats_2J_2J1, A          ; A = OLD_Met(2*j)-T // OLD_(2*j+1)+T
                        DADST *Etats_2J_2J1+0%, B       ; B = OLD_Met(2*j)+T // OLD_(2*j+1)-T
                        CMPS    A, *Etat_J+%            ;NEW_Met(j)=Max[Old_Met(2*j)-T, Old_Met(2*j+1)+T]
                                        ; TRN = TRN << 1                on a donc TRN[0]=0: branche entrante sup.
                                ; If (Old_Met(2*j)-T =< Old_Met(2*j+1)+T) Then TRN[0]=1: branche entrante inf.
                        CMPS    B, *Etat_J8+%   ; NEW_Met(j+8) = Max[Old_Met(2*j)+T, Old_Met(2*j+1)-T]
                                        ; TRN = TRN << 1                on a donc TRN[0]=0: branche entrante sup.
                                ; If (Old_Met(2*j)+T =< Old_Met(2*j+1)-T) Then TRN[0]=1: branche entrante inf.
                        .endm
; 1. Calcul de la distance locale de chaque chemin possible du treillis :
;               données reçues en hard-decision (HD0=U0, HD1=U1) = bits individuels issus d'un démodulateur
;               données reçues en soft-decision data (SD0, SD1) = bits multiples issus d'un égaliseur
;               notre cas considère des codes reçus en hard-decision avec débit de code 1/2
;              4 possibilités de distance locale avec G0(j)=G1(j)=+/-1 (où bit0=+1, bit1=-1),
;   soit  +/-(U0+U1) = +/-somm,  +/-(U0-U1) = +/-diff

                        .asg    AR0, indice
                        .asg    AR1, versCode
                        .asg    AR2, offset
                        .asg    AR7, compteur
                        .asg    AR6, paquet
```

```
decodeur:       stm     #Nwords/2-1, compteur   ; compteur de boucle externe sur la moitié du nombre de codes d'entrée
                ldm     PMST, A                 ; chargement du contenu du registre PMST

                and     #0xFFD7, A              ; OVLY=0 : in-RAM appartient à la mémoire-données
                stlm    A, PMST                 ; DROM=0 : in-ROM appartient à la mémoire-programme
                ssbx    SXM                     ; validation de l'extension de signe
                ssbx    C16  ; mode Dual 16-bit pour mise à jour métrique simultanée de 2 chemins_treillis
                stm     #0x7F00, SP     ; initialisation de la pile au fond de la RAM2
                stm     #8, BK    ; pour 4 buffers circulaires (8 mots par buffer d'anciens et nouveaux états)
                stm     #2, indice          ; pour adressage indexé
                stm     #buffer1, Etats_2J_2J1    ; pointe sur le buffer des anciens états (2j et 2j+1)
                stm     #buffer3, Etat_J          ; pointe sur le buffer des nouveaux états (j)
                stm     #buffer4, Etat_J8               ; pointe sur le buffer des nouveaux états (j+8)
                stm     #code, versCode                 ; pointe sur les données d'entrée à décoder
; ici on évalue le métrique pour 2 mots de code: (2 x 16 = 32bits) > (5 x longueur_contrainte = 25bits)
loop1           stm     #1,     paquet          ; compteur par paquets de 2 mots de codes (soit 32 bits)
loop2           stm     #transition, offset     ; pointe sur la sauvegarde des contenus du registre TRN
                stm     #7, BRC             ; compteur sur les 16/2 couples de bits (U0,U1) d'un mot de code
                rptb    loop3-1
                pshm    offset                  ; *transition -> SP
                ld              *(decale), T    ; T = decale (par adressage absolue)
                stm     #distance, offset       ; pointe sur les distances locales (somm et diff)
                ld              *versCode, TS, B
                ld              B, -1, A
                and             #1, A           ; masque bit LSB:  A = bit(U0)
                and             #1, B           ; masque bit LSB:  B = bit(U1)
                stl     A, *(tempo)             ; tempo = A
                sub     B, A                    ; A = bit(U0) - bit(U1)
                stl     A, *offset+                             ; offset(0) = diff, pointeur sur offset(1)
                add             *(tempo), B                     ; B = bit(U0) + bit(U1)
                stl     B, *offset                              ; offset(1) = somm
                ld              *offset-, T                     ; T = somm, pointeur sur offset(0)
                cellules                                        ; mise à jour métrique de 4 cellules avec j=0,1,2,3
                ld              *offset, T                      ; T = diff, pointeur reste sur offset(0)
                add             #16, A
                stlm    A, Etats_2J_2J1         ; pointeur sur le prochain buffer des anciens états
                popm    offset                                  ; récupération de l'adresse de transition
                cellules                                        ; mise à jour métrique de 4 cellules avec j=4,5,6,7
                addm    #2, *(decale)           ; incrémentation du décalage (par adressage absolue)
                sub             #16, A
                stlm    A, Etats_2J_2J1         ; pointeur sur le buffer précédent des anciens états
                st              TRN, *offset+           ; TRN pour un bit d'entrée (8 TRN par mot d'entrée)
; permutation de pointeurs des buffers d'état (les états récents devenant anciens):
                ldm             Etats_2J_2J1, A         ; A = AR5
                mvmm    Etat_J, Etats_2J_2J1    ; AR5 = AR4
                stlm    A, Etat_J                       ; AR4 = A
                add             #16, A
                stlm    A, Etat_J8                      ; AR3 = A
loop3           st              #-14, *(decale)
; se positionner après loop3 pour les essais avec "ctrl+F10" pour exécuter jusqu'au curseur
                ld              *versCode+, A           ; juste pour mise à jour du pointeur sur les codes d'entrée
```

```
                        pshm    Etat_J                          ; *Etat_J -> SP
                        pshm    Etat_J8                         ; *Etat_J8 -> SP
                        stm     #transition+8, Etat_J8  ; pointe sur la 2ème moitié d'espace de transition
                        stm     #transition, Etat_J             ; pointe sur la 1ère moitié d'espace de transition
                        rpt             #7
                        mvdd    *Etat_J+, *Etat_J8+
                        popm    Etat_J8                         ; SP -> *Etat_J8
                        popm    Etat_J                          ; SP -> *Etat_J
                        banz    loop2, *paquet-         ; puis sortir, avec 2 mots de codes traités
                        nop
                        nop
                        banz    loop1, *compteur-       ; à mettre après la routine derétrotraçage(traceback)
; C/ Traceback Function (décodage de 2 mots à la fois, donc terminer la boucle du "compteur")
; Traceback routine
; For the following example,
; A accumulator = State value
; B accumulator = temp storage
K                       .set    5                               ; constraint length

                        RSBX    OVM                             ; turn off overflow mode (dépassement permis)
  ;                       STM     #TRANS_END, AR2  ; address of end of transition table (dans AR2) cas GSM, la table a un seul mot de sauvegarde de TRN
;                       STM     #NBWORDS-1, AR1                 ; number of output words to compute (dans AR1)
                        MVMM            AR1, AR4                        ; Copy AR1 in AR4 (to use in output reversal)
;                       STM     #OUTPUT+NBWORDS-1, AR3  ; address pointer for output bits (end of buffer)
                        LD              #0, A           ; A contains initial State value (State=0 in this example)
                        STM     #15, BRC                ;Load block repeat counter for inner loop do i=0,NBWORDS-1
BACK            RPTB    TBEND-1         ; do j=0,15 répétition BRC fois, du bloc d'instr. terminé avant l'étiq. TBEND
; Calculate bit position in transition word  les 4 instr. suivantes exécutent l'équation (7) :
                        SFTL    A, -(K-2), B                    ; B = A>>(K-2)  décalage -3 veut dire 3 positions à droite
                        AND     1, B                            ; B = B&1 = msb of State
                        ADD     A, 1, B                         ; B = B+A<<1 = 2*State + msb of State
                        STLM    B, T                            ; T = B (bit position)

                        BITT    *AR2                            ; Test bit in transition word,
                        ROLTC A                                 ; Rotate decision in A
                                                ; enddo (j loop) fin du traitement d'un intervalle de symbole
TBEND           STL     A, *AR3-                ; Store packed output contenant donc les états ordonnés old(2j), old(2j+1)
                        BANZD           BACK, *AR1-             ; repeat j loop if frame not finished
                        STM     #15, BRC                ; Init block repeat counter for next word
                                                        ; enddo (i loop)
; Reverse order of bits within words to output in correct order (car Traceback opère depuis la fin du treillis et remonte vers son début)
                        MAR     *AR3+                           ; Get start of output buffer
                             LD         *AR3, A  ; load first word into A états finaux du dernier symbole (chargement depuis la fin du treillis)
RVS                     SFTA    A, -1, A                ; A>>1,  A[0] -> C
                        STM     #15, BRC                ; éliminer l'instruction similaire précédente !
                        RPTB    RVS2-1                          ; Do i=0,15
                        ROL     B                                       ; B<<1, B[0] = C
                        SFTA    A, -1, A                ; A>>1,  A[0] ->C
; enddo
RVS2            BANZD           RVS, *AR4-      ; done with all words? exécution des 2 instr. suivantes avant BANZ
                        STL     B, *AR3+                ; save just completed word remis dans sa case de départ
                        LD              *AR3, A         ; load next word en remontant vers le début du treillis
```