ECOLE NATIONALE POLYTECHNIQUE D'ALGER

Algorithme de VITERBI à déciaion ferme à atratégie de Trace-Back

Maater 2 Electronique

ZOUAOUI Massiva Yamina

Encadrée Par : Mrs ABDELOUEL et TAGHI

01/06/2012

ECOLE NATIONAL POLYTECHNIQUE D'ALGER

Thèse réalisée au Laboratoire Communication
Ecole Nationale Polytechnique
Avenue Hacen Badi
El harrach 16200, Alger
Algérie
Tél : +213 21 52 53 01/03
Fax : +213 21 52 29 73

Web : http ://www.enp.edu.dz/

Sous la direction de Mohammed TAGHI

Co-encadrement Lahcen ABDELOUEL

ملخص

لقد حاولنا تجريب و تطبيق مفكك Viterbi ذو القرار الصعب على بطاقة، هو مفكك مستغل أو مستعمل داخل مستقبل ضمن أنظمة الاتصالات الرقمية عندما تكون المعلومة مشفرة بمشفر ملفف.

هذه الخوارزمية عبارة عن مفكك قناة و يستعمل لكشف و تصحيح الأخطاء الواردة. افترضنا مشفر ملفف يحتوي قيد طوله $L = 4$ و مردوده $R = {}^{1}/_{2}$ معرف بكثري حدود $g_2 = 13_8$ و $g_2 = 15_8$ أنجزنا تجارب على MATLAB لهذا الطول و لأي طول، كذلك كتابة الخوارزمية بلغة VHDL حيث بينا منه النتائج.

Résumé

Nous avons simulé et tenté d'implémenter un décodeur de Viterbi à décision ferme sur une carte FPGA. Il s'agit d'un décodeur utilisé dans les récepteurs des systèmes de communication numériques lorsque l'information est codée par un code convolutionnel. Cet algorithme concerne le décodage de canal et sert donc à la détection et à la correction des erreurs au sein de l'information reçue. Nous avons supposé un codeur convolutionnel de longueur de contrainte $L = 4$ et de rendement $R = {}^{1}/_{2}$ défini par les polynôme générateurs $g_1 = 13_8$ et $g_2 = 15_8$. Nous avons effectué des simulations sur MATLAB pour cette même longueur mais aussi pour une longueur quelconque. Nous avons aussi tenté de réaliser un programme VHDL dont nous avons exposé l'avancement et les résultats.

Abstract

We have simulated and tried to implement, on FPGA circuit, a hard decision Viterbi decoder. It is used to decode received information in digital communication systems when the information is coded by a convolutionnal code. This algorithm has a canal decoding purpose and is therefore used to detect and correct errors within the received information. We have assumed a convolutionnal coder of constraint length $\mathrm{L} = 4$. We have performed MATLAB simulations for this same length, but also for nondescript length. Besides, we tried to create a VHDL program, which we have outlined the progress and results.

Remerciements

Nous remercions nos parents qui nous ont beaucoup encouragés le long de ce projet. Et à l'issu de nos études faites au sein de l'École Nationale Polytechnique nous voudrions rendre un hommage tout particulier à : Nos promoteurs M. M.Taghi et M. L.Abdelouel qui nous ont honorés en acceptant de nous encadrer. Tous les enseignants du département d'Électronique de l'École Nationale Polytechnique. Tous les enseignants du département des Sciences Fondamentales. Nous remercions tous ceux qui ont participé de prés ou de loin à l'aboutissement de nos études.

Dédicace

A celle qui ma accompagnée et défendue depuis ma plus tendre enfance et que j'aime plus que tout. A ma *sœur*.

A celle qui a consacré tout son temps, son énergie et sa patience à me donner une bonne éducation. A *ma mère.*

A celui qui ma indiqué le chemin à suivre en me rappelant que le travail et la volonté font de l'être humain ce qu'il est. A mon *père.*

Je dédie aussi ce mémoire à tous mes amis et collègues de travail qui m'ont énormément aidés, je pense tout particulièrement à Celia et Rabah.

Merci.

citation :

« L'erreur n'annule pas la valeur de l'effort accompli. »

de Proverbe africain

Massiva Yamina ZOUAOUI

Table des matières

Table des figures

Liste des tableaux

Liste des symboles

ACS Add Compare Select

ACSU Add Compare Select Unit

ARQ Automatic Repeat reQuest

AWGN Additive White Gaussian Noise

BMU Branch Metric Unit

BPSK Binary Phase Shift Keying

BSC Binary Symmetric Channel

CDMA Code Division Multiple Access

DU Decision Unit

DVB Digital Video Broadcasting

EGPRS Enhanced General Packet Radio Service

FSK Frequency Shift Keying

GSM Global System for Mobile communications

LAN Local Area Network

LIFO Last In First Out

MAP Maximum A Posteriori

MCU Metrics Calculating Unit

ML Maximum Likelihood

MPSK M-ary Phase Shift Keying

NRZ Non Return to Zero

PMU Path Metric Unit

QAM Quadrature Amplitude Modulation

QPSK QuadriPhase Shift Keying

RAM Random Access Memory

RE Register Exchange

ROM Read Only Memory

SISO Soft Input Soft Output

SNR Signal to Noise Ratio

SOVA Soft-Output Viterbi Algorithm

TB Trace Back

UMTS Universal Mobile Telecommunications System

UMTS Universal Mobile Telecommunications System

VA Viterbi Algorithm

VD Viterbi Decoder

WiFi Wireless Fidelity

Introduction

Cadre général et objectifs

L'histoire des transmissions a été marquée par plusieurs évolutions. Après les pigeons voyageurs, l'information est écrite sur un support et transmise au destinataire ; le courrier est toujours utilisé dans notre vie quotidienne.

Avec l'arrivée de l'électricité, on a pu commencer à transporter des messages sous une forme moins matérielle, à des vitesses beaucoup plus importantes. De nos jours, nous assistons à une forte augmentation des besoins en termes de communication numérique notamment pour les applications mobiles. La communication à distance (téléphone, télévision, satellite, etc. . .), entre machines et usagers nécessite des canaux de transmission acheminant l'information. Les canaux utilisés sont en général loin d'être parfaits, la différence entre eux est la manière ou la probabilité de modifier une information en une autre.

Parmi les critères de développement des nouveaux standards de communication tels que l'UMTS, la qualité de transmission est devenue un facteur primordial, ceci s'explique par l'augmentation de la gamme des applications potentielles nécessitant une transmission fiable de données, c'est-à-dire le but dans tout système de communication est d'assurer une transmission de donnée sans erreurs. Pour respecter ce cahier de charge plusieurs techniques ont été développées (amélioration des canaux de transmission, augmentation de la puissance d'émission, amélioration du facteur de bruit. . .) cependant, ces techniques s'avèrent très coûteuses en énergie et technologie, ce qui nous pousse à trouver une solution plus abordable. Le contrôle de l'erreur par le codage est donc indispensable. Pour cela, l'information devra être codée d'une manière spéciale permettant de détecter les erreurs, ou ce qui est encore mieux de les corriger automatiquement. On a été amené à concevoir des codes détecteurs et correcteurs d'erreurs. Parmi ces codes on peut citer les codes en bloc. Les codes convolutifs ou convolutionnel, introduits en 1995 par Elias, peuvent être considérés comme un cas particulier et une alternative au codes en blocs linéaires. Pour décoder et reconstituer l'information original, souvent on a recours à l'algorithme le plus répandu : l'algorithme de Viterbi.

Organisation du mémoire

L'objectif de notre travail consiste à étudier la simulation et réaliser l'implémentation sur Circuit Reconfigurable d'un décodeur de VITERBI à décision ferme. Ce circuit sert à décoder l'information au sein des systèmes de communication numérique afin de corriger les erreurs de transmission.

Le travail présenté dans ce mémoire est organisé en quatre chapitre. Dans le premier chapitre, nous présentons un bref rappel théorique concernant la chaine de transmission numérique, ensuite nous enchainons avec des définitions qui concerne le codage de canal et des codes correcteurs, puis nous parlons des Codeurs Convolutionnels et illustrons leurs fonctionnement par un exemple, afin d'introduire notre travail qui porte sur le décodeur de VITERBI à décision ferme. À la fin nous verrons un exemple d'application des ces codeurs.

Dans le second chapitre nous parlons de l'algorithme de VITERBI et ces différents types (SOVA, SISO) puis nous exposons le schéma fonctionnel du décodeur et ses différents étages fonctionnels qui le constituent (BMU, PMU et DU). Le troisième chapitre décrit en détails notre travail. Nous illustrons l'architecture proposé aussi les modifications apportées au schéma vu dans le deuxième chapitre (MCU et DU).

Dans le quatrième chapitre nous présentons brièvement la carte FPGA utilisée (Virtex2 XC2V3000-5FG676) puis nous exposons les simulations faites sous MATLAB® et ISE de Xilinx.

Chapitre 1

Codeurs Convolutionnels

1.1 Introduction

Les codes convolutifs, introduits en 1955 par Elias, peuvent être considérés comme un cas particulier des codes en bloc linéaires, mais un point de vue plus large nous fera découvrir que la structure convolutive additionnelle munit le code linéaire de propriétés favorables qui facilitent à la fois son codage et améliorent ses performances.

Dans ce premier chapitre nous parlerons principalement des codeurs convolutionnels qui constituent une méthode de codage permettant d'effectuer des transmissions sans les erreurs qui peuvent être causées par le bruit du canal de transmission. Nous commencerons par établir quelques définitions essentielles concernant la chaîne de transmission numérique. Puis nous parlerons du codage de canal de son utilité et des codes correcteurs d'erreur. Et enfin nous aborderons les codes convolutionnels qui sont un type très répandu des codes correcteur d'erreur.

1.2 Chaîne de transmission numérique et théorie de l'information

Un signal quelconque ne peut pas être transmis sans modification à travers un canal bruité. Il doit être traité. Il subit donc plusieurs opérations. Il est compressé, codé, modulé avant la transmission mais aussi démodulé, décodé à la réception. Il traverse donc plusieurs étapes constituant la chaîne de transmission numérique.

1.2.1 Chaîne de transmission numérique

C'est l'ensemble d'opérations à appliquer à un signal afin de le transmettre numériquement à travers un canal de transmission.

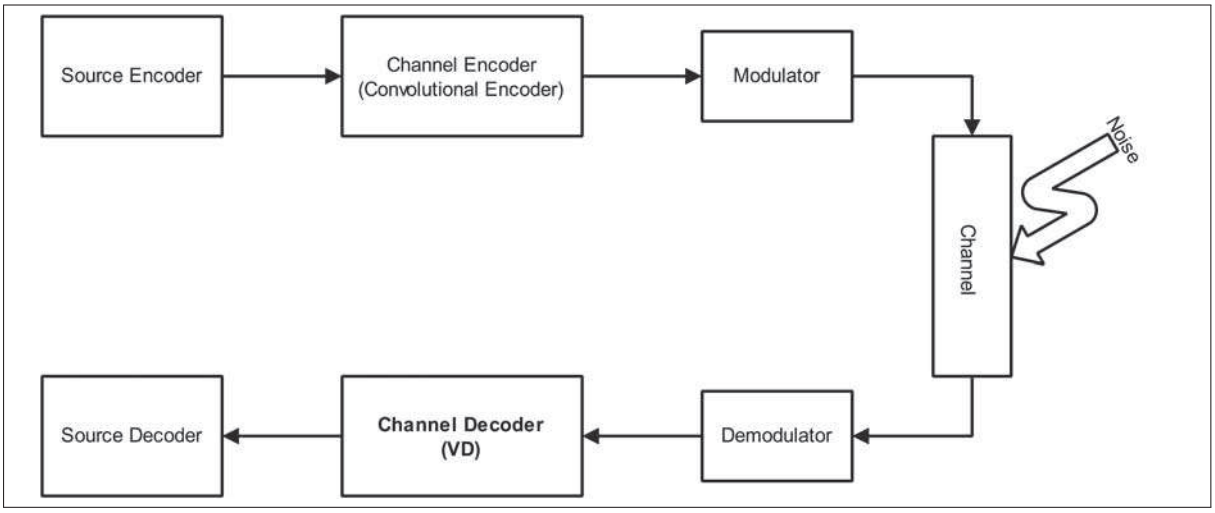

FIGURE 1.1: Chaîne de transmission numérique.

1.2.1.1 Canal

Le canal est le support de transmission de l'information. Il peut être filaire, optique, hertzien...

Le moyen de transmission introduit un nombre d'effets indésirables comme l'atténuation, la distorsion, l'interférence et le bruit. Ces effets indésirables engendrent une incertitude quant à la réception correcte et intègre de l'information envoyée.

La manière dont les symboles transmis sont corrompus est décrite dans les définitions des différents types de canaux :

Canal sans mémoire

Dans un canal sans mémoire, la probabilité d'erreur est indépendante du symbole transmis.

Canal symétrique

Dans un canal symétrique la probabilité qu'un symbole i soit transmis sachant que le symbole j a été reçu est égale à la probabilité de recevoir un i sachant qu'un j a été envoyé. Et ce quelles que soient les valeurs de i et de j.

1.2.1.2 Codeur de source

On donne habituellement à l'information une représentation discrète, probablement en conjonction avec les techniques de suppression de redondance au sein de la donnée. La quantité d'information contenue dans n'importe quel message est définie en terme de

probabilité p que le message soit transmis. L'information H, mesurée en bits, est donnée par[9] :

$$H = p \log_2 \left(\frac{1}{p} \right) \tag{1.1}$$

Par exemple un message avec 1% de chance d'être transmis contiendrait approximativement 6.64 bits d'information. (A ne pas confondre avec le bit informatique qui signifie *binary digit*). S'il y a M messages probables et que la sélection d'un message m a une probabilité de p_m, la quantité moyenne d'information transférée dans les messages est[9] :

$$\overline{H} = \sum_{m=0}^{M-1} p_m \log_2 \left(\frac{1}{p_m} \right) \tag{1.2}$$

et ce dans la condition :

$$\sum_{m=0}^{M-1} p_m = 1 \tag{1.3}$$

Si les messages sont équiprobables, c'est-à-dire $p_m = \frac{1}{M}$, alors l'information moyenne transférée est $\overline{H} = \log_2 (M)$. Par exemple avec 256 messages un code de 8 bits peut être utilisé pour représenter n'importer quel message, et s'ils sont équiprobables, l'information contenue dans n'importe quel message est aussi de 8 bits.

1.2.1.3 Codeur de canal

Globalement, le codage de canal sert à introduire de la redondance dans la séquence transmise afin de détecter et corriger les erreurs dus à la transmission, lors du décodage. Le nombre de bits transmis est donc supérieur au nombre de bits nécessaires pour représenter l'information (Ajout de redondance). Sans cela, le code ne nous permettrai même pas de détecter la présence d'erreur. Par voie de conséquence ne nous donnerais aucun contrôle sur les erreurs introduites. Ce qui veut dire que, théoriquement, une compression incomplète effectuée par le codeur de source comprendrait des propriétés de gestion d'erreur. Néanmoins, en pratique il serait préférable de compresser l'information de la source autant que possible puis de réintroduire de la redondance d'une manière à pouvoir l'exploiter correctement à la réception : c'est le principe de fonctionnement du codeur de canal.

1.2.1.4 Modulation

Le modulateur peut être considéré comme un convertisseur numérique analogique, préparant le flux de bits au canal de transmission qui est analogique. Initialement, le flux de bit est dans une représentation en bande de base, c'est-à-dire que ça fréquence est basse et que le changement de signaux se fait à une fréquence comparable à celle

des symboles numériques représentés. Une représentation appropriée est le format NRZ (Non Return to Zero), qui représente les bits par des niveaux $+V$ ou $-V$.

Même s'il est possible de transmettre ce signal, il est généralement translaté à une fréquence plus élevée. La raison à cela est la possibilité d'utiliser différentes régions du spectre pour différentes transmissions, ainsi que le fait que les hautes fréquences ont des longueurs d'ondes plus petites et nécessitent donc de plus petites antennes.

Un schéma de modulation très utilisé est le BPSK (Binary Phase Shift Keying). Dans ce schéma, on multiplie le signal NRZ en bande de base par une porteuse dont la fréquence est un multiple du débit de bits. Il en résulte que le signal transmis durant l'intervalle de temps correspondant à un bit est soit la porteuse elle même, soit son inverse.

Il est aussi possible d'utiliser une seconde porteuse déphasé de $\frac{\pi}{2}$ de la première, la moduler et l'ajouter à la première. En d'autres termes, si le signal BPSK est $\pm\cos(2\pi f_c t)$ (f_c étant la fréquence de la porteuse et t le temps) le second signal sera $\pm\sin(2\pi f_c t)$ et la résultante sera alors :

$$s(t) = \sqrt{2}\cos\left(2\pi f_c t + i\frac{\pi}{4}\right); \quad i = -3,\ -1,\ +1,\ +3 \tag{1.4}$$

Ceci est connu sous le nom de QPSK (QuadriPhase Shift Keying), est possède l'avantage _par rapport à la modulation BPSK_ de permettre la transmission de deux fois plus de bits durant le même temps et en utilisant la même bande passante[11].

Un diagramme de phase est décrit dans la Figure 1.2. La distribution des valeurs des bits sur les phases implique que chaque porteuse est indépendamment modulée en alternant les bits du flux de données codées.

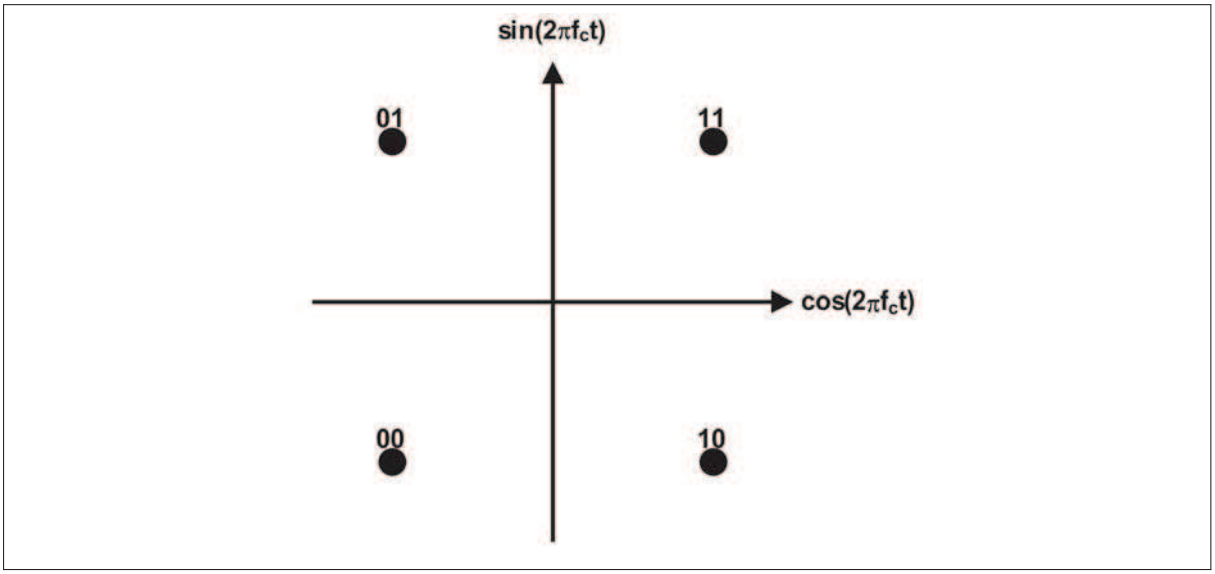

FIGURE 1.2: Diagramme de la modulation QPSK utilisant le code GRAY.

D'autres méthodes de modulations comprennent la FSK, dans laquelle la donnée module la fréquence du signal transmis ; l'avantage de la FSK est la simplicité d'implémentation. Néanmoins, sa résistance au bruit est inférieure à celles des modulations BPSK et QPSK. Dans la MPSK un nombre plus important de phases peut être transmis. En ce qui concerne la modulation QAM, une paire de porteuses en quadrature de phase sont utilisées ; on leur attribue deux amplitudes différentes avant de les additionner pour produire un signal contenant différentes amplitudes et différentes phases. La méthode QAM résiste plus au bruit que son équivalent MPSK, mais la différence d'amplitude peut poser problème dans les systèmes comportant des éléments non linéaires[11].

1.2.1.5 Bruits

Un signal est toujours affecté de petites fluctuations plus ou moins importantes. Ces fluctuations, dont les origines peuvent être diverses sont appelées bruit électrique, bruit de fond, ou tout simplement bruit. On peut citer :

– Le bruit de « friture » dans un récepteur radio entre deux stations ;
– Le bruit de souffle à la sortie d'un amplificateur.

Un signal n'est jamais sans bruit ; simplement il peut être affecté d'un bruit plus ou moins important. Si celui-ci est très faible par rapport au signal, il devient invisible à l'oscilloscope.

Cause externe à la chaine

C'est le bruit qui affecte déjà le signal à l'entrée de la chaîne et qui est amplifié et filtré avec le signal.

• L'antenne du récepteur capte le signal de l'émetteur mais aussi des parasites industriels et le bruit de fond cosmique de tous les astres qui rayonnent des ondes électromagnétiques ;

• La tête de lecture du tourne-disque qui capte la musique inscrite dans le sillon mais aussi les bruits de surface, les vibrations de la platine et les décharges électrostatiques.

Pour ce type de bruit, une conception soignée du produit (mécanique, électronique, blindages, filtrages, etc . . .) nous donne un bruit minimal en sortie.

Cause interne à la chaine

L'agitation thermique des électrons provoque des fluctuations aléatoires de la tension en tout point d'un circuit. C'est le bruit thermique qui existe toujours. On peut le diminuer en choisissant des composants à faible bruit mais on n'arrivera jamais à le supprimer.

Outre ce bruit thermique, il existe d'autres catégories de bruits, toujours associés à la nature discontinue des porteurs de charges.

Les fluctuations étant aléatoires, il est clair qu'en moyenne, $b(t)$ est aussi souvent positive que négative. Nous en déduisons un premier résultat important :

La valeur moyenne d'une tension de bruit est nulle $b(t) = 0$, par contre, si nous élevons cette tension au carré avant de prendre la valeur moyenne, le résultat ne sera plus nul. Nous en déduisons que :

La valeur efficace d'une tension de bruit n'est pas nulle $B_{eff} = \sqrt{\overline{b(t)^2}} \neq 0$. Une tension de bruit sera donc caractérisée par sa valeur efficace.

1.2.2 Théorie de l'information

Shannon démontre en 1948 que l'on peut réduire autant que l'on veut les erreurs et donc transmettre de l'information vierge d'erreur du moment que l'on réduit le débit de transmission. Il y a donc un compromis à faire entre le débit de transmission des bits et le nombre d'erreurs. Le débit maximal pour lequel la transmission se fait sans erreur est appelé Capacité du canal de transmission. Il est possible de réaliser une transmission sans erreur à travers un canal bruité, tant que le débit de transmission n'excède pas la capacité du canal. Pour expliquer ce résultat nous devons mesurer l'information.

1.2.2.1 Entropie

Dans le reste de cette partie nous adopterons les notations ci-dessous :

Considérons une source discrète et sans mémoire. A un instant donné, la source discrète d'information émet le symbole aléatoire discret $X = x_i$ qui peut prendre une des M valeurs possibles $x_1, x_2, \ldots, x_M$. Les probabilités d'apparition de ces symboles sont : $P_X(x_1), P_X(x_1), \ldots, P_X(x_M)$ avec $P_X(x_i) = \Pr\{X = x_i\}$.

L'information moyenne associée à la variable aléatoire X ou entropie comme définie précédemment, est[11] :

$$I(X) = -\sum_{i=1}^{M} P_X(x_i) \log_2 P_X(x_i) \tag{1.5}$$

1.2.2.2 Capacité de canal

A l'aide du concept d'entropie, nous pouvons modéliser le canal selon le diagramme de Berger[2] ; Figure 1.3 :

Ici, X correspond au symbole d'entrée et R correspond au symbole reçu. Nous considérons aussi que M valeurs d'entrée sont possibles $x_1, x_2, \ldots, x_M$ et N valeurs de sorties $r_1, r_2, \ldots, r_N$. A l'aide des lois de probabilités conditionnelles nous avons :

$$\Pr_{X|R}(x_i|r_j) = \Pr\{X = x_i|R = r_j\} \tag{1.6}$$

et

$$\Pr_{R|X}(r_j|x_i) = \Pr\{R = r_j|X = x_i\} \tag{1.7}$$

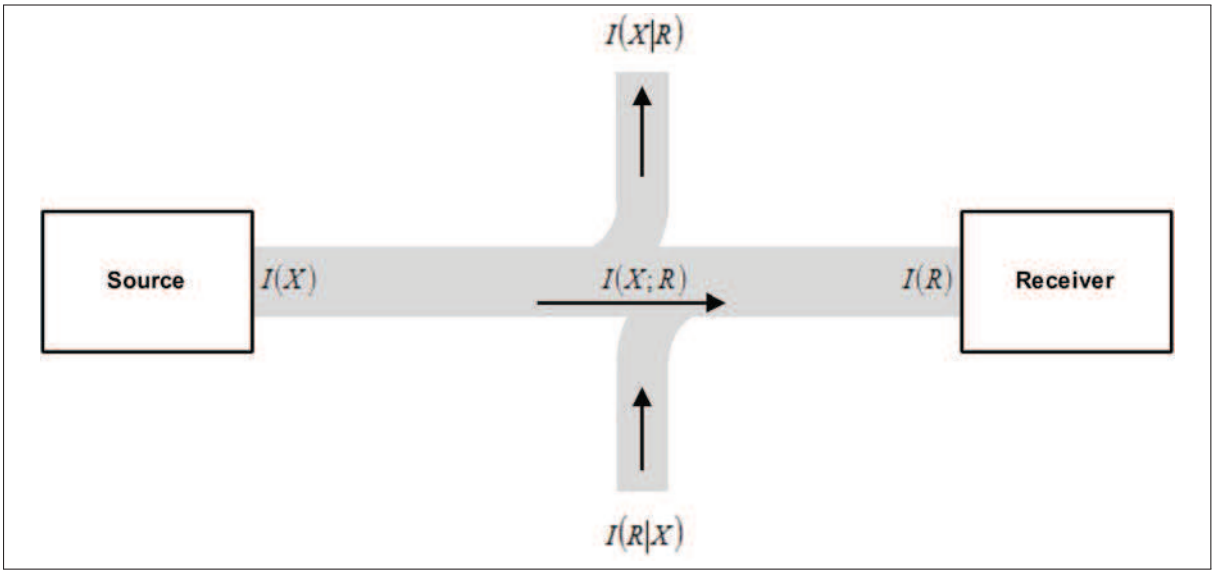

FIGURE 1.3: Diagramme de Berger.

Les entropies conditionnelles sont données par :

$$I\left(X|R\right) \tag{1.8}$$

et

$$I\left(R|X\right) \tag{1.9}$$

Avec ces probabilités conditionnelles, l'information mutuelle :

$$I\left(X;\ R\right) \tag{1.10}$$

peut être déduite. Elle mesure la quantité d'information transmise à travers le canal de l'entrée à la sortie pour une source d'information donnée.

La capacité C de ce canal est obtenue en maximisant l'information mutuelle $I\left(X;\ R\right)$ tout en respectant les caractéristiques statistiques de la variable d'entrée X, c'est-à-dire en choisissant de manière appropriée les probabilités $\{\Pr_X\left(x_i\right)\}_{1\leq i\leq M}$. Cela mène à :

$$C = \max_{\{\Pr_X(x_i)\}_{1\leq i\leq M}} I\left(X|R\right) \tag{1.11}$$

Si l'entropie d'entrée $I\left(X\right)$ est inférieure à la capacité du canal $I\left(X\right) < C$, alors l'information peut être transmise à travers un canal bruité avec une probabilité d'erreur aussi petite que l'on veut. De cette manière la capacité C quantifie réellement la capacité de transmission d'information du canal.

1.2.2.3 Canal Symétrique Binaire

Un exemple très important de canal sans mémoire est le Canal Symétrique Binaire ou BSC. Le canal de la Figure 1.4 montre le diagramme d'un canal symétrique binaire avec une probabilité d'erreur p.

En calculant l'information $I(X;\ R)$, la capacité du canal symétrique binaire est :

$$C = 1 + p\log_2(p) + (1-p)\log_2(1-p) \tag{1.12}$$

La capacité du canal est égale :

$$C = \begin{cases} 1 & p = 0 \text{ ou } p = 1 \\ 0 & p = 1/2 \end{cases} \tag{1.13}$$

Contrastant avec le canal symétrique binaire qui possède des entrées et sorties discrètes, le canal AWGN est définit par des variables aléatoires réelles continues.

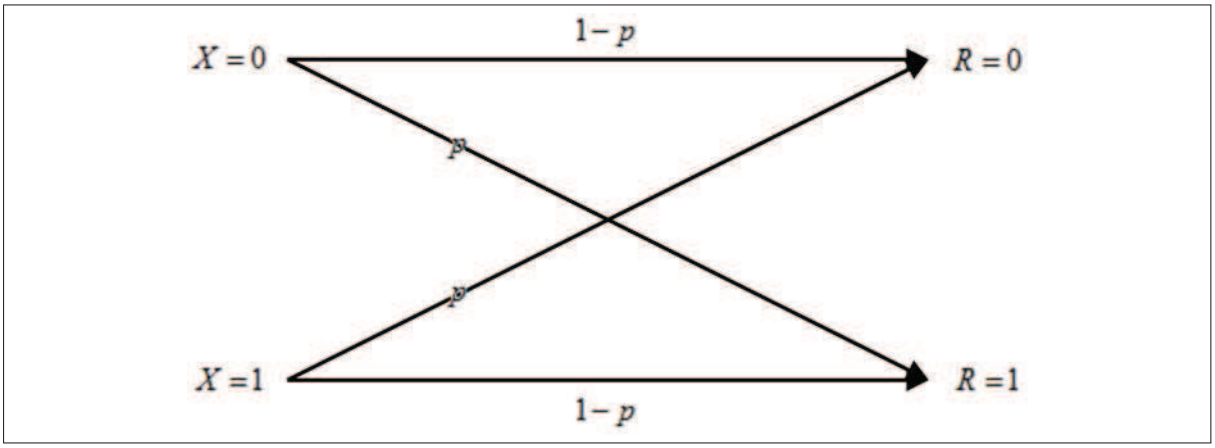

FIGURE 1.4: Canal Binaire Symétrique.

1.2.2.4 Canal AWGN

Jusqu'ici nous n'avons considéré que des symboles à valeurs discrètes. Le concept d'entropie peut être transféré aux variables aléatoires continues réelles, en incluant la dite entropie différentielle. Il en résulte que le canal à entrée et sorties réelles peut aussi bien être caractérisé par l'information mutuelle $I(X;\ R)$ et son maximum par la capacité C.

- Puissance du signal : $S = \mathrm{E}\{X^2\}$;
- Puissance du bruit : $N = \mathrm{E}\{Z^2\}$.

La capacité du canal AWGN est donné par :

$$C = \frac{1}{2}\log_2\left(1 + \frac{S}{N}\right) \tag{1.14}$$

La capacité du canal dépend exclusivement du SNR ; $SNR = \frac{S}{N}$.

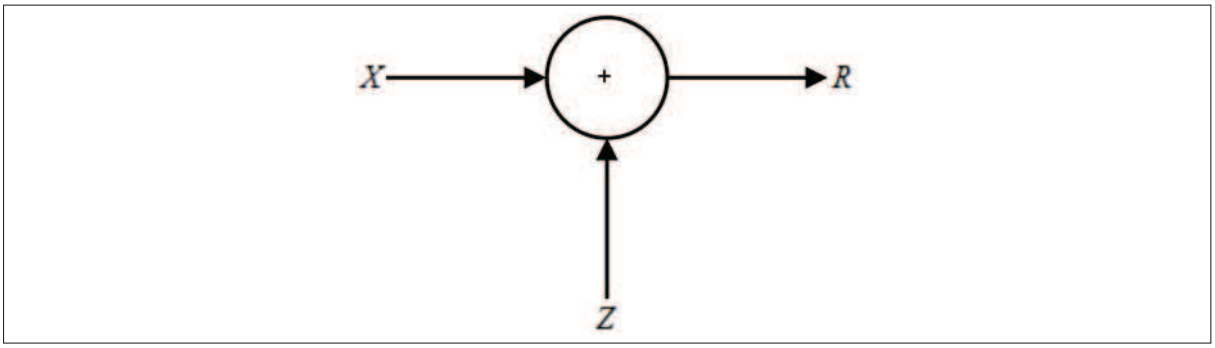

FIGURE 1.5: Canal AWGN.

Pour comparer les capacités du canal symétrique binaire et du canal AWGN, on suppose que le signal est modulé (BPSK), et que la réception est optimisée à l'aide d'un filtre adapté. Le rapport signal sur bruit (SNR) de la sortie réelle R du filtre adapté est donnée par :

$$\frac{S}{N} = \frac{E_b}{N_0/2} \tag{1.15}$$

Avec E_bl'énergie par bit, et N_0 la densité spectrale de puissance du bruit. Si la sortie R du filtre adapté est comparée avec 0, on obtient le canal symétrique binaire avec une probabilité d'erreur :

$$p = \frac{1}{2}\text{erfc}\left(\sqrt{\frac{E_b}{N_0}}\right) \tag{1.16}$$

Ici, erfc est la fonction d'erreur complémentaire.

1.3 Codage de canal et codes correcteurs d'erreur

Même si la théorie de l'information nous dit qu'il est possible de trouver un code de canal qui nous donne une probabilité d'erreur aussi petite que l'on veut pour un canal donné, il n'en demeure pas moins que d'un point de vue pratique cela ne soit pas si aisé.

Il est généralement difficile de concevoir un bon code de canal. Dans cette section nous parlerons des paramètres utilisés pour décrire un code de canal puis nous aborderons le sujet essentiel nous concernant : les codes convolutionnels (ou convolutifs). Dans les chapitres ultérieurs, nous aborderons l'algorithme de Viterbi pour le décodage de ces dits codes, ainsi que ça simulation et son implémentation.

1.3.1 Principes fondamentaux des codes en blocs

1.3.1.1 Paramètres du code

Les codes de canal sont caractérisés par plusieurs paramètres. Les plus importants sont le taux de codage et la distance de HAMMING minimale. A l'aide de ces paramètres l'efficacité du processus d'encodage peut être évaluée pour un code donné.

Dans ce qui suit nous considèrerons les variables suivantes :

- Mot information à coder $u = (u_0,\, u_1, \ldots,\, u_{k-1})$ de longueur k ;
- Mot code émis $b = (b_0,\, b_1, \ldots,\, b_{n-1})$ de longueur n ;
- Mot reçu à décoder $r = (r_1,\, r_2, \ldots,\, r_{n-1})$ de longueur n ;
- Mot décodé $\widehat{u} = (\widehat{u_0},\, \widehat{u_1}, \ldots,\, \widehat{u_{k-1}})$ de longueur k.

FIGURE 1.6: État de l'information dans la chaîne de transmission.

Le mot décodé $\widehat{u}$ peut être identique à u ou différent, selon que les erreurs soient corrigées ou pas.

Rendement du codage

En partant du principe que tous les symboles u_i de l'information peuvent prendre q valeurs, le nombre de mots informations possibles est donné par :

$$M = q^k \tag{1.17}$$

Sachant que le mot code de longueur n est plus long que le mot d'information de longueur k, le taux auquel l'information transmise à travers le canal est réduite est appelé le taux de codage du code :

$$R = \frac{\log_q (M)}{n} = {}^{k}/_{n} \tag{1.18}$$

Par exemple, si le code consiste uniquement à répéter chaque symbole trois fois, on aura alors $k = 1$, et $n = 3$; ce qui donne un taux de codage de ${}^{k}/_{n} = {}^{1}/_{3} \simeq 0.3333$.

Poids et distance de Hamming

A chaque mot code $b = (b_0,\, b_1, \ldots,\, b_{n-1})$, peut être associée un poids $wt\,(b)$, défini comme le nombre de composantes non nulles $b_i \neq 0$: C'est le poids de HAMMING[1].

$$wt\,(b) = |\{i :\ b_i \neq 0;\ 0 \leq i < n\}| \tag{1.19}$$

La distance entre deux mots b et $\widehat{b}$ est donnée par la Distance de HAMMING[1] :

$$\text{dist}\left(b,\, \widehat{b}\right) = \left|\left\{i :\, b_i \neq \widehat{b}_i;\, 0 \leq i < n\right\}\right| \tag{1.20}$$

La distance de HAMMING traduit le nombre de composantes différentes entre b et $\widehat{b}$, et mesure ainsi la proximité entre b et $\widehat{b}$.

Pour un code consistant en M mots codes $b_0,\, b_1, \ldots,\, b_M$, la distance de HAMMING minimale est donnée par :

$$d = \text{dist}_{\min} = \min_{\forall b \neq \widehat{b}} \left\{\text{dist}\left(b,\, \widehat{b}\right)\right\} \tag{1.21}$$

1.3.1.2 Décodage au maximum de vraisemblance

Il existe plusieurs stratégies de décodage :

FIGURE 1.7: Stratégie de décodage.

Pour le décodage au minimum de probabilité d'erreur ou *Maximum A Posteriori* (MAP) la règle de décodage est la suivante :

$$\widehat{b}\,(r) = \max\left\{\Pr\left\{b|r\right\}\right\} \tag{1.22}$$

Pour le décodage au maximum de vraisemblance (ML) la règle est la suivante :

$$\widehat{b}\,(r) = \max\left\{\Pr\left\{r|b\right\}\right\} \tag{1.23}$$

Le décodage ML et identique au décodage MAP si tous les mots code sont équiprobables, c'est-à-dire $\Pr\left\{b\right\} = \frac{1}{M}$.

1.3.1.3 Détection et correction d'erreur

En se basant sur la règle de décodage au minimum de distance, nous pouvons évaluer la capacité du code à détecter et corriger les erreurs. Si on considère deux mots codes b et $\widehat{b}$ distant de $d = \text{dist}\left(b,\, \widehat{b}\right)$, où d est la distance minimale de HAMMING. Nous pouvons détecter l'erreur tant que le mot erroné reçu r n'est pas égal à un autre mot code existant. Cette capacité de détection est garantie tant que le nombre d'erreurs est inférieur à la distance minimale de HAMMING. Et ce, pour la simple raison qu'il est nécessaire et non suffisant de changer d composante de b pour arriver à $\widehat{b}$. En conséquence le nombre d'erreurs détectables est :

$$e_{det} = d - 1 \tag{1.24}$$

Pour l'analyse de la capacité de correction d'erreur, nous définissons pour chaque mot b la sphère de correction correspondante de rayon α qui n'est autre que l'ensemble des mots les plus proches de b que de tout autre mot $\widehat{b}$ du codeur. Tous les mots dans cette sphère seront décodés comme un b, c'est-à-dire tous les mots distants de 1, 2, ..., α de b. Ceci est vrai uniquement lorsque les sphères ne possèdent aucun point d'intersection les unes avec les autres, c'est-à-dire si $\alpha < \frac{d}{2}$.

D'où la capacité de correction d'erreur est :

$$e_{cor} = \left[\frac{d-1}{2}\right] \tag{1.25}$$

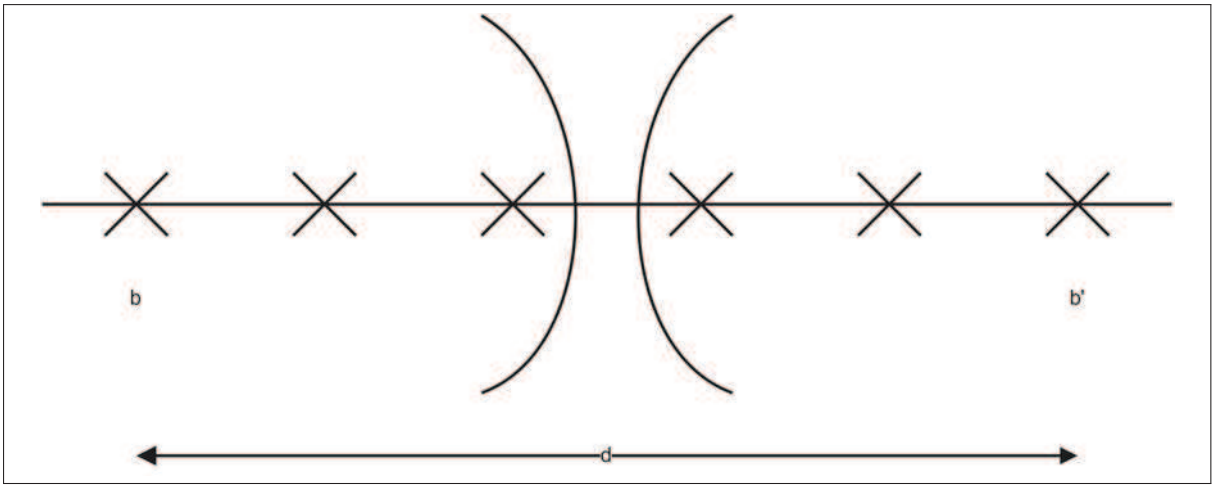

FIGURE 1.8: Détection et correction d'erreurs.

1.4 Codeurs convolutionnels

Aujourd'hui, les codes convolutionnels sont utilisés par exemple dans l'UMTS, et dans les systèmes cellulaires numériques utilisant le GSM. Ils sont aussi utilisés dans les communications par satellite, dans les réseaux sans fil locaux (LAN) et nombre d'autres applications. La principale raison de cette popularité est l'existence d'algorithmes de décodage efficaces. En particulier le très répandu l'algorithme de VITERBI qui est un décodeur utilisant ; en générale ; le décodage ML. Il peut être implémenté avec un niveau de complexité matérielle raisonnable.

1.4.1 Fonctionnement

La manière la plus simple d'introduire les codes convolutionnels est de prendre un exemple simple (voir Figure 1.9).

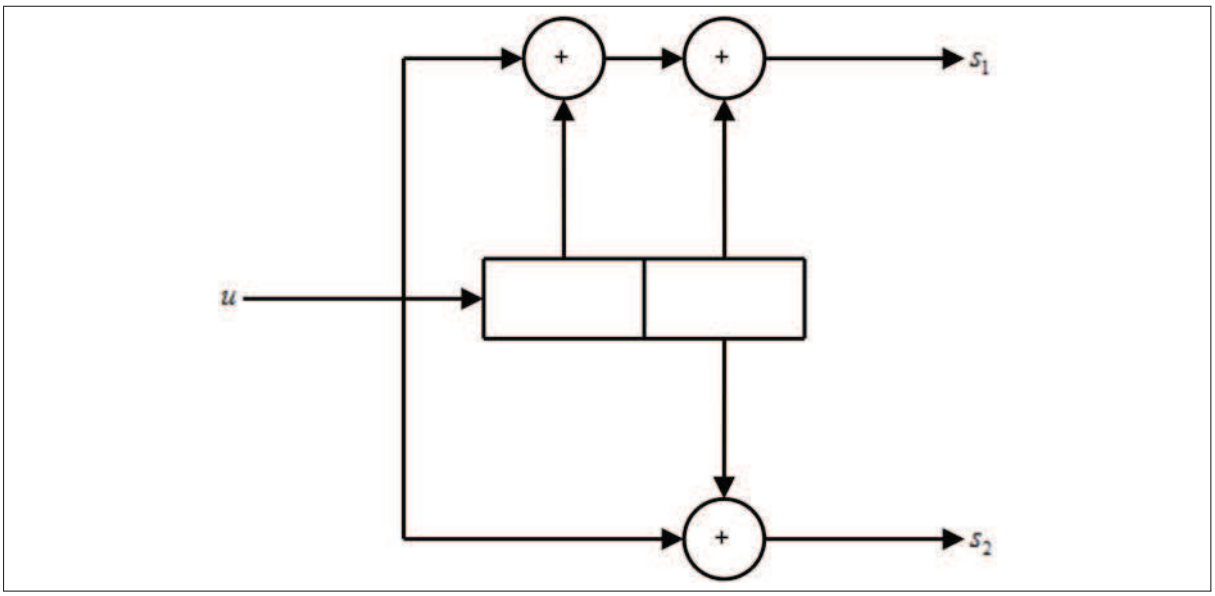

FIGURE 1.9: Codeur convolutionnel $m = 2$ et $R = 1/2$.

L'information entre dans un registre à décalage de longueur $m = 2$ (ce qui correspond à une longueur de contrainte de $L = 3$ et un nombre d'états $N = 2^2 = 4$). La séquence de sortie résulte du multiplexage de deux séquences dénotées s_1 et s_2. Chaque bit de sortie est généré par l'addition modulo 2 de l'entrée actuelle et de certains bits contenus dans le registre. Par exemple, la séquence d'information $u = (1, 1, 0, 1, 0, 0, \ldots)$ sera codée par $s_1 = (1,\ 0,\ 0,\ 0,\ 1,\ 1, \ldots)$ et $s_2 = (1,\ 1,\ 1,\ 0,\ 0,\ 1, \ldots)$ et la séquence de sortie après multiplexage sera $b = (1,\ 1,\ 0,\ 1,\ 0,\ 1,\ 0,\ 0,\ 1,\ 0,\ 1,\ 1, \ldots)$.

Un codeur convolutionnel est un circuit linéaire séquentiel, c'est-à-dire un système invariant dans le temps. Il est donc entièrement caractérisé par sa réponse impulsionnelle. Dans le cas du codeur de la Figure 1.9 les réponses impulsionnelles sont : $g_1 = (1,\ 1,\ 1,\ 0, \ldots)$ et $g_2 = (1,\ 0,\ 1,\ 0, \ldots)$ respectivement. Ces vecteurs générateurs de réponses impulsionnelles sont utiles pour calculer les séquences de sortie pour une séquence d'entrée arbitraire : c'est les polynômes générateurs écrits souvent dans la base 8.

$$s_1 = u \star g_1 \tag{1.26}$$

$$s_2 = u \star g_2 \tag{1.27}$$

Les équations de génération de s_1 et s_2 peuvent être considérées comme la convolution des séquences d'entrée et des générateurs de réponse impulsionnelles $g_1$et g_2. Ceci justifie le nom de codes convolutionnels.

$s_1$et s_2 sont convolutions de la même séquence d'entrée. Cela signifie qu'un seul bit d'information est codé deux fois, le taux d'information utile dans la séquence de sortie est de moitié. En conséquence on dit que le rendement R du codeur est de $R = k/n = 1/2$, où k est le nombre d'entrées du codeur et n le nombre de sorties.

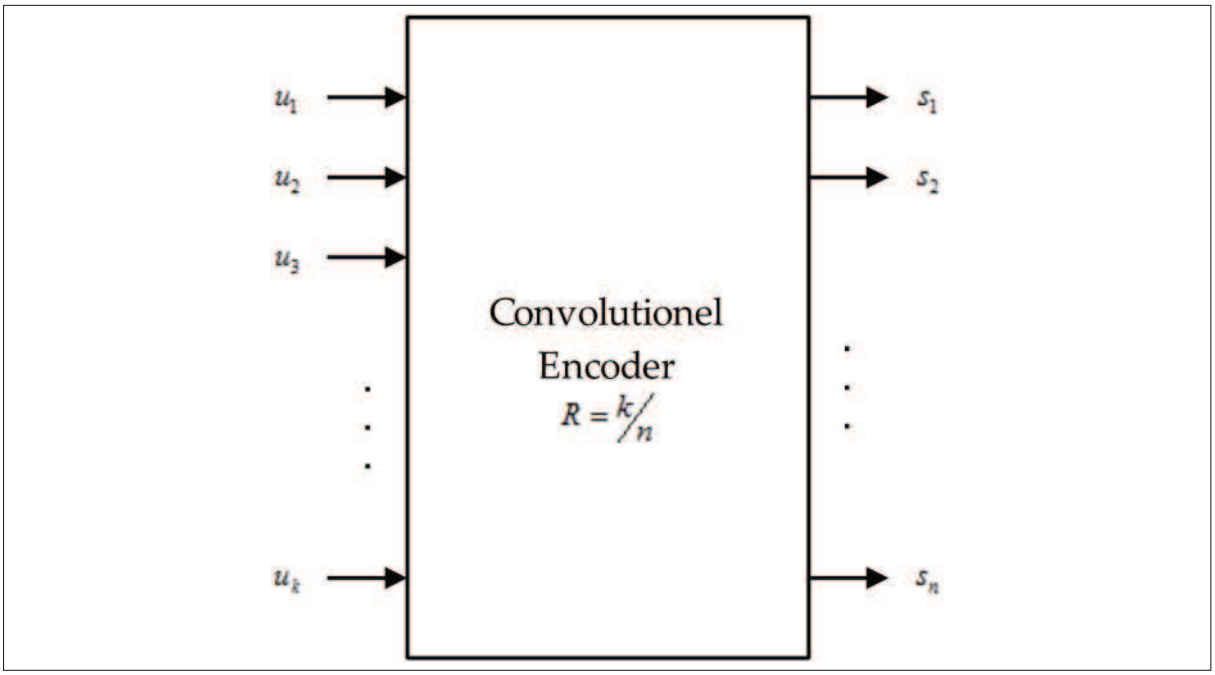

FIGURE 1.10: Codeur convolutionnel de rendement $R = k/n$.

Avec les codeurs convolutionnels, il est possible de construire différents circuits résultant en une même séquence de sortie. Le codeur de l'exemple précédent n'a pas de boucle de retour cependant en règle générale un codeur peut avoir des boucles de retour donc une réponse impulsionnelle infinie (voir Figure 1.3.3[2]).

1.4.2 Caractéristiques du codeur convolutionnel utilisé

Pour les besoins de notre sujet nous avons choisi un codeur convolutionnel de rendement $R = 1/2$ une mémoire de taille $m = 3$ (donc une longueur de contrainte $L = 4$ et un nombre d'états $n = 2^3 = 8$). Les polynômes générateurs de notre codeur convolutionnelles sont $g_1 = (1, 1, 0, 1)$ et $g_2 = (1, 0, 1, 1)$, ou encore en écriture octale : $G = (15_8, 13_8)$.

Le schéma du codeur est celui de la Figure 1.12.

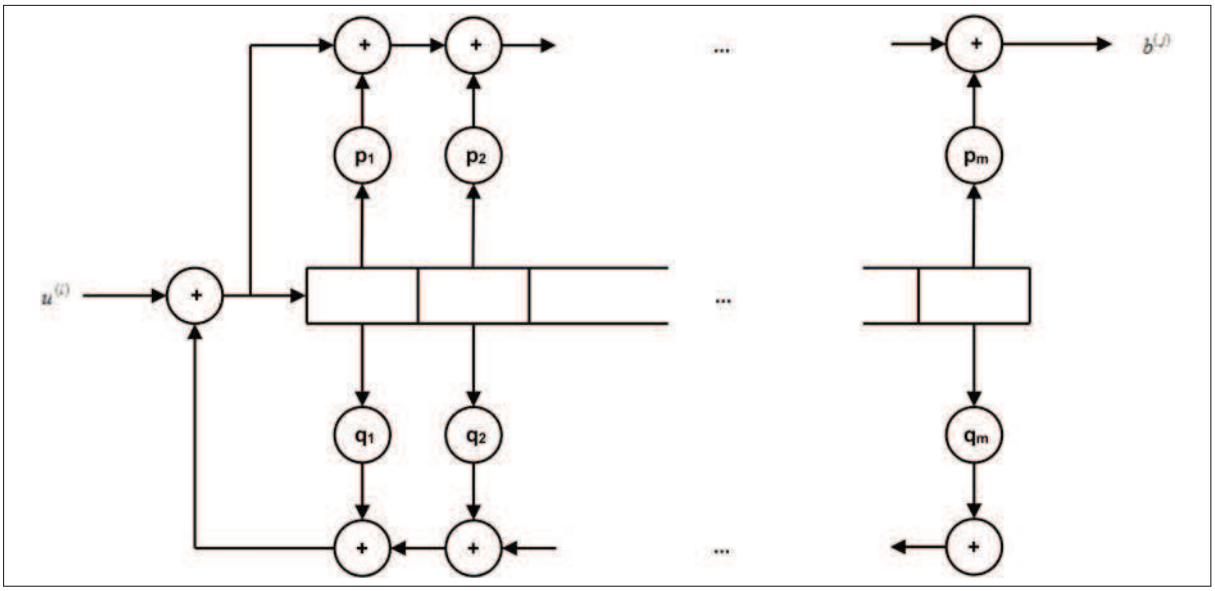

FIGURE 1.11: Registre à décalage dans la forme canonique du codeur.

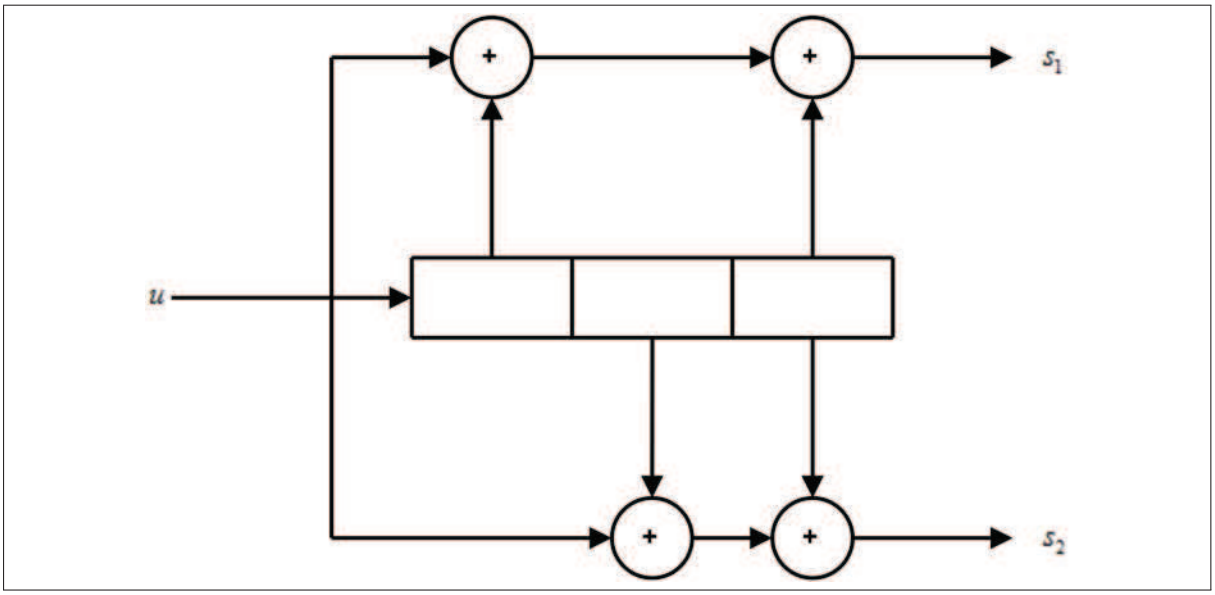

FIGURE 1.12: Schéma du codeur convolutionnel choisi.

1.4.3 Diagramme d'état et ses représentations

Le diagramme d'état du codeur convolutionnel que nous avons choisi est exposé dans le paragraphe précédent et décrit dans la Figure 1.13.

- Chaque case représente un état du codeur, c'est-à-dire le contenu binaire des mémoires du registre à décalage ;
- Chaque branche représente une transition entre deux états et est étiquetée de l'entrée et des deux sorties correspondantes.

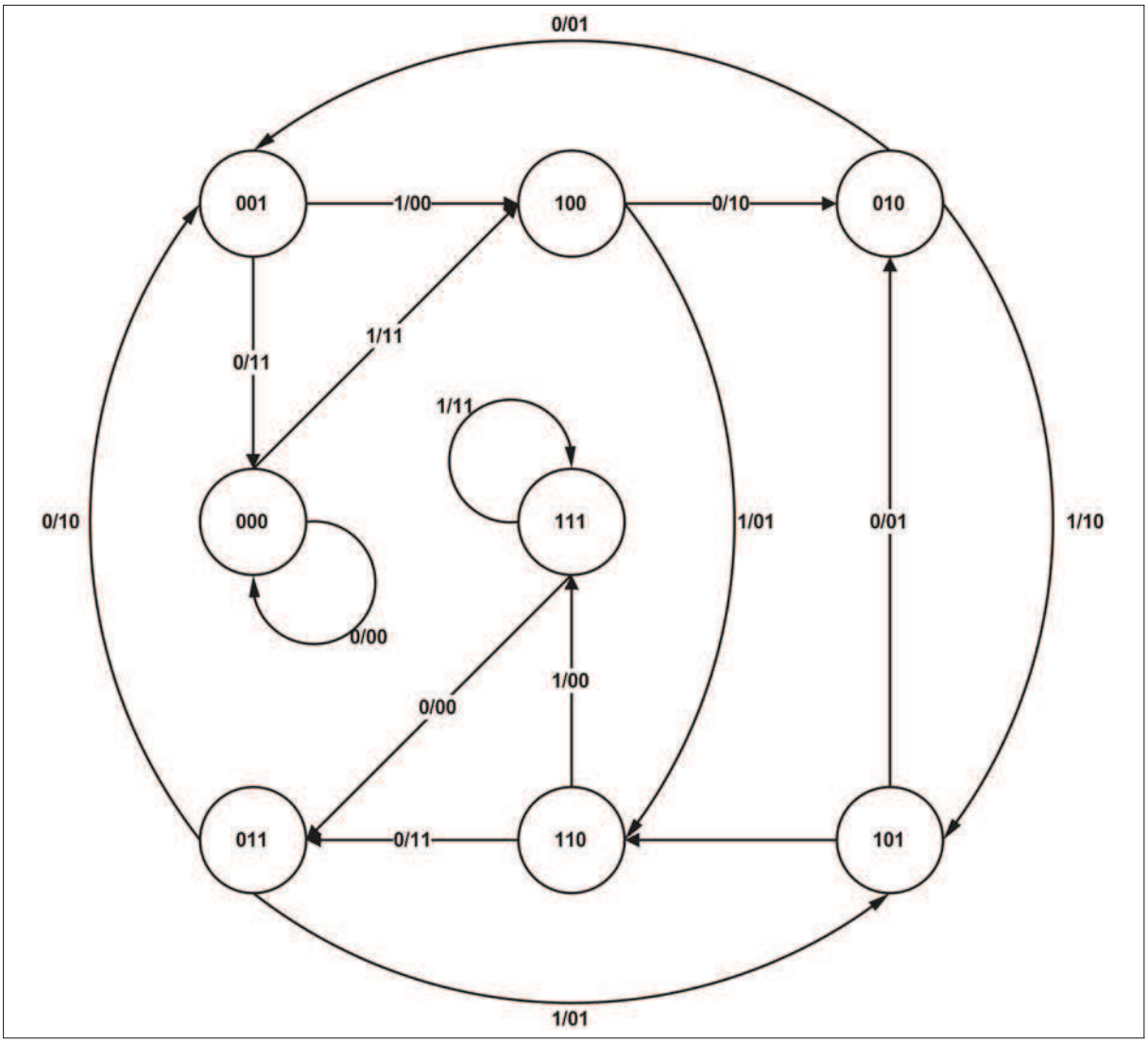

FIGURE 1.13: Diagramme d'état du codeur convolutionnel.

Une autre représentation possible des états du codeur convolutionnel est le diagramme en arbre (Voir Figure 1.14).

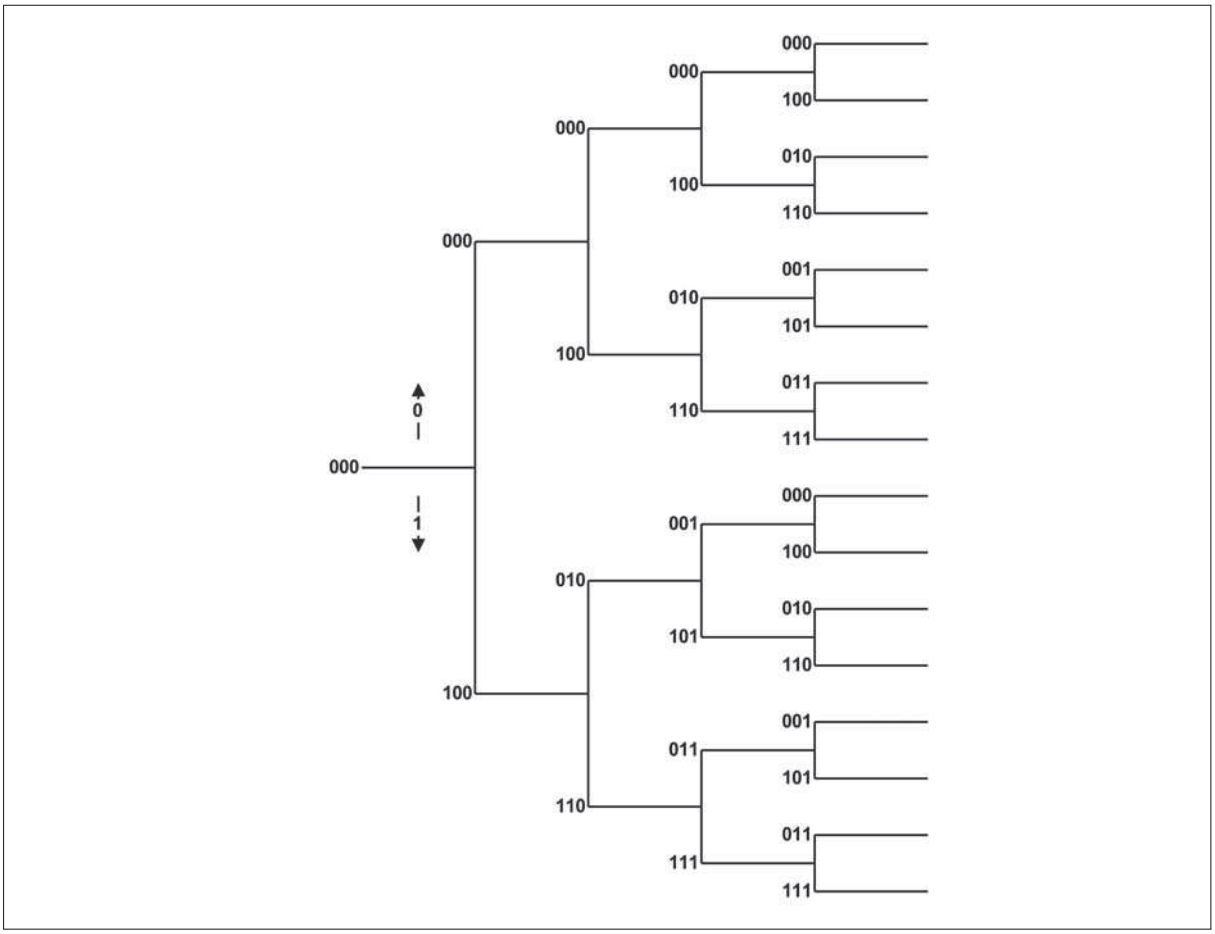

FIGURE 1.14: Représentation en arbre du codeur convolutionnel.

Toujours, en peut aussi représenter le codeur par un tableau qui contient le bit d'entrée, l'état courant du codeur, l'état suivant et le mot code de sortie (Voir Tableau 1.1).

Bit d'entrée	État courant			État suivant			s_1	s_2
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	1	1
0	0	0	1	0	0	0	1	1
1	0	0	1	1	0	0	0	0
0	0	1	0	0	0	1	0	1
1	0	1	0	1	0	1	1	0
0	0	1	1	0	0	1	1	0
1	0	1	1	1	0	1	0	1
0	1	0	0	0	1	0	1	0
1	1	0	0	1	1	0	0	1
0	1	0	1	0	1	0	0	1
1	1	0	1	1	1	0	1	0
0	1	1	0	0	1	1	1	1
1	1	1	0	1	1	1	0	0
0	1	1	1	0	1	1	0	0
1	1	1	1	1	1	1	1	1

TABLE 1.1: Autre représentation du codeur convolutionnel.

1.5 Applications des codes convolutionnels

Dans les communications mobiles, il est nécessaire d'utiliser un codeur de canal afin d'éviter les pertes dues aux erreurs de transmission. Les standards de la téléphonie mobile comme GSM et UMTS distinguent la voix et la communication de données. La transmission de la voix codée est un service à temps réel qui ne permet qu'une petite latence, mais qui tolère les erreurs résiduelles dans le le flux de données. Les erreurs résiduelles pouvant survenir sont masquées efficacement par le décodeur vocal. La transmission de données, quant à elle, requière des taux d'erreurs résiduels très faibles. Les standards GSM et UMTS définissent différents codages de vocaux.

Le codage de canal pour la GSM utilise un codeur convolutionnel ayant une mémoire $m = 4$ (Voir Figure 1.15). Pour terminer le code, quatre zéros (*tail bits*) sont ajoutés. Alors que le standard UMTS utilise un puissant codeur de mémoire $m = 8$ et un taux $R = 1/3$[2].

Le protocole ARQ est une autre application des codes convolutionnels pour la téléphonie mobile. Le principe de l'ARQ est de détecter et de retransmettre les données

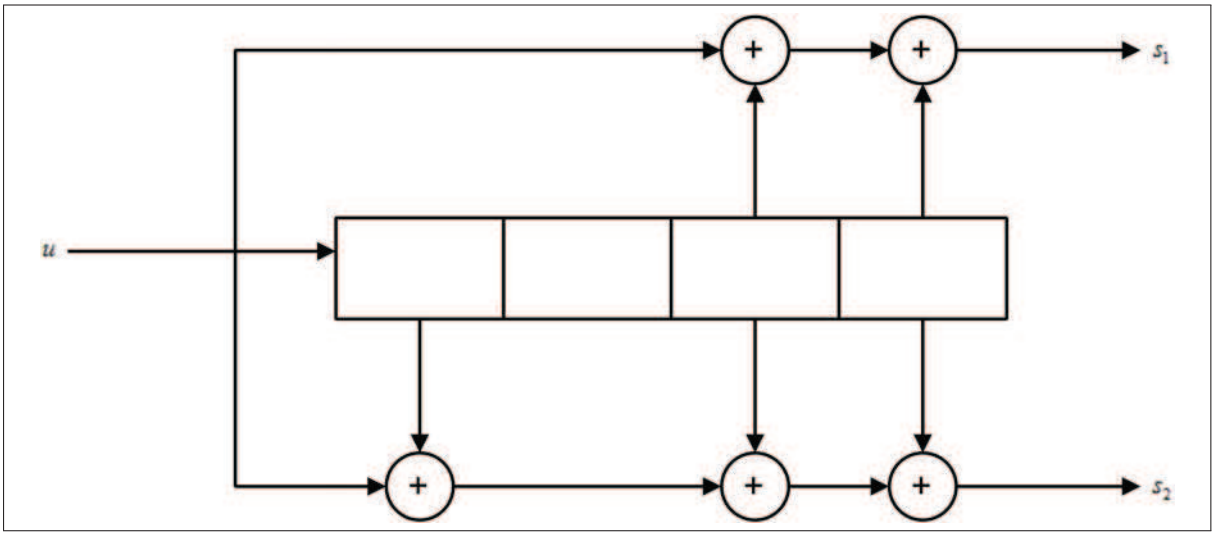

FIGURE 1.15: Codeur utilisé dans le GSM.

corrompues. Ce qui veut dire que le récepteur doit vérifier si l'information reçue est correcte ou erronée. Pour cela, tous les schémas ARQ nécessitent un mécanisme de détection d'erreur[2].

Pour le protocole EGPRS, le codage de canal doit prendre en considération un canal qui varie beaucoup en fonction du temps. Aussi, la modulation et la protection contre les erreurs doivent être ajustables. La donnée transmise est protégée par des codes convolutionnels dont les rendements varient de 0.37 à presque 1.0. Et comme le canal varie très rapidement, le passage d'un schéma de codage et modulation à un autre doit se faire très vite[2].

Chapitre 2

Décodeur de Viterbi

2.1 Introduction

Il existe plusieurs algorithmes de décodage des codes convolutifs. Le plus célèbre est probablement l'algorithme de VITERBI qui repose sur la représentation en treillis des codes [12, 7]. Il permet de trouver, à partir de la séquence des symboles reçus, la séquence d'états dans le treillis la plus probable.

Le décodeur de VITERBI (VD) utilise l'algorithme de VITERBI (VA) pour décoder un flux de bits codés avec un codeur convolutionnel. Il est utilisé dans la CDMA, la GSM, les modems, la communication par satellite (ex. DVB) et le 802.11 Wireless (WiFi 802.11). Le tableau 2.1 résume les paramètres du décodeur pour quelques techniques utilisées dans la communication.

L'algorithme de VITERBI, d'ANDREW VITERBI [1], permet de corriger dans une certaine mesure les erreurs survenues lors d'une transmission à travers un canal bruité.

Son utilisation s'appuie sur la connaissance de la nature du canal, c'est-à-dire la probabilité qu'une information a été modifiée en une autre tout en permettant de simplifier radicalement la complexité de la recherche du message d'origine le plus probable.

2.2 Algorithme de Viterbi

L'algorithme de VITERBI reste la méthode la plus utilisée pour décoder les codes convolutionnels : c'est un décodage dit à maximum de vraisemblance (ou *Maximum Likelihood* en Anglais). Toutefois, l'algorithme de VITERBI reste applicable sur des codes convolutionnels de faible longueur de contrainte ($L \leq 8$). Au-delà de cette limite, sa

1. ANDREA VITERBI devenu après sa naturalisation américaine ANDREW JAMES VITERBI (né à *Bergame* en *Lombardie* le 9 mars 1935) est un ingénieur et entrepreneur américain d'origine italienne. Il est connu pour être l'inventeur de l'algorithme de VITERBI ainsi que pour être l'un des cofondateurs de l'entreprise de télécommunications Qualcomm.

Technologie	Taux R	Longueur de contrainte L	Générateurs g_i
CDMA 2000	$1/4$	9	$g_1 = 765_8$ $g_2 = 671_8$ $g_3 = 513_8$ $g_4 = 473_8$
CDMA IS-95A	$1/2$	9	$g_1 = 153_8$ $g_2 = 561_8$
Cassini	$1/6$	15	$g_1 = 42631_8$ $g_2 = 47245_8$ $g_3 = 56507_8$ $g_4 = 73363_8$ $g_5 = 77267_8$ $g_6 = 64537_8$
UWB (802.15)	$1/3$	7	$g_1 = 171_8$ $g_2 = 165_8$ $g_3 = 133_8$

TABLE 2.1: Paramètres du VD pour certaine normes de communication.

complexité de mise en œuvre impose d'avoir plutôt recours à un algorithme de décodage séquentiel tel que celui de FANO [6].

Dans cette section nous décrivons l'algorithme de VITERBI celui utilisé dans notre travail, mais avant d'entamer cette description, nous allons parler des différents types du VA qui existent.

L'algorithme de VITERBI originel effectue un décodage à sortie ferme, c'est-à-dire qu'il fournit une estimation binaire de chacun des symboles transmis. C'est cet algorithme qui est utilisé dans notre travail ; il s'agit d'un algorithme de décodage à entrée et à sortie fermes.

Des adaptations de l'algorithme de VITERBI telles que celles proposées dans [3], [8] ou [5] ont conduit aux versions à sortie pondérée dites SOVA.

Il existe un troisième algorithme appelé *Forward-Backwad*, il s'agit d'un algorithme à entrée et sortie souples (SISO) : c'est l'algorithme MAP. L'algorithme MAP permet le calcul de la valeur exacte de la probabilité à postériori associée à chaque symbole transmis en utilisant la séquence reçue [10].

2.2.1 Principe de l'algorithme

Un mot code est un chemin faisant partie du diagramme en treillis, possédant un état de départ et un état d'arrivée.

Le décodage ML est basé sur la recherche du mot de code u qui a la plus petite distance du mot reçu. Il est important de connaître la nature du canal de transmission,

car dans le cas d'un canal binaire symétrique, le décodage à maximum de vraisemblance (ML) s'appuie sur la distance de HAMMING, alors que dans celui d'un canal gaussien, il s'appuie sur la distance euclidienne.

Le décodage consiste à trouver le chemin le plus proche, appartenant au treillis, de la séquence reçue à décoder. Il explore le treillis en comparant tous les chemins possibles avec la séquence reçue et sélectionne celui qui est le plus vraisemblable.

L'algorithme de VITERBI permet d'apporter une réduction notable de la complexité de calcul. Il est basé sur l'idée que, parmi l'ensemble des chemins du treillis qui convergent en un état à un instant donné, seul le chemin le plus probable peut être retenu pour les étapes suivantes de recherche. Cette méthode est appelée stockage des chemins survivants.

Soit b un message code transmis sur un canal BSC avec une probabilité d'erreur p et r le message code correspondant reçu. Supposons que b et r sont de même longueur n et z est la distance de HAMMING entre eux, la fonction log-vraisemblance, $\log \Pr(b \mid r)$, est donnée par la formule suivante [4] :

$$\begin{aligned} \log \Pr(b \mid r) &= \log \left[p^z (1-p)^{n-z} \right] && (2.1) \\ &= n \log(1-p) - z \log \left(\frac{1-p}{p} \right) && (2.2) \end{aligned}$$

ou bien :

$$\log \Pr(b \mid r) = -A - Bz \tag{2.3}$$

Où $p \leq 0.5$, ce qui implique que $A,\ B > 0$.

Donc pour maximiser la fonction log-vraisemblance, il faut minimiser la distance de HAMMING.

Il est important de parler de quelques notions de métriques essentielles au décodage par algorithme de VITERBI : Pour effectuer un décodage par algorithme de VITERBI on a besoin de deux métriques :

Métrique de branche

C'est la distance de HAMMING (cas d'un Canal Binaire Symétrique) entre le symbole reçu (symbole de n bits) et tous les symboles possibles de sortie du codeur. Donc, à chaque instant i, on doit calculer 2^n métriques de branches.

Le symbole de sortie correspond à une transition d'un état a_p à l'instant $i-1$, noté $a_p^{(i-1)}$ vers un état a_q à l'instant i, noté $a_q^{(i)}$. Ainsi la notation de la métrique de branche entre le symbole reçu (affecté par le bruit) et le symbole qui correspond à une transition quelconque est :

$$BM\left(a_p^{(i-1)},\ a_q^{(i)}\right)_{p \text{ prend } 2^k \text{ valeurs différentes}} \tag{2.4}$$

Métrique de chemin (ou métrique d'état)

La métrique de chemin d'un état a_q à l'instant i, notée $PM\left(a_q^{(i)}\right)$ est le cumul des métriques de branches des instants précédents conduisant à cet état tout en respectant le critère des chemins survivants.

Pour un codeur de rendement $R = {}^{k}\!/_{n}$, on a chaque état $a_q^{(i)}$ généré au maximum par 2^k états précédents, notés $a_p^{(i-1)}$. Donc pour calculer la métrique de chemin $PM\left(a_q^{(i)}\right)$, il faut cumuler les métriques de branches précédentes, les plus petites. Pour ce faire, on applique la loi suivante :

$$PM\left(a_q^{(i)}\right) = \min\left\{PM\left(a_p^{(i-1)}\right) + BM\left(a_p^{(i-1)},\ a_q^{(i)}\right)\right\}_{p \text{ prend } 2^k \text{ valeurs différentes}} \tag{2.5}$$

$PM\left(a_q^{(i)}\right)$: métrique de chemin de l'état a_q à l'instant i ;

$PM\left(a_p^{(i-1)}\right)$: métriques de chemin de l'état a_p à l'instant $i-1$;

$BM\left(a_p^{(i-1)},\ a_q^{(i)}\right)$: métrique de branche de la transition de l'état $a_p^{(i-1)}$ vers l'état $a_q^{(i)}$.

Cette opération décrite dans l'équation (2.5) est appelé ACS, illustrée dans la Figure 2.1 pour un codeur de rendement $R = {}^{1}\!/_{n}$.

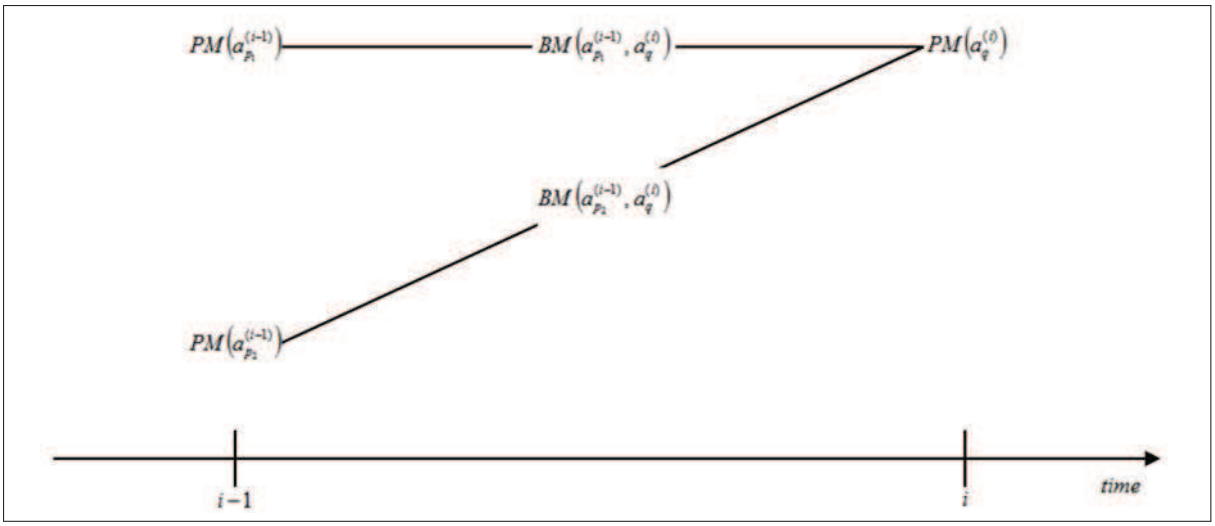

FIGURE 2.1: Illustration de l'opération ACS Taux $R = {}^{1}\!/_{n}$.

Après avoir vu ces deux définitions des métriques, on va voir étape par étape comment se fait le décodage :

1. Initialisation : Il s'agit d'initialiser le décodeur au même état initial que le codeur ; généralement c'est l'état 0. Il faut aussi attribuer une métrique de chemin nulle à l'état initial.
2. Calcul des métriques de branches : Pour chaque symbole de n bits on calcule les métriques de branches entre le symbole reçu (affecté par le bruit) et les 2^n symboles probable.

3. Calcul des métriques de chemins : Après avoir calculé les métriques de branches, on calcule les métriques de chaque état : On additionne chaque métrique de branche à la métrique d'état précédente. Cet état précédent doit correspondre au même état du début de transition avec laquelle on a calculé la métrique de branche pour avoir la métrique de chemins qui correspond à l'état de la fin de transition.
4. Stockage des survivants : Après l'étape précédente on aura pour chaque état a_q, 2^k métriques de chemin. Parmi ces 2^k métriques de chemins, on doit sélectionner la plus petite et bien sur ce souvenir de la décision appliquée.
5. Décodage : Après avoir exécuté les étapes 2 à 4 pour toutes la séquence reçue (prise symbole par symbole de n bits). On cherche dans la dernière colonne du treillis l'état qui a la métrique de chemins la plus petite. De cet état on remonte le treillis en utilisant les décisions de l'étape 4 jusqu'à l'état initial. Remarquez que la séquence décodée est en inverse, car nous commençons par la fin du treillis.

En résumé pour réaliser le décodage de VITERBI, il faut calculer les métriques de branches puis les accumuler en sélectionnant la métrique de chemin la plus petite pour chaque état et à n'importe quel instant. C'est le stockage des chemins survivants. A la fin il faut sélectionner le chemin qui a la métrique la plus petite, puis remonter le treillis à partir de cet état jusqu'à l'état initial.

2.2.2 Exemple de décodage

Soit la séquence à coder $u = 111001$ par un codeur de longueur de contrainte $L = 4$ (ce qui correspond à une mémoire $m = 3$ et un nombre d'états $N = 2^3 = 8$), un rendement $R = 1/2$ ($k = 1$ et $n = 2$), et les polynômes générateurs $g_1 = 15_8$ et $g_2 = 13_8$, ce qui donne une séquence codée $b = 111010101101$. Ce message b est transmis à travers un canal BSC avec une probabilité d'erreur $p = 0.2$; le message reçu est $r = 111010111101$.

Initialement le codeur et à l'état 000 ; donc il faut que le décodeur démarre du même état avec une métrique de chemin nulle. Le premier symbole reçu est 11 ; il faut donc calculer les métriques de branches entre ce symbole et tous les symboles possibles(c'est-à-dire 00, 01, 10 et 11). Il faut refaire ce même calcul pour les cinq prochaines itérations et pour chaque état, à condition qu'il soit déjà généré, en respectant toujours la minimisation de la métrique de chemins. Les Figures 2.2 à 2.7 illustrent le treillis à chaque instant.

FIGURE 2.2: État du treillis à l'instant $i = 1$.

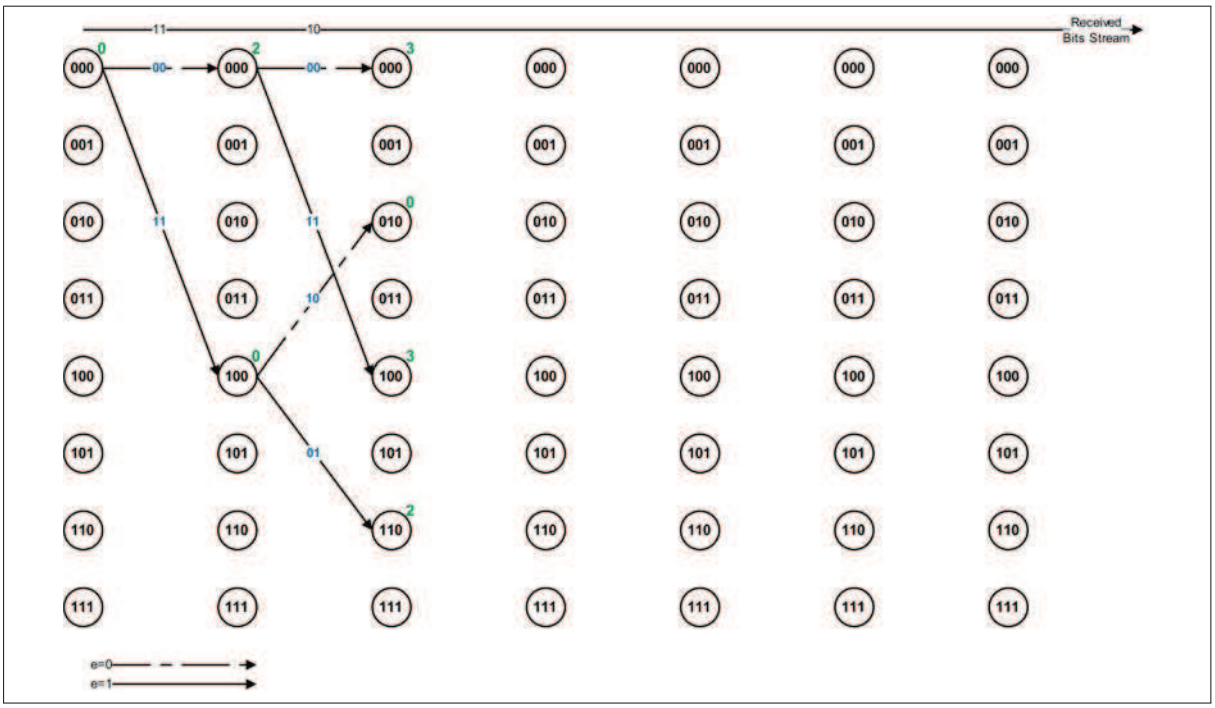

FIGURE 2.3: État du treillis à l'instant $i = 2$.

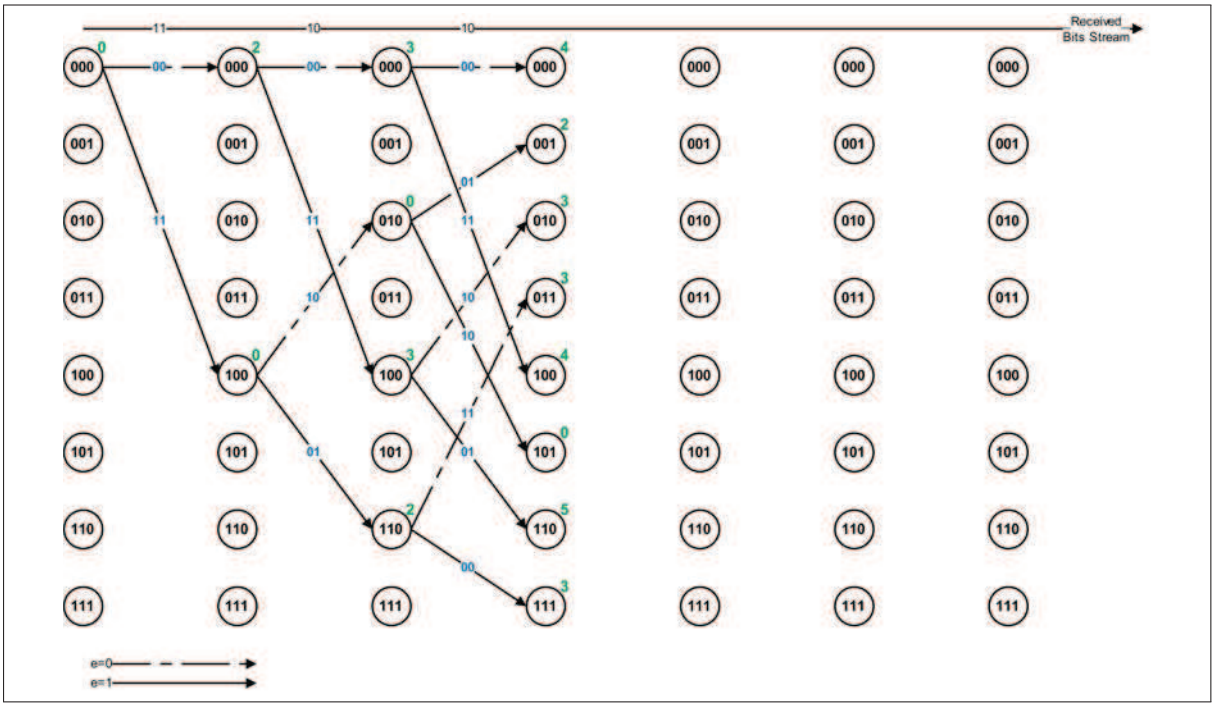

FIGURE 2.4: État du treillis à l'instant $i = 3$.

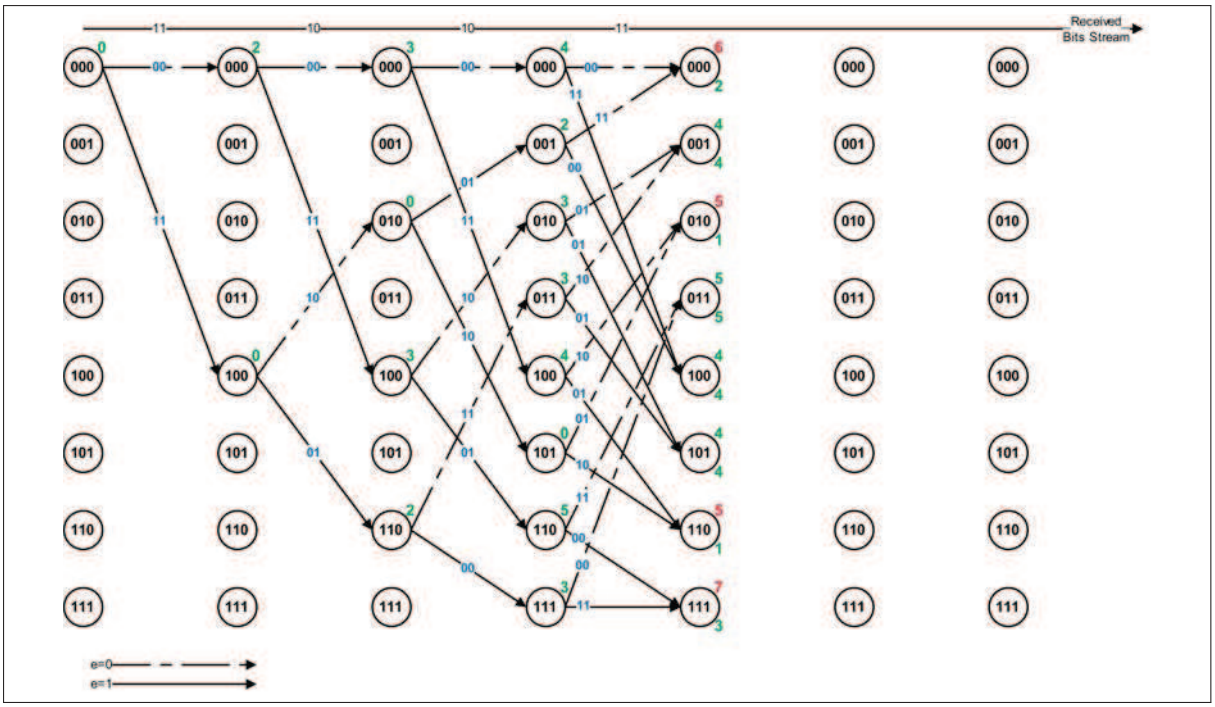

FIGURE 2.5: État du treillis à l'instant $i = 4$.

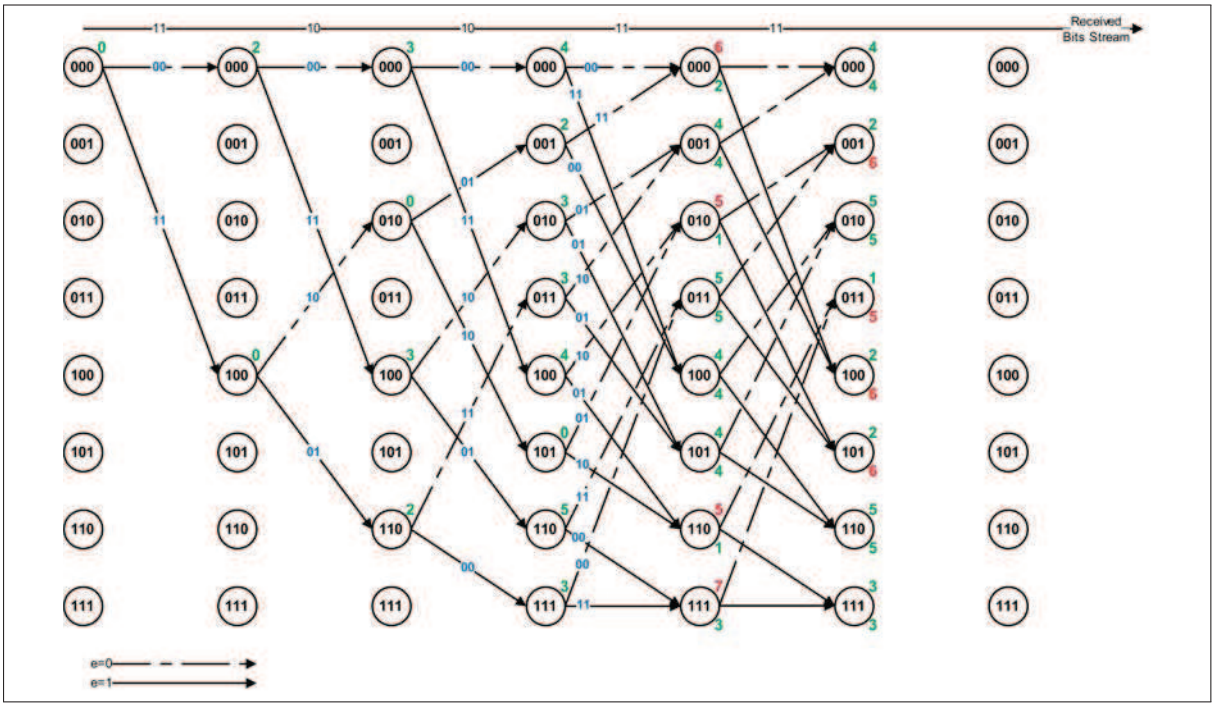

FIGURE 2.6: État du treillis à l'instant $i = 5$.

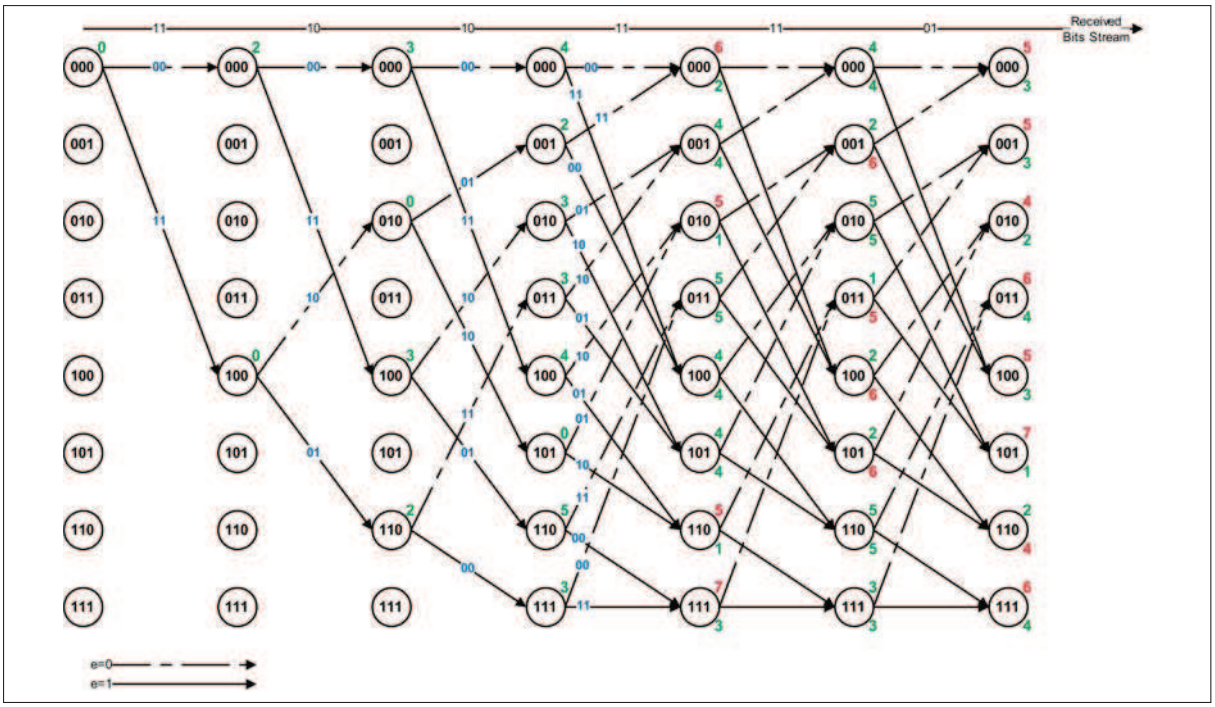

FIGURE 2.7: État du treillis à l'instant $i = 6$.

Après la sixième itération on remonte le treillis en démarrant de l'état qui possède la plus petite métrique de chemin, c'est-à-dire l'état 101. La Figure 2.8 illustre cette opération.

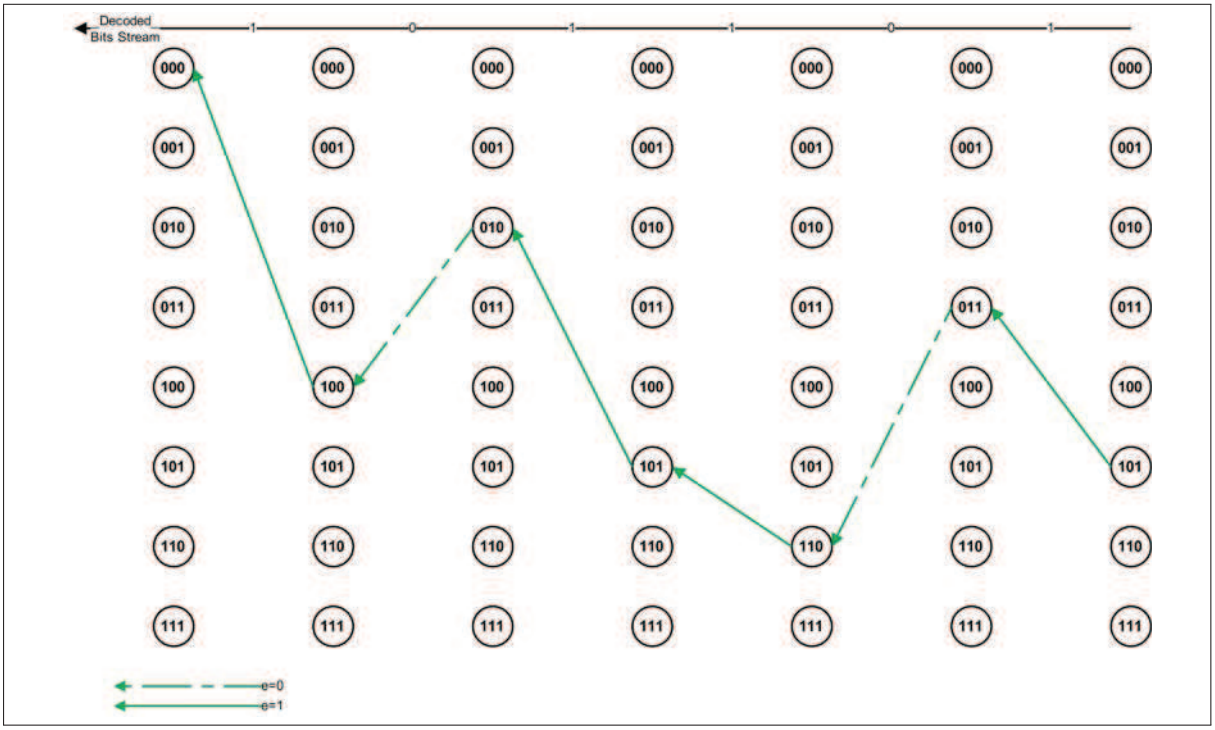

FIGURE 2.8: Décodage en remontant le treillis.

Ainsi la séquence décodée ($\widehat{u} = 101101$) est bien la même que la séquence codée, bien qu'il existe un bit erroné à la huitième position de la séquence codée.

2.3 Schéma fonctionnel du décodeur

Après avoir parlé du principe de l'algorithme de VITERBI, nous allons voir les modules qui constituent le décodeur de VITERBI du point de vue implémentation. Pour réaliser le décodeur de VITERBI il faut souvent implémenter trois modules fonctionnels. Ces trois modules sont représentés dans la Figure 2.9.

D'après la Figure 2.9 on voit bien que le décodeur de VITERBI est un montage en cascade de trois modules :

– **BMU :** C'est l'Unité de Calcul des Métriques de Branches. Elle permet de calculer la métrique de branche c'est-à-dire ; la distance de HAMMING entre le mot code reçu (affecté par le bruit) et le mot code exact correspondant à une transition quelconque du diagramme en treillis. Pour chaque symbole reçu de n bit, elle calcule

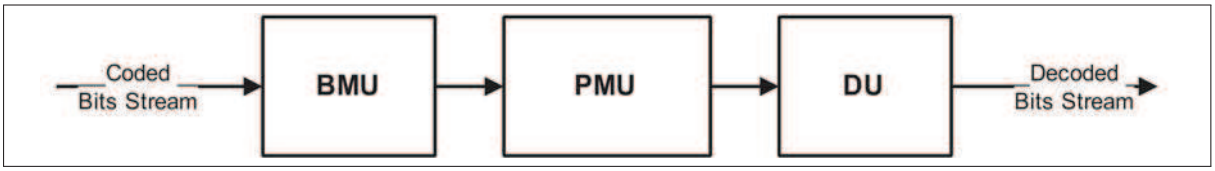

FIGURE 2.9: Illustration des différents modules du VD.

2^n métriques de branches et les fait passer à la PMU pour calculer les métriques de chemins.

- **PMU :** Unité de calcul des Métriques de Chemin, appelée aussi ACSU, constituée principalement des ACS. Ce module additionne les métriques de branches aux précédentes métriques de chemin et sélectionne la métrique la plus petite pour chaque état, puis met à jour les nouvelles métriques. Aussi cette unité délivre à chaque instant i, 2^m bits de décision qui correspondent aux chemins choisis.
- **DU :** C'est l'Unité de Décision. Ce module peut être réalisé par deux approches :
  - *Trace Back* (TB) : Cette approche consiste à remonter le treillis à partir de l'état final qui possède la métrique d'état la plus petite en utilisant les bits de décisions délivrées par la PMU pour tracer le chemin le plus vraisemblable à la séquence reçue.
  - *Register Exchange* (RE) : Cette méthode consiste a allouer pour chaque état un registre qui contient les bits de la séquence décodée à chaque instant. A la fin la séquence décodée se trouve dans le registre qui correspond à l'état qui a la plus petite métrique de chemin. Cette méthode utilise beaucoup de mémoire. Cette mémoire augmente exponentiellement avec la mémoire du code, car pour un codeur de mémoire m, on alloue 2^m registres d'une certaine longueur.

2.3.1 Unité de Calcul des Métriques de Branches

Pour chaque symbole reçu de n bit, la BMU calcule 2^n métriques de branches. Il existe plusieurs types de métriques ; celle qui nous intéresse est la distance de HAMMING.

Distance de Hamming

La distance de HAMMING, définie par RICHARD HAMMING, est utilisée en informatique en traitement du signal et dans les télécommunications. Elle joue un rôle important en théorie algébrique des codes correcteurs. Elle permet de quantifier la différence entre deux séquences de symboles ou tout simplement elle calcule le nombre de bits différents entre deux séquences binaires de même longueur. Cette distance est utilisée dans le cas d'une transmission dans un canal BSC.

Donc pour implémenter la BMU, on doit réaliser une fonction qui calcule la distance de HAMMING entre le symbole reçu et le symbole attendu, stocké dans une ROM. Cette

fonction est réalisée par des XOR et un additionneur. La Figure 2.10 illustre le schéma fonctionnel de la BMU pour $n = 2$.

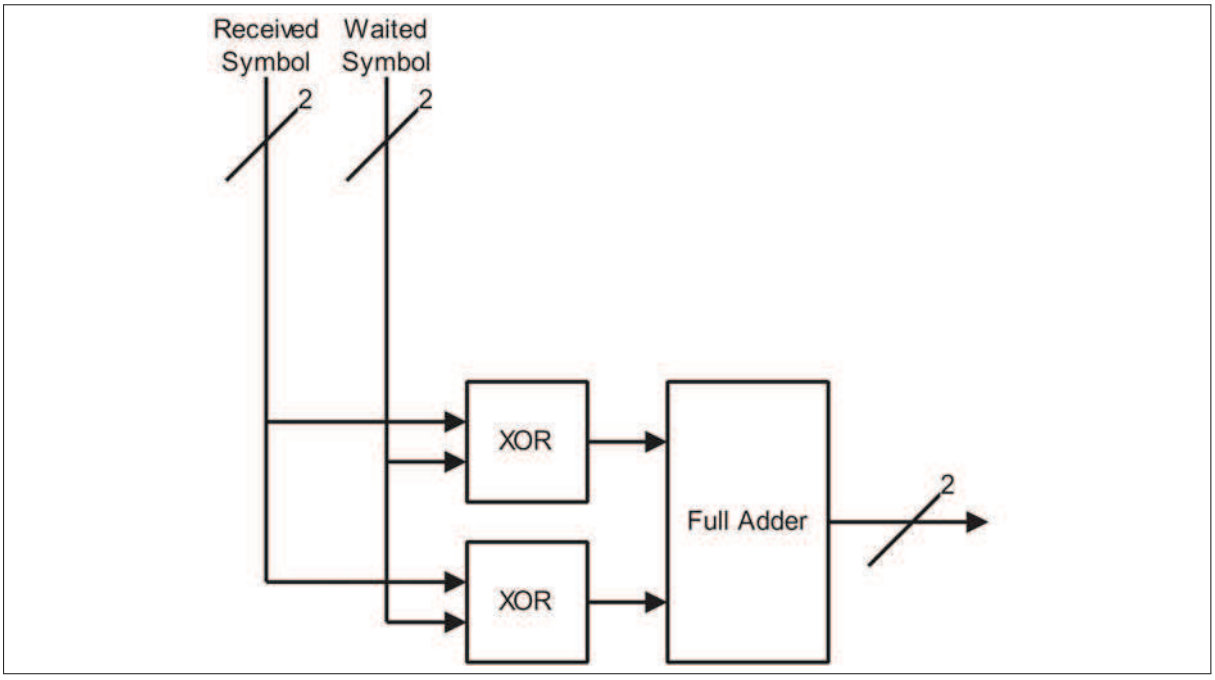

FIGURE 2.10: Illustration de la BMU.

2.3.2 Unité de calcul des Métriques de Chemin

Cette unité calcule et sélectionne le chemin le plus vraisemblable parmi les 2^k chemins entrants dans chaque état. Elle fournit également à chaque instant i, k bits de décisions renseignant nous sur les chemins choisis. Cette principale fonction est réalisée par le sous-module ACS.

L'ACS est constitué de 2^k additionneurs pour calculer toutes les métriques d'état possibles, un comparateur qui sélectionne la position de la métrique d'état la plus petite (sélection des survivants) et un multiplexeur qui fournit en sortie la métrique la plus petite qui sera stockée dans la RAM, pour être utilisé dans la prochaine itération. La Figure 2.11 illustre le schéma bloc de l'opération ASC pour $R = 1/n$.

Donc pour réaliser la PMU (ou ACSU), on doit réaliser toutes les ACS du treillis. Souvent on regroupe les ASC qui possèdent les mêmes états de départ et utilisent les mêmes métriques de branches pour n'en faire qu'un module appelé papillon (*Butterfly*), La Figure 2.12 illustre le regroupement des papillons pour un treillis de $N = 2^3 = 8$, et un $R = 1/2$.

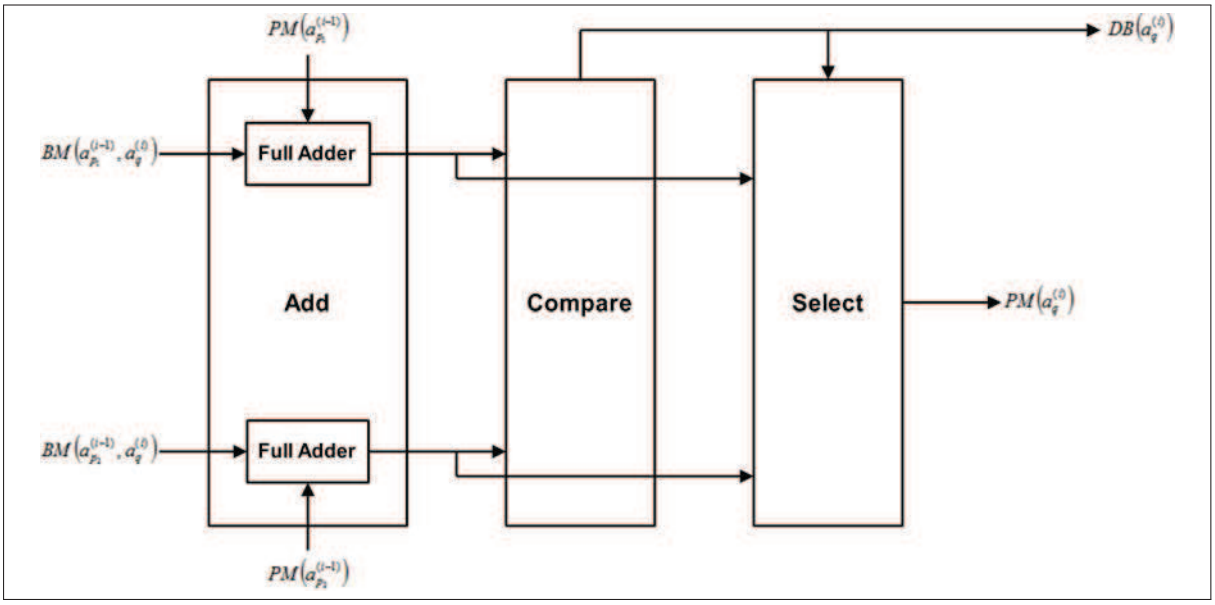

FIGURE 2.11: Schéma de l'ACS pour $R = 1/n$.

2.3.3 Unité de Décision

Dans cette unité on génère la séquence décodée. Dans notre travail, pour cette unité on a choisie l'approche *Trace Back.* Donc pour réaliser la DU on doit implémenter une fonction qui recherche la position du minimum et donne en sortie l'état correspondant, et un module qui n'est autre qu'un registre à décalage (décalage à gauche), qui prend en entrée les bits de décision, et délivre en sortie une séquence de bits, c'est la séquence décodée. Ce registre à décalage doit être initialisé à l'état calculé par la fonction de recherche de du minimum. On remarque dans la Figure 2.8 que la séquence décodée est générée en inverse ; donc on doit ajouter une pile LIFO dans la DU pour inverser la séquence. La Figure 2.13 illustre le schéma bloc de cette unité.

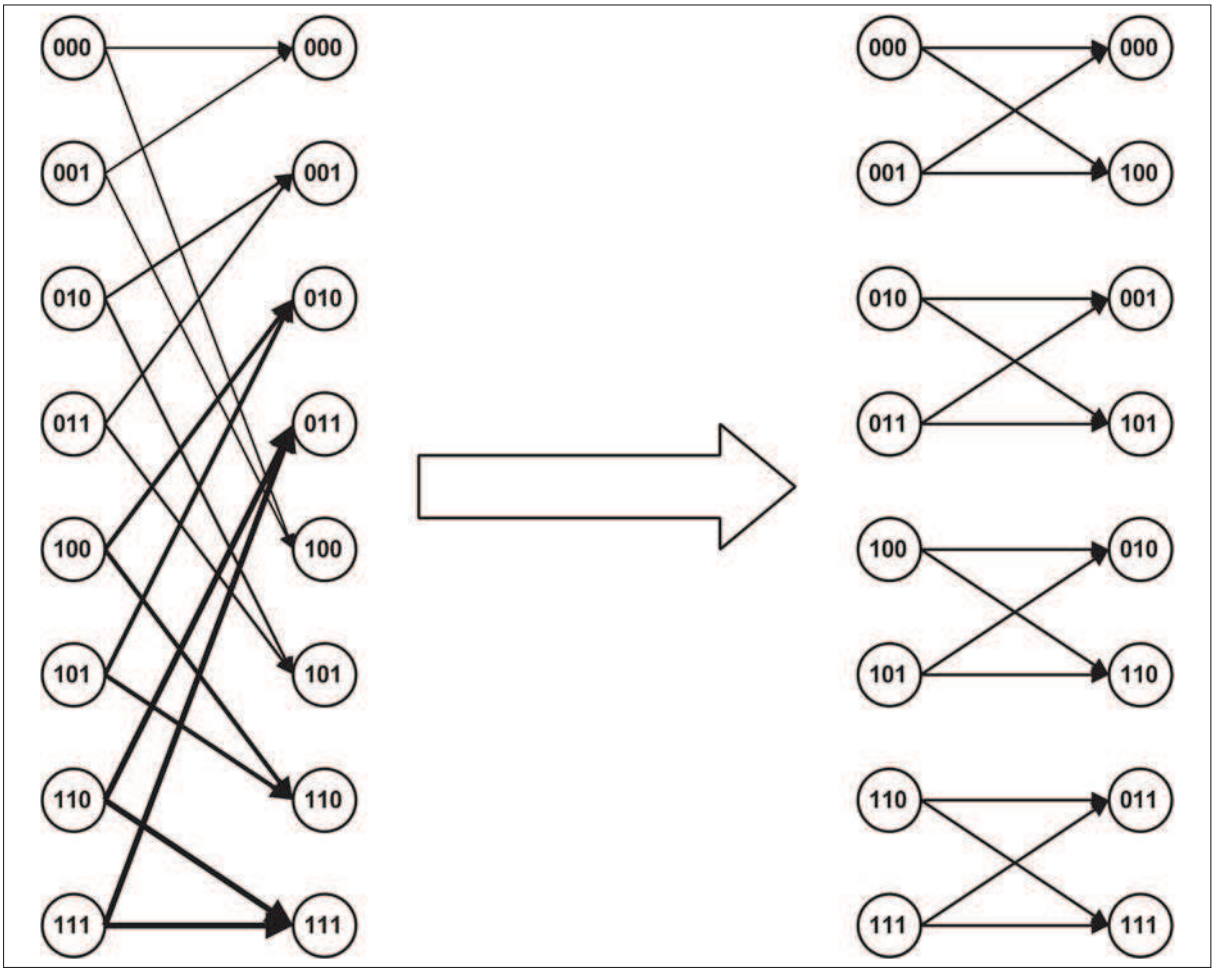

FIGURE 2.12: Regroupement des papillons pour un treillis $N = 8$ et $R = 1/2$

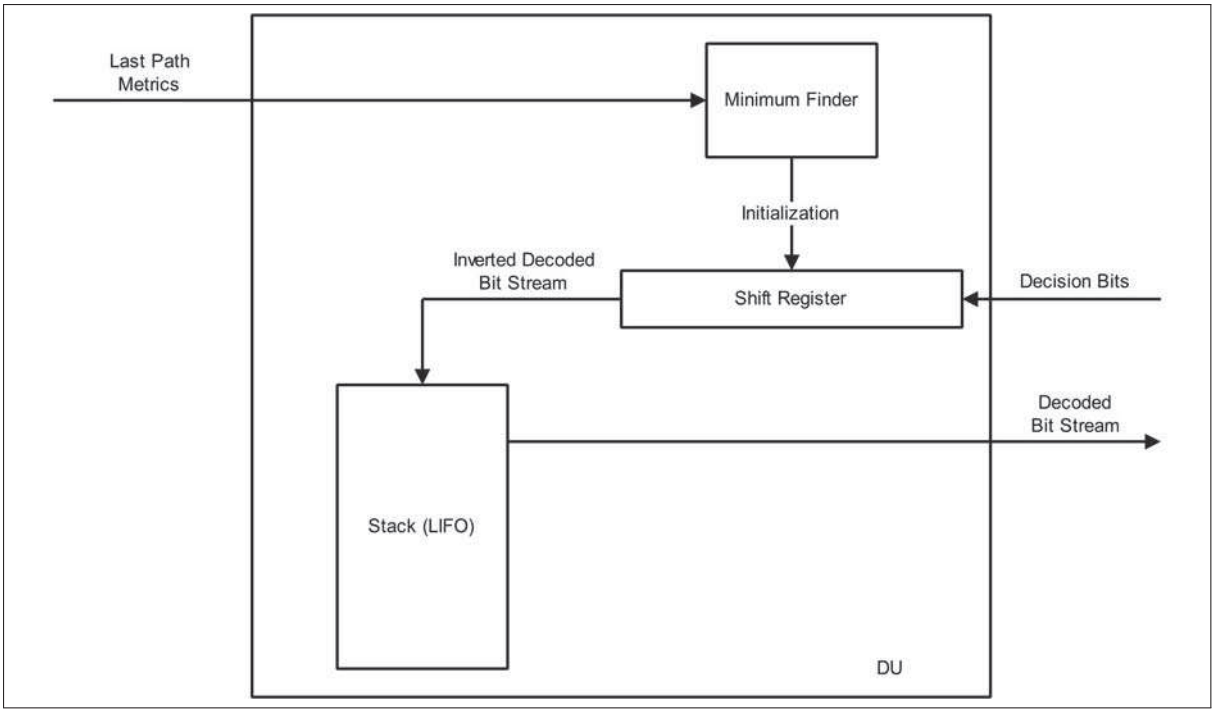

FIGURE 2.13: Illustration de la DU.

Chapitre 3

Architecture proposée

3.1 Introduction

La structure et le principe de fonctionnement du décodeur de VITERBI ont été décrits de façon détaillée dans le chapitre précédent. Dans ce chapitre non seulement nous parlerons de l'architecture proposée, mais nous passerons aussi en détail tous ces composants qui constituent chaque unité en définissant l'unité de contrôle qui synchronise et gère les unités du décodeur.

Dans notre travail, le décodeur est capable de décoder une séquence de bit codée avec un codeur convolutionnel de rendement $R = 1/2$, une mémoire $m = 3$ (ce qui donne un nombre d'état $N = 2^3 = 8$ et une longueur de contrainte $L = 4$) et des polynômes générateurs $g_1 = 15_8$ et $g_2 = 13_8$.

3.2 Schéma proposé du décodeur

L'architecture en bloc du décodeur de VITERBI est souvent celle montrée à la Figure 2.9. Dans notre travail nous avons assemblé la BMU et la PMU pour n'en faire qu'une seule unité appelée Unité de Calcul des Métriques (*Metrics Calculating Unit* ou MCU). La Figure 3.1 illustre le schéma en bloc de notre décodeur.

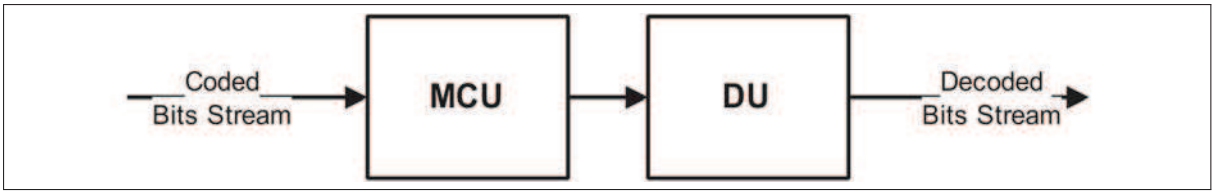

FIGURE 3.1: Schéma bloc du décodeur.

Pour cette architecture l'unité de calcul des métriques de branches est incluse dans

l'unité de calcul des métriques de chemins. Plus précisément le calcul des métriques de branches se fait à l'intérieur des papillons. Chaque papillon peut calculer les deux métriques de branches dont il a besoin.

3.3 Unité de Calcul des Métriques

Cette unité calcule les métriques de branche ainsi que les métriques de chemins et sauvegarde les chemins survivants à chaque instant et pour chaque état en stockant les bits de décision. Aussi doit-elle stocker les métriques de chaque état à chaque instant pour pouvoir les utiliser lors de la prochaine itération. Pour inclure la BMU dans la PMU, qui n'est autre que quatre papillons, il faut que les papillons soient capables de calculer les métriques de branches.

3.3.1 ACS à deux chemins

L'ACS à deux chemins (2-ways ACS) est l'élément de base pour réaliser chaque état du treillis. Sa principale fonction est de calculer la métrique de chemin $PM\left(a_q^{(i)}\right)$ de l'état $a_q^{(i)}$ en partant de deux métriques de chemins $PM\left(a_{p_1}^{(i-1)}\right)$ et $PM\left(a_{p_2}^{(i-1)}\right)$ qui correspondent aux états $a_{p_1}^{(i-1)}$ et $a_{p_2}^{(i-1)}$ respectivement. Donc l'ACS effectue deux additions simultanément. Il additionne une métrique de chemin de l'instant précèdent d'un état quelconque avec la métrique de branche qui correspond à la transition de cet état vers l'état futur. Les deux additions qu'effectue l'ACS sont données par les relations suivantes :

$$NPM_1\left(a_q^{(i)}\right) = PM\left(a_{p_1}^{(i-1)}\right) + BM\left(a_{p_1}^{(i-1)},\ a_q^{(i)}\right) \tag{3.1}$$

$$NPM_2\left(a_q^{(i)}\right) = PM\left(a_{p_2}^{(i-1)}\right) + BM\left(a_{p_2}^{(i-1)},\ a_q^{(i)}\right) \tag{3.2}$$

Après avoir calculé ces deux métriques, l'ACS doit effectuer une comparaison entre elles afin de choisir la plus petite métrique. Aussi, pour sauvegarder les chemins survivants, cette décision doit être stockée dans une mémoire (la RAM de décision qui se trouve dans la DU). La Figure 3.2 est l'organigramme appliqué par l'ACS pour effectuer les trois opérations. Le schéma détaillé de l'ACS est illustré dans la Figure 3.3.

3.3.2 Papillons

Comme nous l'avons décrit précédemment, le papillon est tout simplement la combinaison de deux ACS possédant les mêmes états de départ et le même état d'arrivée. La Figure 3.4 illustre l'architecture d'un papillon dans le cas général.

Notre architecture de papillon est un peu modifiée de l'architecture connue. Avant d'effectuer l'opération ACS, chaque papillon adresse la ROM afin de lire les deux symboles

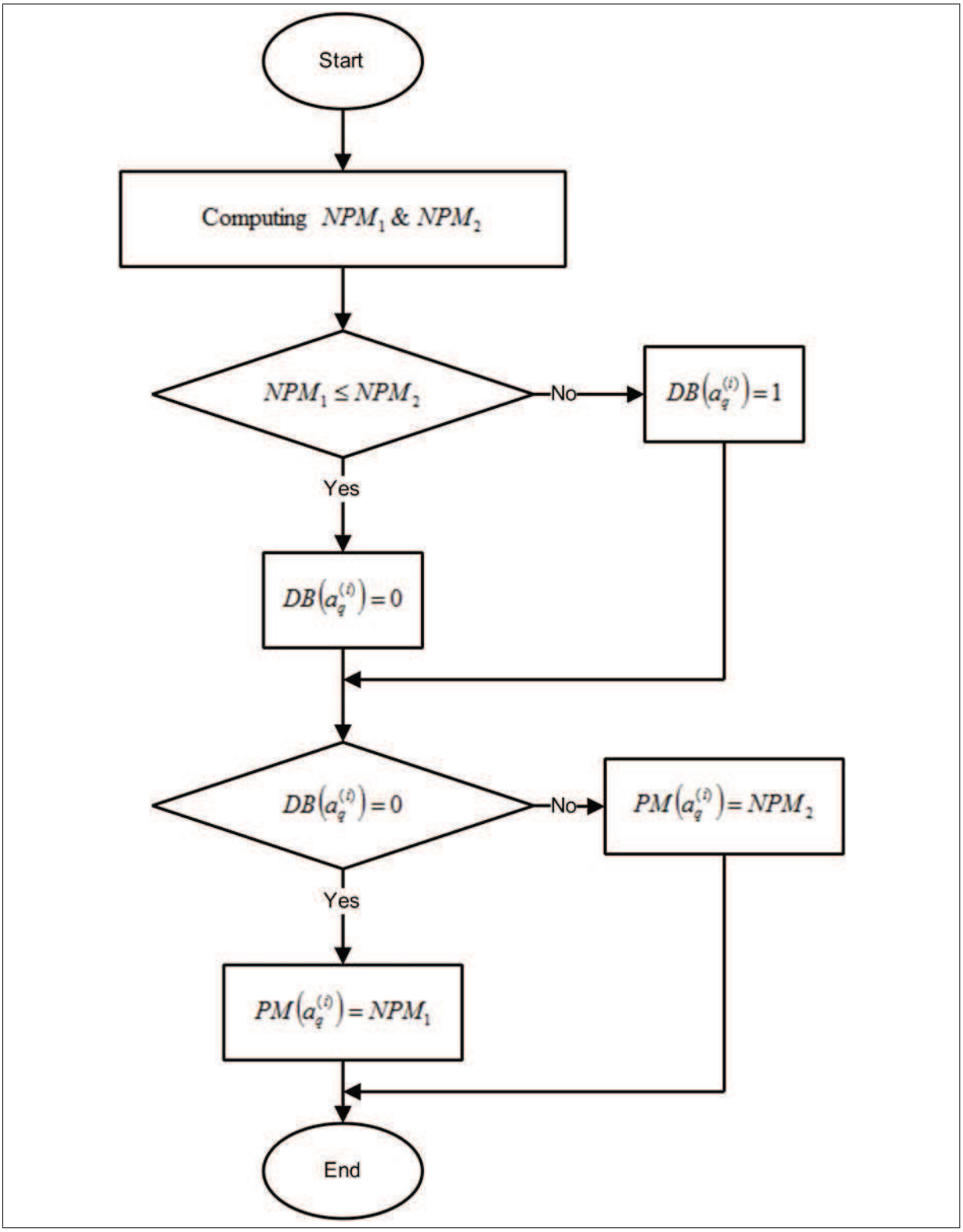

FIGURE 3.2: Organigramme appliqué par l'ACS.

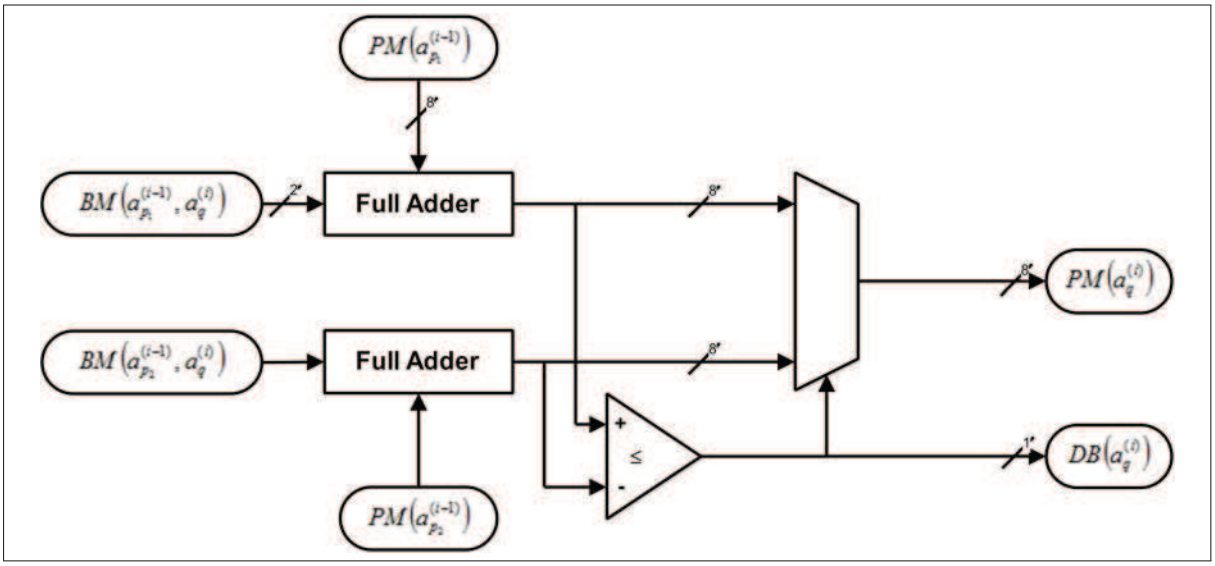

FIGURE 3.3: Schéma détaillé de l'ACS.

attendus. Pour calculer les deux métriques de branches essentielles à l'opération ACS. De ce fait, ce papillon effectue non seulement l'opération ACS, mais calcule au préalable les métriques de branches. La Figure 3.5 illustre l'architecture détaillée du papillon.

En ce qui concerne le fonctionnement des papillons un problème survient lors des premières itérations _exactement $i \leq 2$_ car il existe des états qui ne sont pas encore valides. Le décodeur démarre pour état initial 000, donc tous les autres états ne sont pas valides. Donc le papillon qui effectue le calcul des nouvelles métriques de chemins pour deux états de départ quelconque doit retourner une métrique nulle tant qu'il n'existe pas au moins un état valide à l'entrée de ce même papillon. Pour régler ce problème nous avons eu recours à un registre d'état (*Flag Register*) à l'intérieur de la MCU. Ce registre est de taille 8 bits (un bit pour chaque état). Chacun des bits informe que l'état qui correspond à sa position est actif ou non. Donc l'opération ACS s'effectue seulement sur les états qui sont valides, pour ce faire on doit tester les bits du registre à chaque symbole reçu.

3.3.3 Registre d'état

Après avoir parlé d'un seul élément qui constitue la MCU, nous allons examiner le rôle et l'interconnexion du registre d'état. Après l'utilisation du registre d'état, le multiplexeur de la Figure 3.2.1 doit être modifié pour faire une sélection non seulement avec le bit issu du comparateur (bit de comparaison) mais aussi avec les bits flag. Comme nous l'avons déjà cité avant, un état à l'instant i est généré au maximum par deux états de l'instant $i-1$. Il se peut que l'un de ces deux états ne soit pas valide, donc même le bit de décision va être généré en fonction des bits flags. Le Tableau 3.1 résume toutes

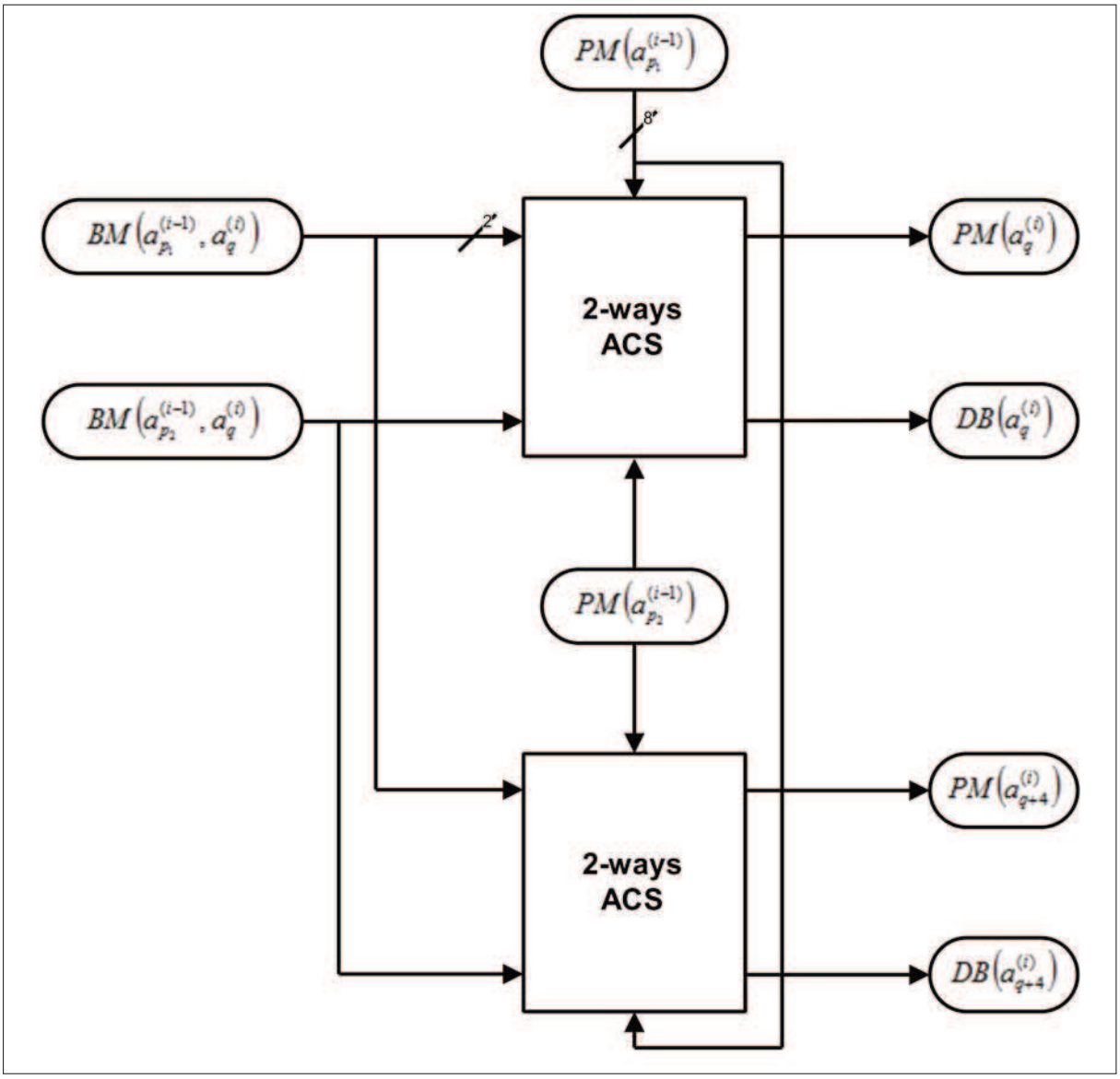

FIGURE 3.4: Architecture d'un papillon.

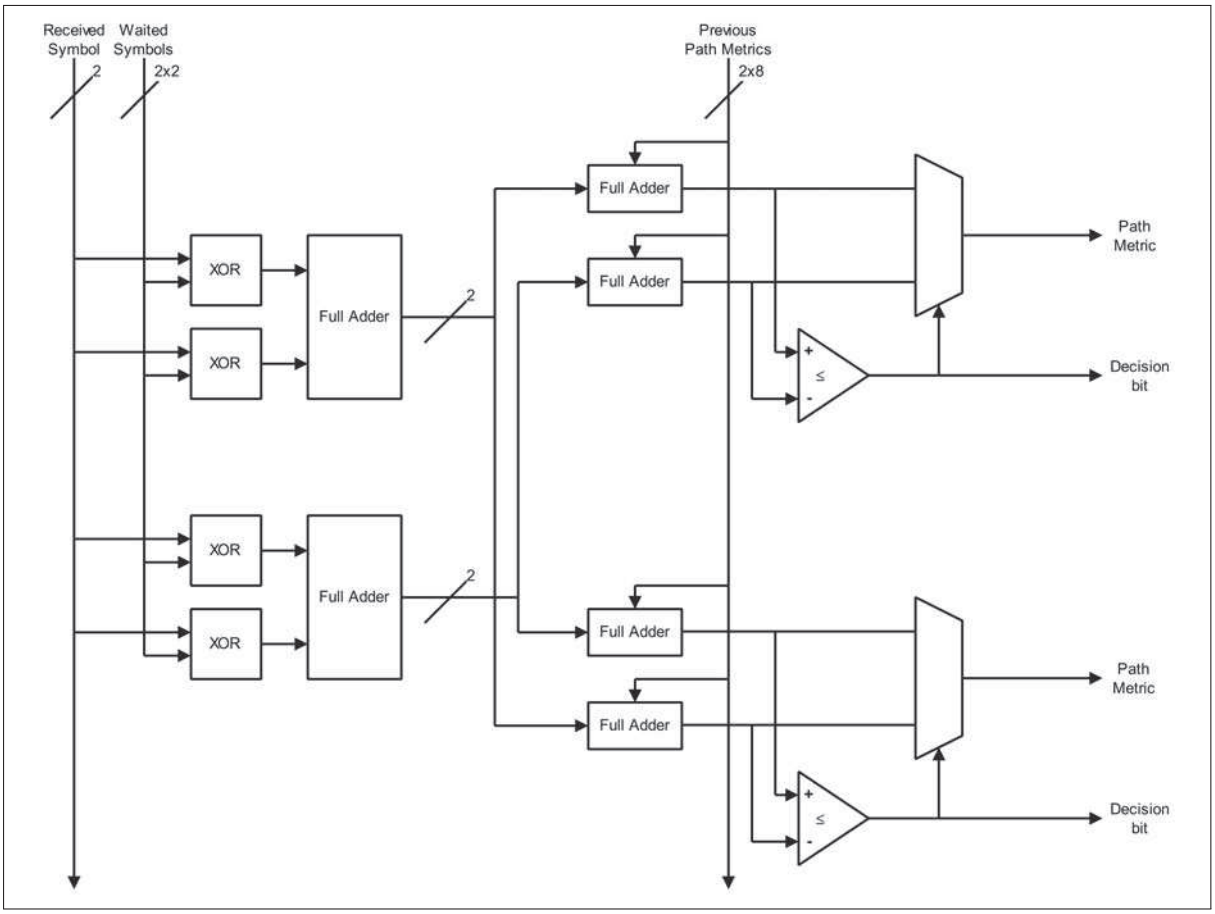

FIGURE 3.5: Architecture détaillée du papillon.

les possibilités et les résultats pour une seule opération ACS. Nous devons aussi ajouter un élément dans la MCU qui, en fonction des bits d'état, génère un bit d'état informant sur la validité de l'état généré.

$Flag\left(a_{p_1}^{(i-1)}\right)$	$Flag\left(a_{p_2}^{(i-1)}\right)$	Bit de comparaison	$PM\left(a_q^{(i)}\right)$	$DB\left(a_q^{(i)}\right)$	$Flag\left(a_q^{(i)}\right)$
0	0	X	0	0	0
1	0	X	NPM_1	0	1
0	1	X	NPM_2	1	1
1	1	0	NPM_1	0	1
1	1	1	NPM_2	1	1

Table 3.1: Résultats pour une seule opération ACS

$Flag\left(a_{p_1}^{(i-1)}\right)$: bits informant que l'état $a_{p_1}^{(i-1)}$ est valide ou non ;
$Flag\left(a_{p_2}^{(i-1)}\right)$: bits informant que l'état $a_{p_2}^{(i-1)}$ est valide ou non ;
$PM\left(a_q^{(i)}\right)$: métrique de chemin pour l'état $a_q^{(i)}$;
NPM_1 : New Path Metric 1 ;

$$NPM_1\left(a_q^{(i)}\right) = PM\left(a_{p_1}^{(i-1)}\right) + BM\left(a_{p_1}^{(i-1)},\ a_q^{(i)}\right) \tag{3.3}$$

NPM_2 : New Path Metric 2 ;

$$NPM_2\left(a_q^{(i)}\right) = PM\left(a_{p_2}^{(i-1)}\right) + BM\left(a_{p_2}^{(i-1)},\ a_q^{(i)}\right) \tag{3.4}$$

$DB\left(a_q^{(i)}\right)$: bit de décision pour l'état $a_q^{(i)}$;
$Flag\left(a_q^{(i)}\right)$: bit informant que l'état $a_q^{(i)}$ sera valide ou non.
La Figure 3.6 illustre toutes les modifications apportées à l'unité de calcul des métriques pour un seul éléments ACS.

3.3.4 Architecture de la MCU

Dans cette section nous assemblerons les quatre papillons, la mémoire de stockage des métriques de chemins et le registre d'état pour avoir l'architecture finale de l'unité de calcul des métriques. Tout ceci est illustré dans la Figure 3.7.

Il y'a lieu de noter que pour des contraintes de mémoire nous devons limiter la taille des métriques de chemins. Dans notre travail nous avons attribué une taille de 8 bits pour les métriques de chemins, ce qui donne une métrique maximale égale à 255. Cette métrique maximale est le cumul de 127 fois environ la métrique de branche maximale ($255 \simeq 127 \times 2$), ce qui donne une longueur minimale de 127 bits à transmettre.

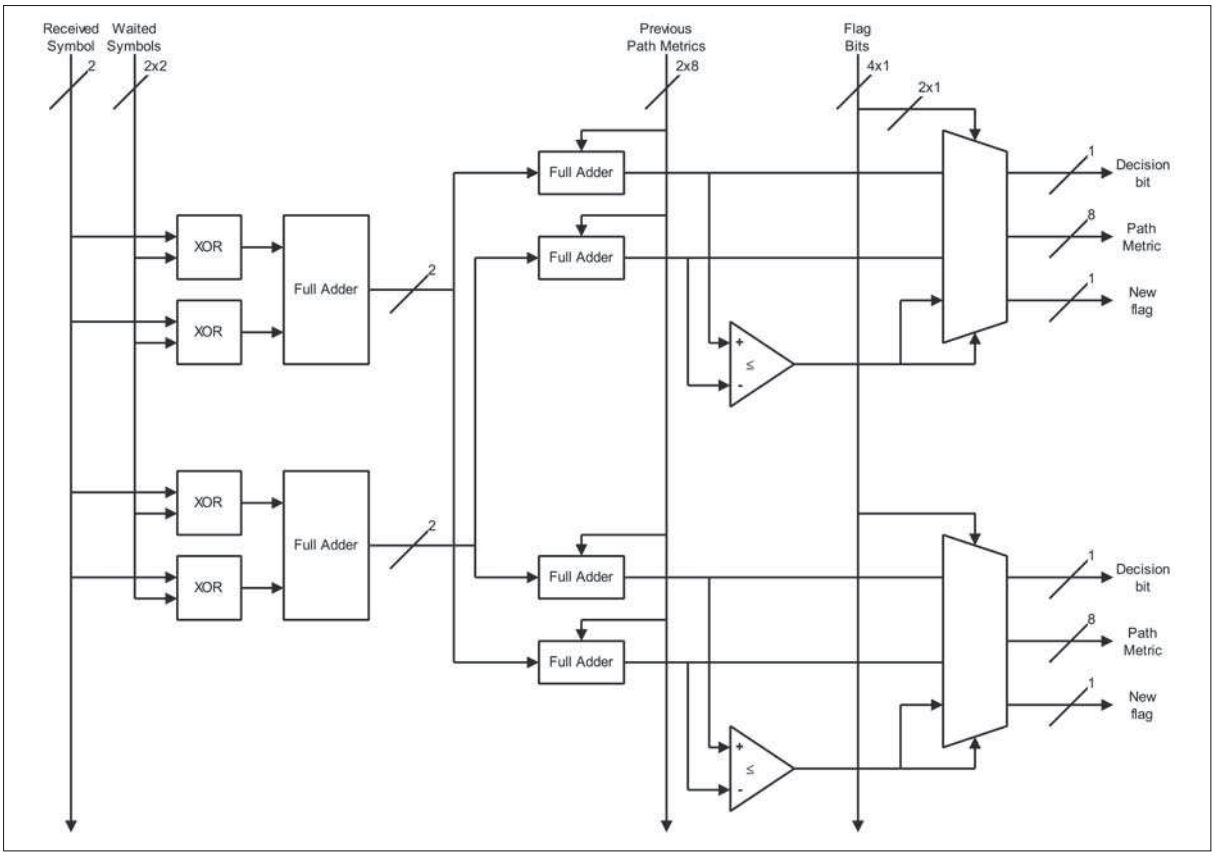

FIGURE 3.6: Unité de calcul des métriques modifiée (illustration pour un seul papillon).

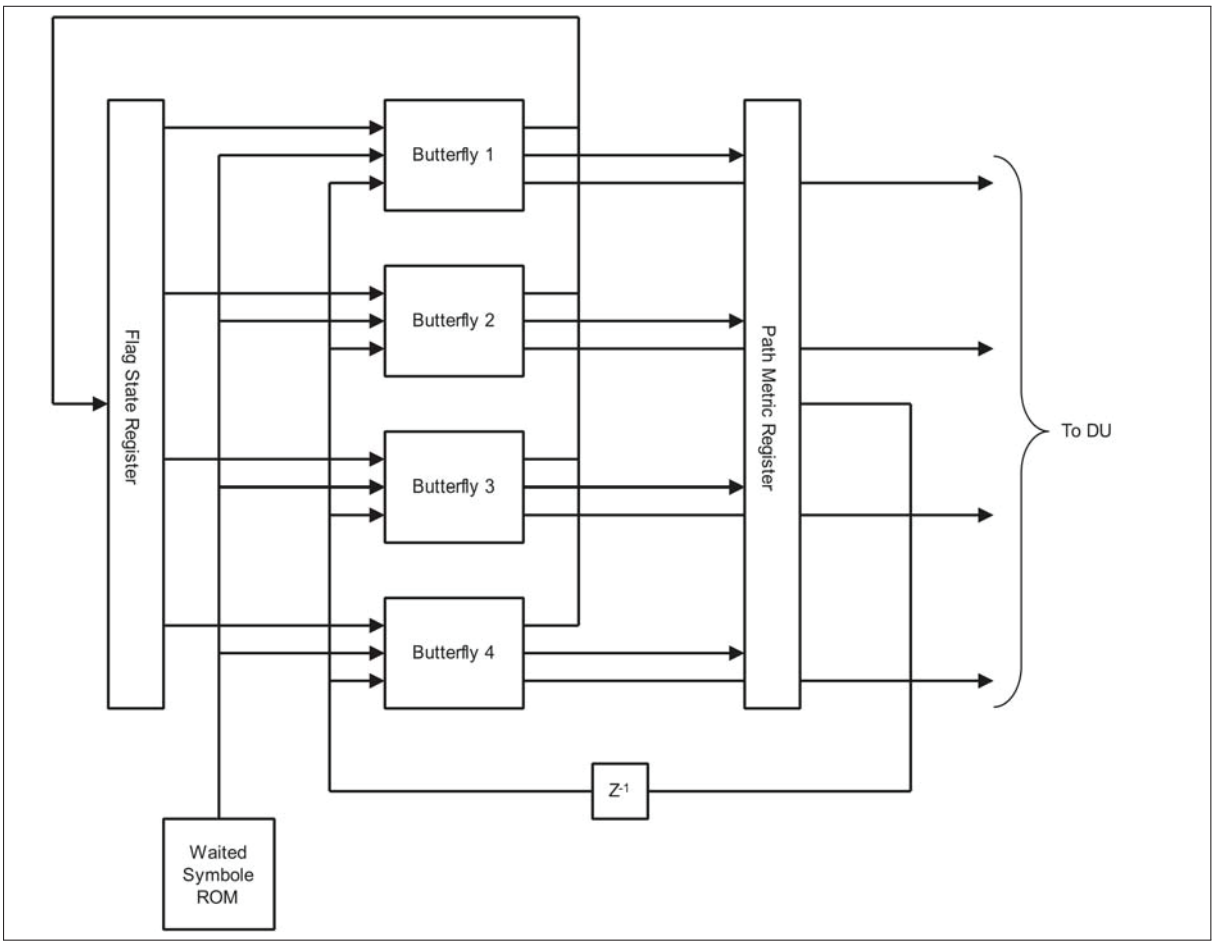

FIGURE 3.7: Architecture finale de la MCU.

3.4 Unité de décision

Cette unité utilise les bits de décision générés par l'unité de calcul des métriques pour décoder la séquence reçue. Il y'a lieu de noter que cette unité ne travaille pas en continu, mais seulement après avoir reçu un nombre de symboles de n bits bien déterminé : C'est la longueur ou la profondeur du *Trace Back* notée L_{TB}. Le décodeur de VITERBI reçoit un flux de symbole continu et il génère un flux de bits de décision qui est aussi continu ; donc on doit diviser le flux des décisions en paquets de longueur $L_{TB} \times N$ qui seront stockés dans la mémoire du TB pour être utilisés une fois le décodage lancé.

3.4.1 Organisation de la mémoire du Trace Back

En ce qui concerne la mémoire du TB, nous ne pouvons utiliser qu'une seule mémoire (RAM) de taille $L_{TB} \times N$ bit, mais un problème se posera sans doute :

Il est clair que le temps du décodage est beaucoup plus grand que le temps nécessaire pour générer une décision : De ce fait, si nous utilisons une seule mémoire, les décisions essentielles, au décodage courant, seront écrasées par les nouvelles décisions des symboles reçus au moment du décodage.

Pour régler ce problème nous avons utilisé deux mémoires de taille $n \times L_{TB}$. La particularité de ces deux mémoires, c'est quand l'une d'elles est en écriture (pour sauvegarder les nouvelles décisions) la deuxième est en lecture (pour servir au décodage courant), sauf pour le premier paquet reçu, la deuxième mémoire sera en mode repos. La Figure 3.8 illustre la mémoire utilisée.

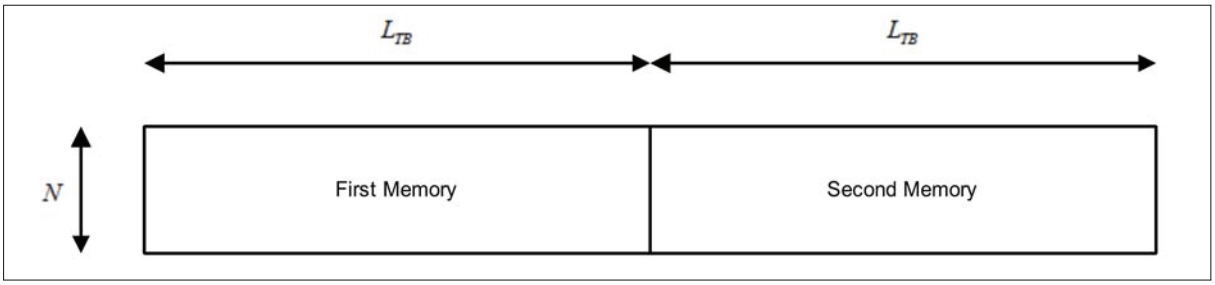

FIGURE 3.8: Mémoire du Trace Back.

Pour bien comprendre le cheminement du flux des décisions dans la mémoire du *Trace Back*, nous décrivons ci-après le contenu de la mémoire pour une longueur de données $3L_{TB}$ en trois étapes :

1. $0 \leq i < L_{TB}$:
   Pour chaque symbole reçu on génère 8 bits de décision (8 états). Ce flux de bits sera stocké dans la première mémoire tandis que la deuxième sera en repos (*Wait*) ; (voir Figure 3.9).

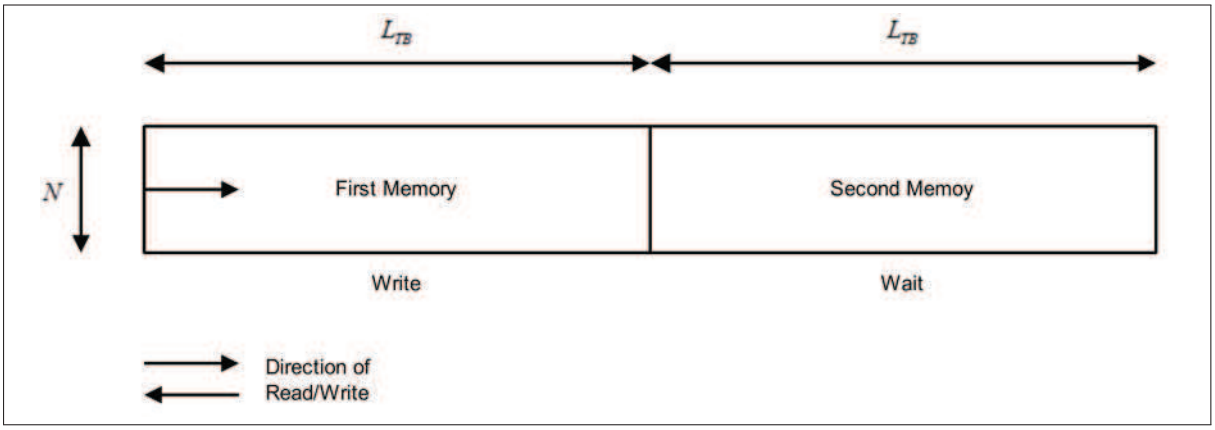

FIGURE 3.9: État de la mémoire du TB pour $0 \leq i < L_{TB}$.

2. $L_{TB} \leq i < 2L_{TB}$:
   La première mémoire est pleine, donc le processus de décodage est lancé. Le décodeur reçoit toujours des symboles et génère des bits de décision qui seront stockés dans la deuxième mémoire. La première mémoire est maintenant en lecture ; (voir Figure 3.10).

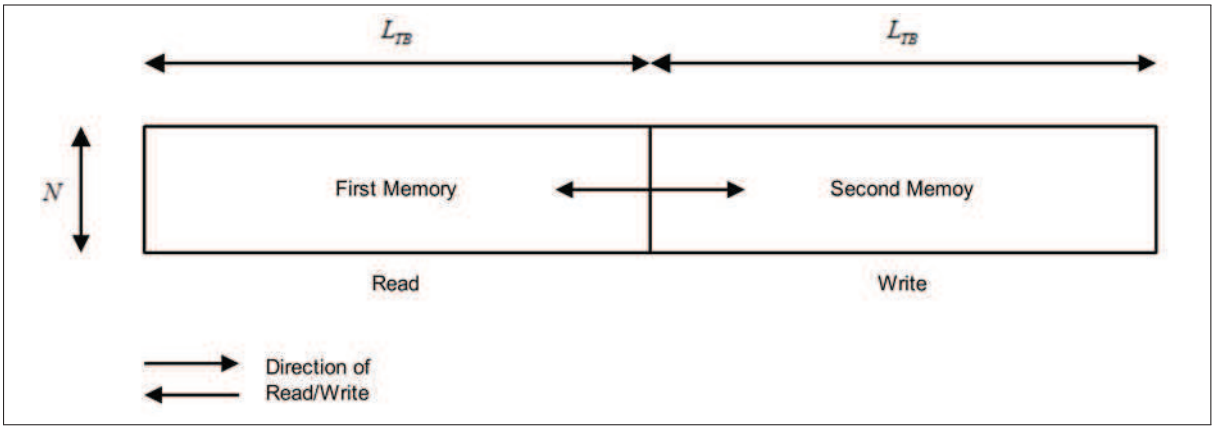

FIGURE 3.10: État de la mémoire du TB pour $L_{TB} \leq i < 2L_{TB}$.

3. $2L_{TB} \leq i < 3L_{TB}$:
   Cette fois c'est la deuxième mémoire qui contient les bits de décision, donc le processus de décodage ; une fois lancé ; utilise cette RAM. La première RAM est en écriture pour sauvegarder les décisions du troisième paquet de données (voir Figure 3.11).

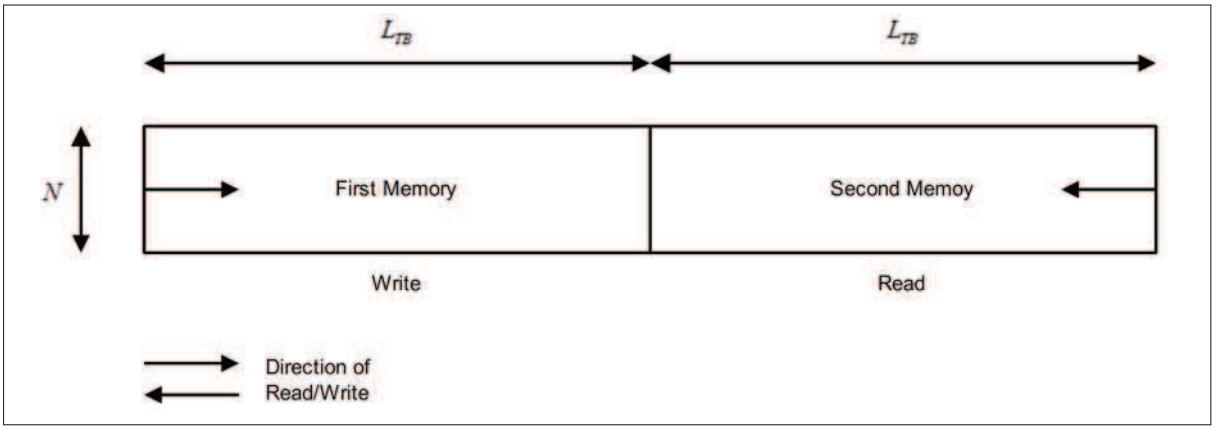

FIGURE 3.11: État de la mémoire du TB pour $2L_{TB} \leq i < 3L_{TB}$.

3.4.2 Fonction de recherche du minimum

Cette fonction a pour but de recherche la position du minimum dans le registre de stockage des métriques de chemin. Puis initialiser le registre du *Trace Back* avec cette position. Pour réaliser cette fonction on applique l'organigramme de la Figure 3.12.

3.4.3 Module *Trace Back*

Cet élément et la pièce maitresse du décodage. Il comporte le registre du *Trace Back* (registre à décalage gauche), et un compteur à décrémentation (de $L_{TB} - 1$ à 0) pour parcourir tous les paquets de décision.

Le registre est initialisé par la fonction de recherche du minimum. En combinant le contenu du registre avec le compteur du *Trace Back* on calcule l'adresse avec laquelle on doit adresser la mémoire du *Trace Back* (celle qui est en mode lecture) pour avoir le bit de décision. Ce bit de décision sera l'entrée du registre TB, et ça sortie c'est le bit de la séquence décodée. A chaque instant on stocke le bit de sortie dans la pile LIFO. Une fois le Trace Back terminé on doit décharger la pile pour avoir la séquence décodée dans l'ordre. La Figure 3.13 est le schéma détaillé du module TB.

3.4.4 Architecture de DU

Après avoir exposer l'architecture de chaque module constituant la DU, nous les assemblerons pour n'en faire qu'une unité : l'Unité de Décision. La Figure 3.14 illustre l'architecture finale de la DU.

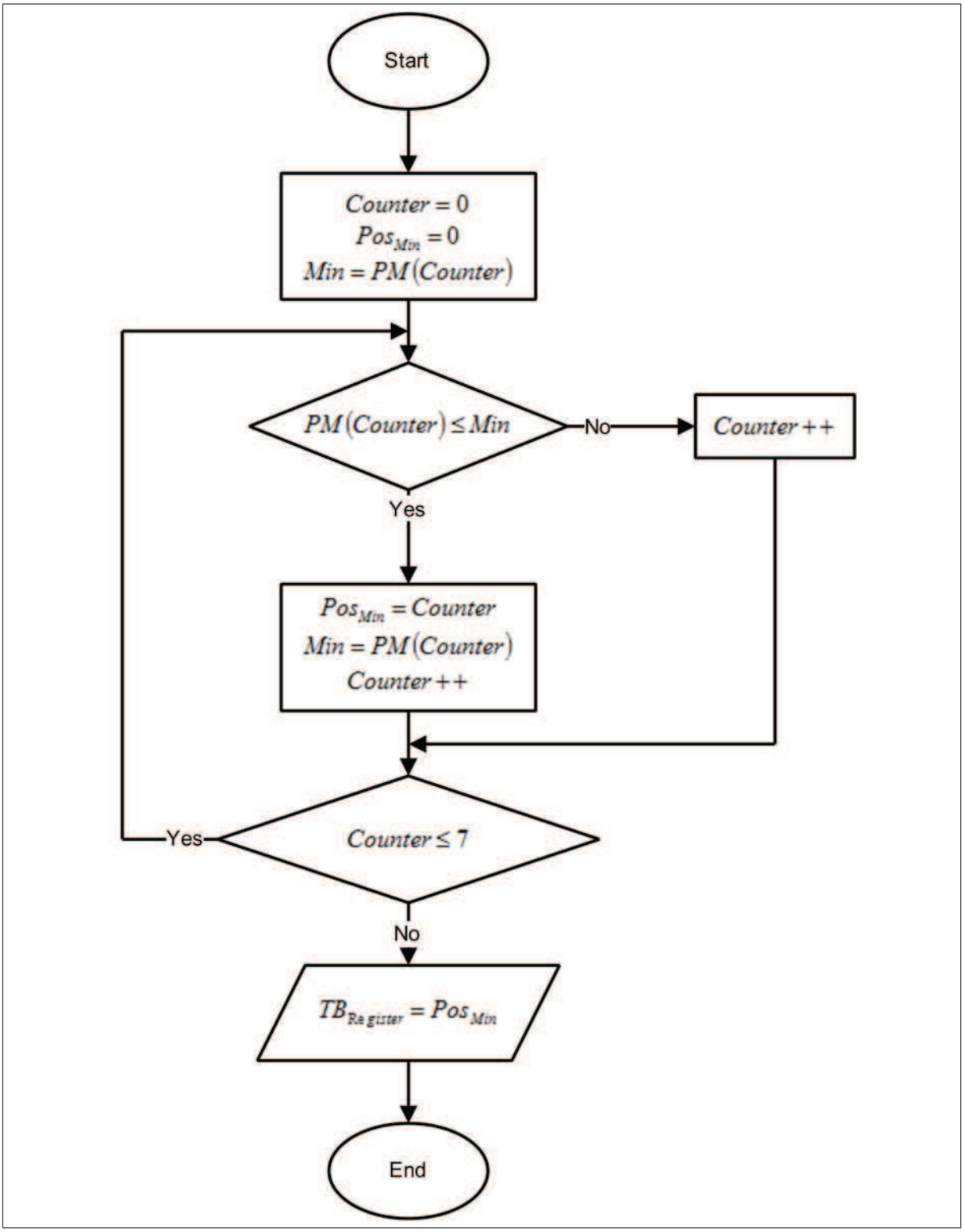

FIGURE 3.12: Organigramme de la fonction de recherche du minimum.

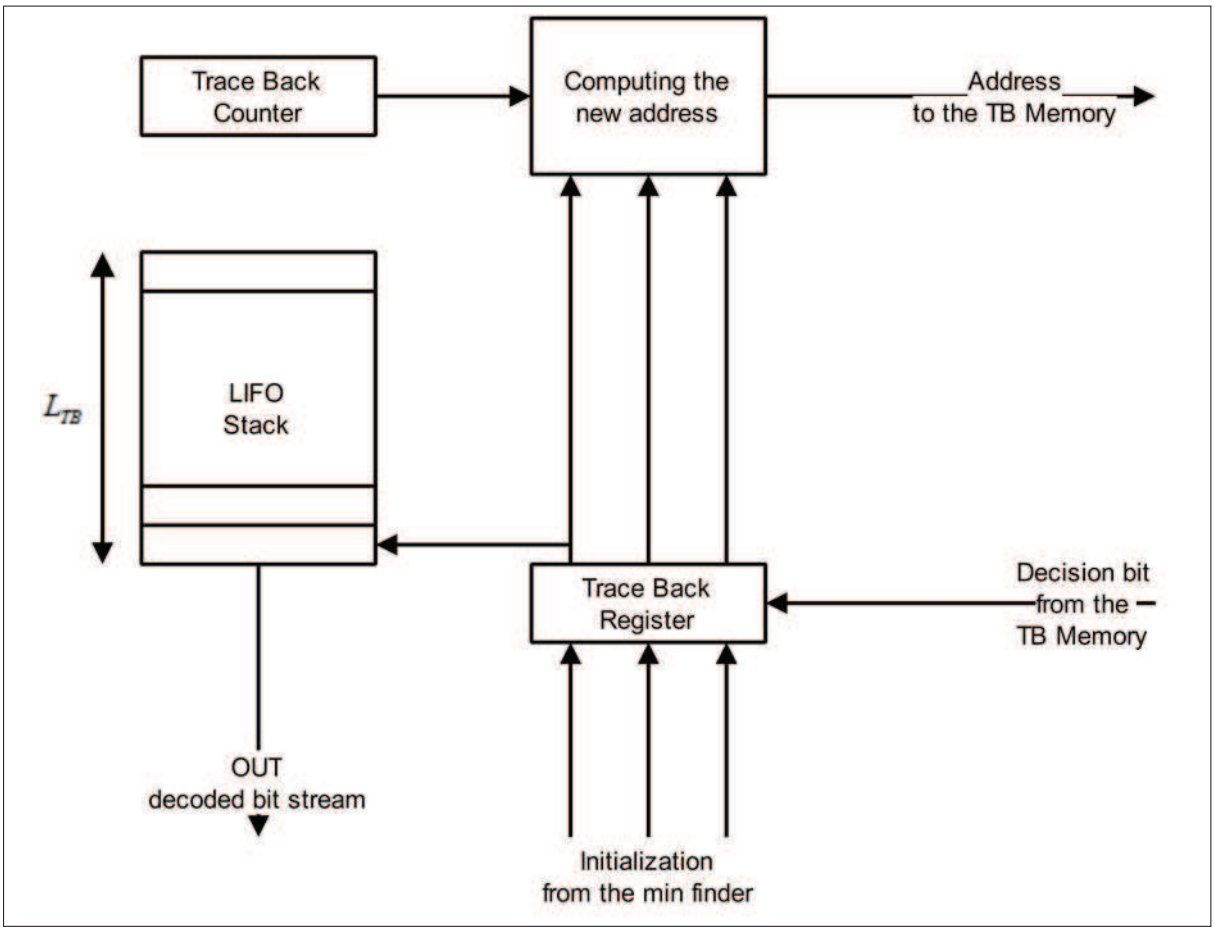

FIGURE 3.13: Schéma détaillé du module TB.

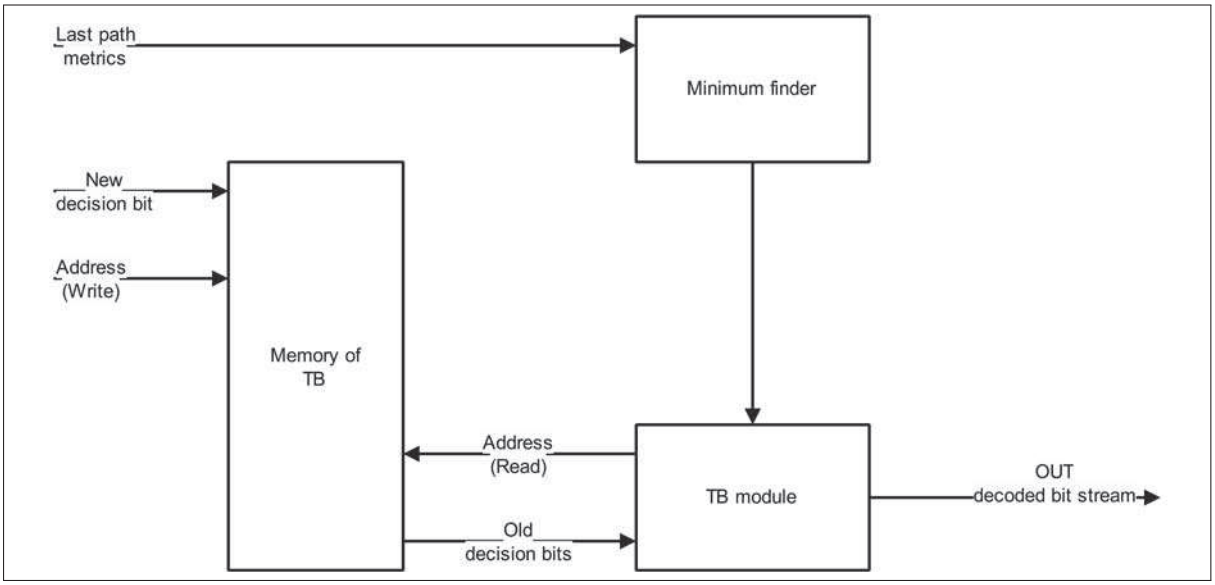

FIGURE 3.14: Architecture finale de la DU.

3.5 Circuit de contrôle

Dans un système logique, l'unité de contrôle ou séquenceur commande et contrôle le fonctionnement d'un système. Une unité de contrôle est un circuit logique séquentiel qui réalise un automate fini, plus précisément une machine de MOORE ou de MEALY, qui génère des signaux de contrôle pour piloter les différents modules de traitement de la donnée.

3.5.1 Synchronisation

Afin de faciliter la synchronisation nous avons décidé de faire transiter toute information d'adressage ou de signaux de contrôle par le circuit principale. Ainsi lorsque les papillons sont prêts à envoyer les métriques de chemin calculées à la RAM des métriques, ils active leur sortie *ready* et cette sortie est considérée par la machine d'état comme une condition de passage à l'étape suivante qui est l'écriture en RAM de ces mêmes métriques de chemin. Une horloge principale permet donc de gérer chacune des horloges internes des différents blocs. Ainsi l'horloge et les Flags donnent la cadence à laquelle se font les opérations.

Des entrée asynchrones permettent de remettre à zéro tous les blocs à n'importe quel moment et ce même si un processus de décodage est en cours.

3.5.2 Chemin de données

C'est aussi le circuit de contrôle qui définie le chemin que suit la donnée dans l'architecture du décodeur. En effet c'est dans cette partie que sont définies les connexions entre différents blocs de traitement.

3.5.3 Le circuit de contrôle du Trace Back

Le processus de Trace Back constitue lui même un circuit de contrôle partiel, synchronisant uniquement les deux blocs suivants :

- Le registre à décalage
- La pile LIFO

Mais au niveau du circuit de contrôle global du décodeur de Viterbi, cette unité ne constitue rien d'autre qu'un composant qui constitue son architecture.

Conclusion et perspectives

L'objectif de ce mémoire est la conception et la mise en œuvre d'un décodeur de VITERBI à décision ferme. Nous avons décrit le principe de fonctionnement de celui-ci, ainsi que l'architecture que nous avons décidé d'adopter.

Nous avons effectué après une étude théorique, dans laquelle nous avons rappelé les principes de base de la communication numérique, de la théorie de l'information, de la théorie des codes correcteurs d'erreur et plus particulièrement, des codeurs Convolutionnels. Nous avons aussi étudié la théorie concernant le décodeur de Viterbi et ses variantes. Nous avons traité la stratégie de trace back et celle du registre exchange.

Nos simulations MATLAB ont confirmé le bon fonctionnement de notre décodeur et de son architecture.

A l'aide de ce logiciel, nous avons pu atteindre nos objectifs de simulation. Nous avons en effet simulé le décodeur de VITERBI à stratégie de trace back mais aussi celui à stratégie registre exchange. De plus nous avons réussi à simuler un décodeur dont les paramètres sont définissables par l'utilisateur. Cette dernière simulation à été appliquées pour plusieurs techniques de communication : CASSINI, CDMA 2000, UWB (802.15).

Les applications du décodeur de VITERBI sont très variées et nécessitent l'adaptation des paramètres de ce dernier afin de pouvoir l'adapter à toutes sortes de communications. Par ailleurs il n'existe pas de lois explicites ou d'études indiquant le choix optimal de ces paramètres. D'où l'idée de simuler sur MATLAB un décodeur paramétrable.

Nous avons écrit des programmes VHDL pour le codeur Convolutionnel choisi, et avons aussi implémenté ce dernier sur carte FPGA SPARTAN 3E. Nous avons aussi écrit le programme VHDL du décodeur de VITERBI suivant l'architecture choisie dans notre projet, et nous avons simulé à l'aide de chronogrammes le fonctionnement des différents blocs constituant le décodeur.

De plus nous avons tenu à construire une architecture (en VHDL), permettant de décoder l'information en continue. En effet le jeu des ram des métriques ainsi que des ram des bits de décision permet d'avoir toujours une ram disponible pour l'unité de calcul des métriques, et une autres ram pour l'unité de trace back. Et cela nous permet de ne pas interrompre ou perdre les données reçues lorsque le décodeur est entrain de remonter le treillis.

A l'avenir, il serait pertinent d'apporter des modifications à la logique que nous avons utilisée, car celle-ci peut être optimisée afin de la rendre plus rapide et moins gourmande

en ressources.

Il serait aussi pertinent de penser à construire un décodeur à décision douce qui utilise la distance euclidienne au lieu de la distance de Hamming définie dans ce mémoire.

Bibliographie

[1] Channel coding for telecommunications. *John Wiley and Sons*, 1999.

[2] Jurgen Freudenberger André Neubauer and Volker Kuhn. Coding theory algorithms architectures and applications. *John Wiley and Sons*, 2007.

[3] G. Battail. Pondération des symboles décodés par lálgorithme de viterbi. *Annales des Télécommunications*, 42, n1-2, pp. 31-38, 1987.

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

[5] E. Angui C. Berrou, P. Adde and S. Faudeuil. A low complexity softoutput viterbi decoder architecture. *Proceedings of IEEE ICC, Geneva*, pp. 737-740, 1993.

[6] R. M. Fano. A heuristic discussion of probabilistic decoding. *IEEE Transaction on Information Theory*, IT-9, pp. 64-74, 1963.

[7] G. D. Forney. The viterbi algorithm. *Proceedings of the IEEE*, 3, 1973.

[8] J. Hagenauer and P. Hoeher. A viterbi algorithm with soft-decision outputs and its applications. *IEEE Global Communications Conference, Globecom, Dallas, Texas*, pp. 1680-1686, 1989.

[9] R. Johannesson and K. Sh. Zigangirov. Fundamentals of convolutional coding. *IEEE Press.*

[10] F. Jelinek L. R. Bahl, J. Cocke and J. Raviv. Optimal decoding of linear codes for minimizing symbol error rate. *IEEE Transactions on Information Theory*, IT-20, pp. 284-287, 1974.

[11] J. G. Proakis. Communication systems engineering. *Prentice Hall*, 2nd edition, 2002.

[12] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transaction on Information Theory*, IT-13, pp. 260-269, 1967.