RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

École Nationale Polytechnique

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

Département d'Électronique
Master

présenté pour obtenir le titre de MASTER en
Électronique

Implémentation sur Circuit Reconfigurable d'un décodeur de Viterbi à Décision Ferme : Register Exchange

Abderrezak HADJI

Soutenu publiquement le 25/06/2012 devant un jury composé de :

Président	M. C.Larbes	Professeur	(ENP)
Examinateur	M. S.Ait Cheikh	Maitre de Conférence A	(ENP)
Encadreur	M. M.Taghi	Chargé de Cours	(ENP)
Co-encadreur	M. L.Abdelouel	Chargé de Cours	(ENP)

Master réalisé au Laboratoire des Dispositifs de Communications
et de Conversions Photovoltaïques
Équipe 03 : Commande des dispositifs.
École Nationale Polytechnique
10 avenue Hacène Badi – El Harrach
BP182–16200 Alger
Algérie

Tél : (+213) 21 52 53 01/03
Fax : (+213) 21 52 29 73
Web : http ://www.enp.edu.dz/

ملخص

يتمحور هذا العمل حول محاكاة ثم تثبيت خوارزميه فكّ الرموز فيتربي (Viterbi) ذو القرار الصعب على بطاقة FPGA. المفكك يستعمل داخل أجهزة استقبال أنظمة الاتصالات الرقمية لكشف و تصحيح الأخطاء الواردة عندما تكون المعلومة مشفرة بمشفر ملفف (Codeur Convolutionnel) قمنا بدراسة مرمزة التفافيّة .

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.

الكلمات المفتاحية : فيتربي، الاتصالات الرقمية، **مشفر ملفف.**

Résumé

Ce travail s'articule autour de la simulation et l'implémentation sur FPGA d'un décodeur de Viterbi à décision ferme.
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 réalisé un programme VHDL dont nous avons exposé l'avancement et les résultats.

Mots clé : Viterbi, Code convolutionels, Codage de canal.

Abstract

Our project aims to simulate and 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. A convolutionnal coder of constraint length $\mathrm{L} = 4$ has been studied. MATLAB simulations were performed for this same length, but also for nondescript length. At last, a VHDL program has been written, which we have outlined the progress and results.

Keys words: Viterbi, Convolutionnal encoder, Channel coding.

Remerciements

Je remercie mes parents qui m'ont beaucoup encouragé le long de ce projet. Et à l'issu de mes études faites au sein de l'École Nationale Polytechnique je voudrais rendre un hommage tout particulier à : Mes promoteurs M. M.Taghi et M. L.Abdelouel qui m'ont honoré en acceptant de m'encadrer. Tous les enseignants du département d'Électronique de l'École Nationale Polytechnique. Tous les enseignants du département des Sciences Fondamentales. Je remercie tous ceux qui ont participé de prés ou de loin à l'aboutissement de nos études.

Dédicace

À mes chers Parents,

Vous avez guidé mes premiers pas et veillé sur moi depuis ma plus tendre enfance, tout en m'indiquant le savoir et les bonnes manières.

Si aujourd'hui je suis devenu ce que je suis, c'est bien grâce à vous. C'est pourquoi je vous dédie humblement ce travail qui n'est autre que le fruit de vos sacrifices.

Maman, Papa je vous dois tous !

À la mémoire

de mes Grands Parents paternels,
de ma Tante et de mon Oncle paternels ;

À mes Grands Parents maternels ;

À mes Tantes et mes Oncles maternels ;

À mes Cousines et mes Cousins paternels et maternels ;

À ma Sœur,

À mes deux Frères ;

À mes amis en tout particulièrement Inès, Hayet, Marwa, Amir, Ridha, Hichem, Hakim, Abdellatif, Noredine, Houssem, Khaled, Zaki...

« Donnez-moi un point fixe et un levier, et je soulèverai la terre ».

Archimède

Abderrezak HADJI

Table des matières

Table des figures

Liste des tableaux

Liste des symboles

ACS Add Compare Select

ACSU Add Compare Select Unit

BMU Branch Metric Unit

CDMA Code Division Multiple Access

DU Decision Unit

DVB Digital Video Broadcasting

MCU Metrics Calculating Unit

PMU Path Metric Unit

RAM Random Access Memory

RE Register Exchange

REU Register Exchange Unit

ROM Read Only Memory

SISO Soft Input Soft Output

SOVA Soft-Output Viterbi Algorithm

TB Trace Back

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 à pouvoir 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 1955 par Elias[2], 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 avec approche de décodage : *Register Exchange*. 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 deux chapitre. Dans le premier 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 deuxième 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).

Chapitre 1

Décodeur de Viterbi

1.1 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 801.11 Wireless (WiFi 801.11). Le tableau 1.1 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.

1.2 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 (801.15)	$1/3$	7	$g_1 = 171_8$ $g_2 = 165_8$ $g_3 = 133_8$

TABLE 1.1: 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].

1.2.1 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] :

$$\begin{aligned} \log \Pr(b \mid r) &= \log \left[p^z (1-p)^{n-z} \right] && (1.1) \\ &= n \log(1-p) - z \log \left(\frac{1-p}{p} \right) && (1.2) \end{aligned}$$

ou bien :

$$\log \Pr(b \mid r) = -A - Bz \tag{1.3}$$

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 :

$$BM\left(a_p^{(i-1)},\ a_q^{(i)}\right)_{p \text{ prend } 2^k \text{ valeurs différentes}} \tag{1.4}$$

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 :

$$PM\left(a_q^{(i)}\right) = \min\left\{PM\left(a_p^{(i-1)}\right) + BM\left(a_p^{(i-1)},\ a_q^{(i)}\right)\right\}_{p \text{ prend } 2^k \text{ valeurs différentes}} \tag{1.5}$$

$PM\left(a_q^{(i)}\right)$: 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 (1.5) est appelé ACS, illustrée dans la Figure 1.1 pour un codeur de rendement $R = {}^{1}/_{n}$.

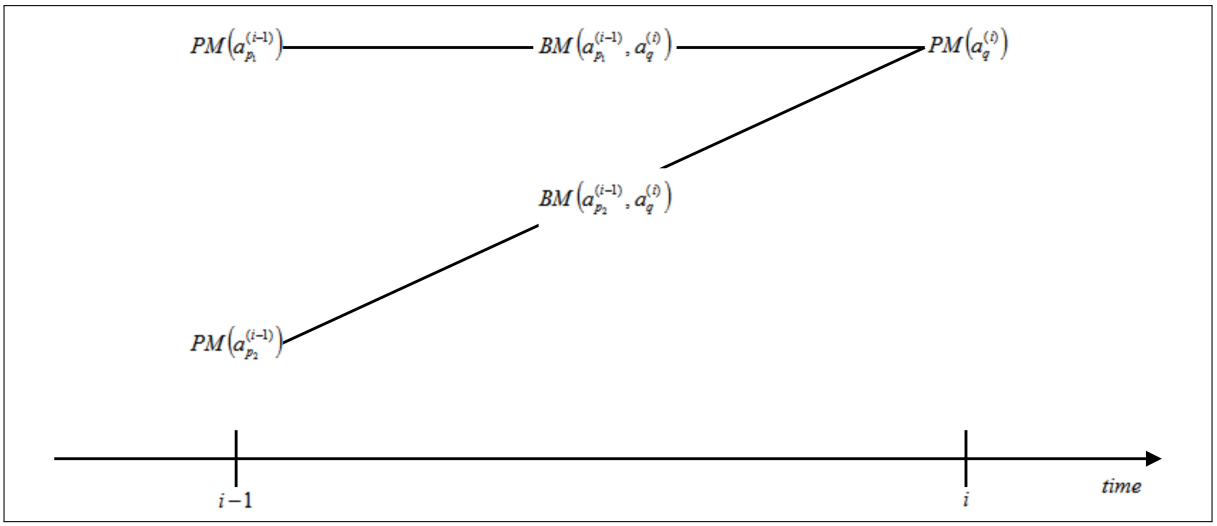

FIGURE 1.1: Illustration de l'opération ACS Taux $R = {}^{1}/_{n}$.

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.

1.2.2 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 1.2 à 1.7 illustrent le treillis à chaque instant.

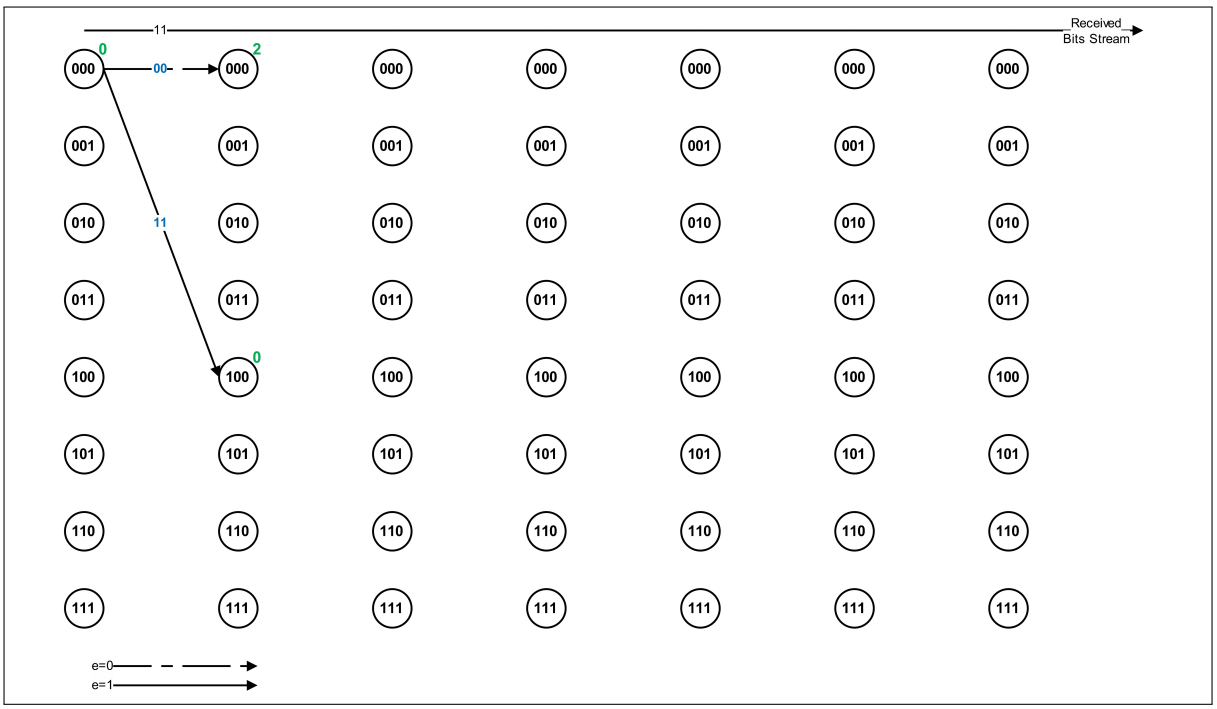

FIGURE 1.2: État du treillis à l'instant $i = 1$.

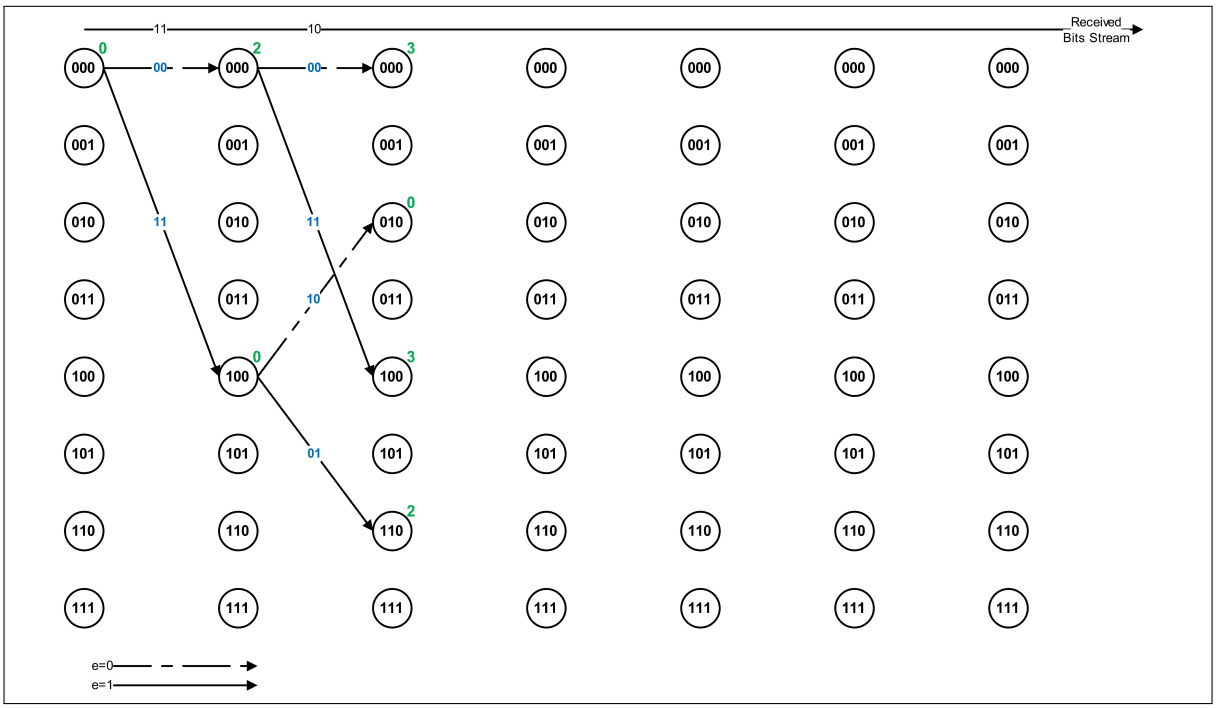

FIGURE 1.3: État du treillis à l'instant $i = 2$.

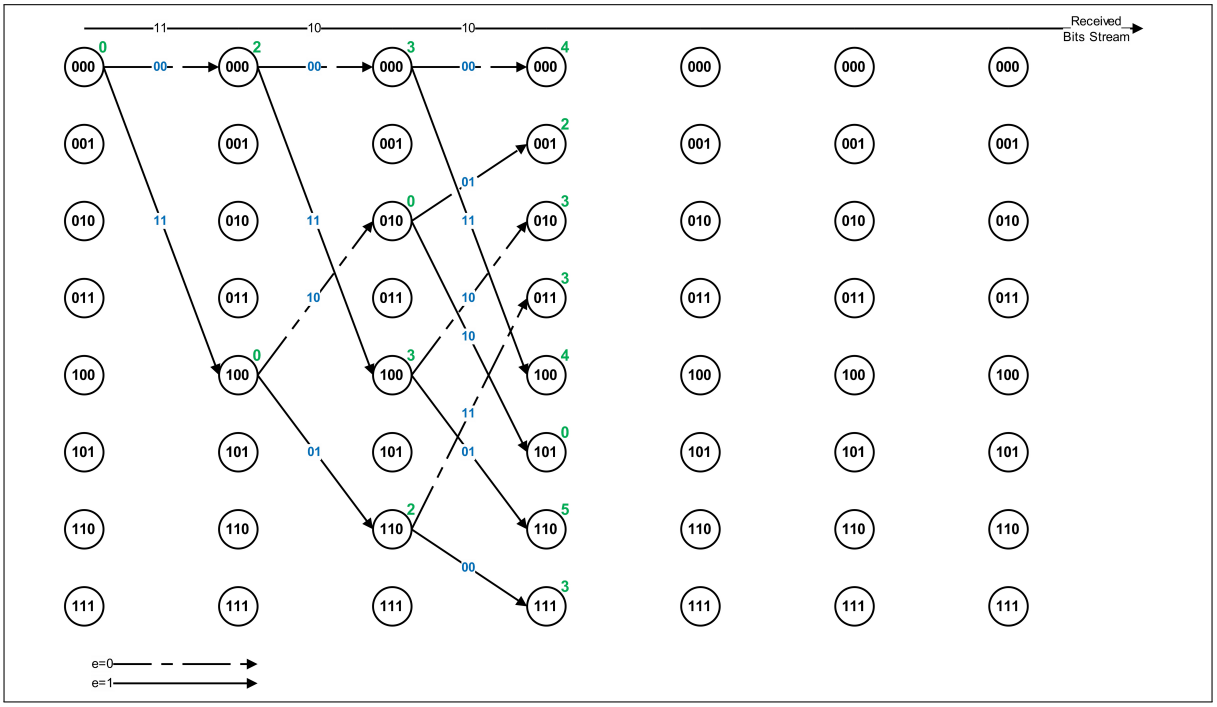

FIGURE 1.4: État du treillis à l'instant $i = 3$.

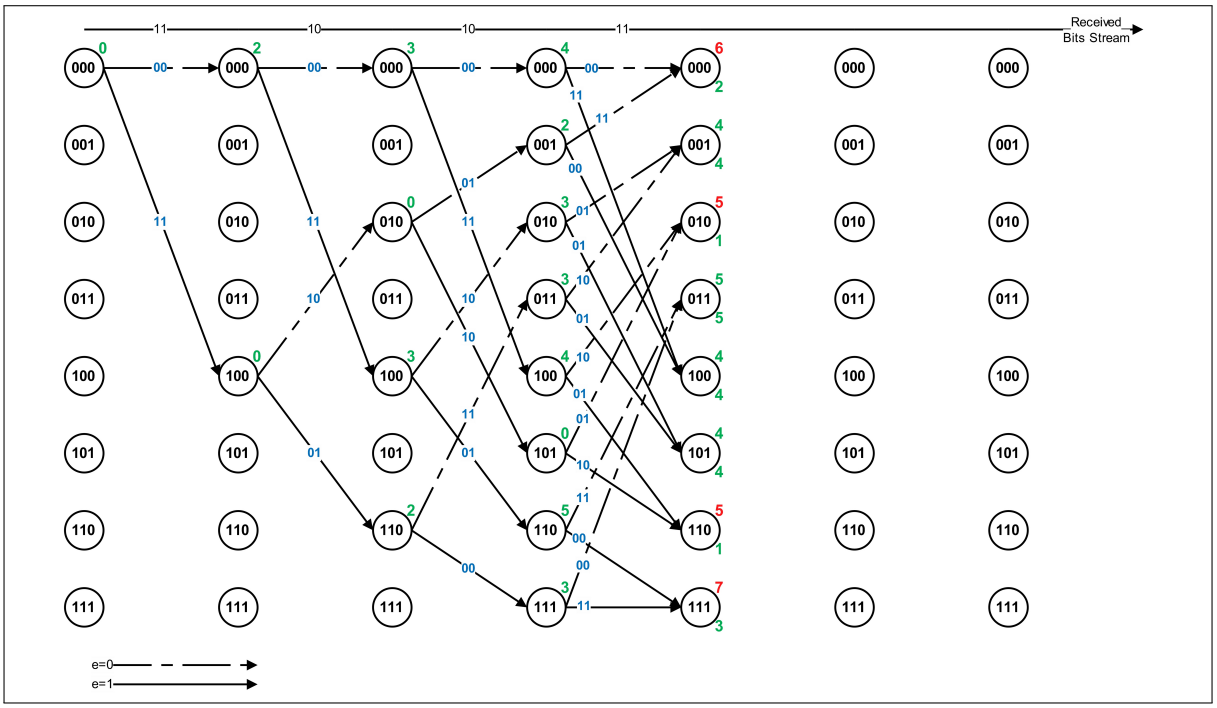

FIGURE 1.5: État du treillis à l'instant $i = 4$.

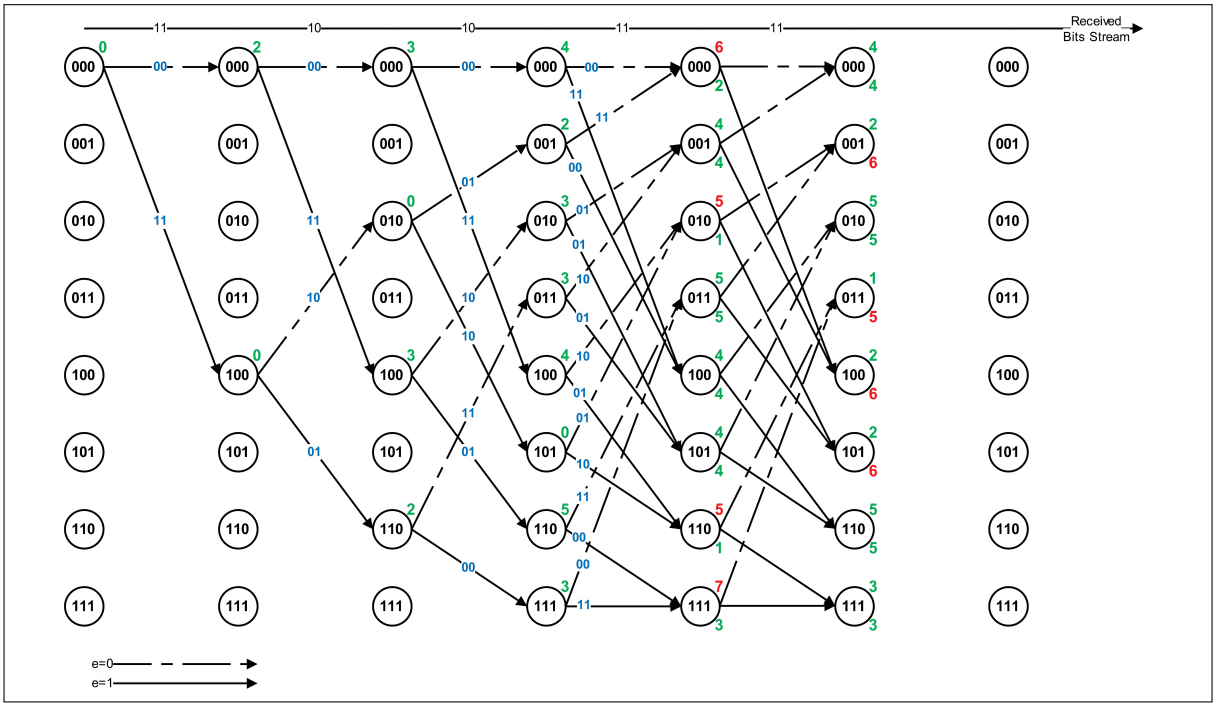

FIGURE 1.6: État du treillis à l'instant $i = 5$.

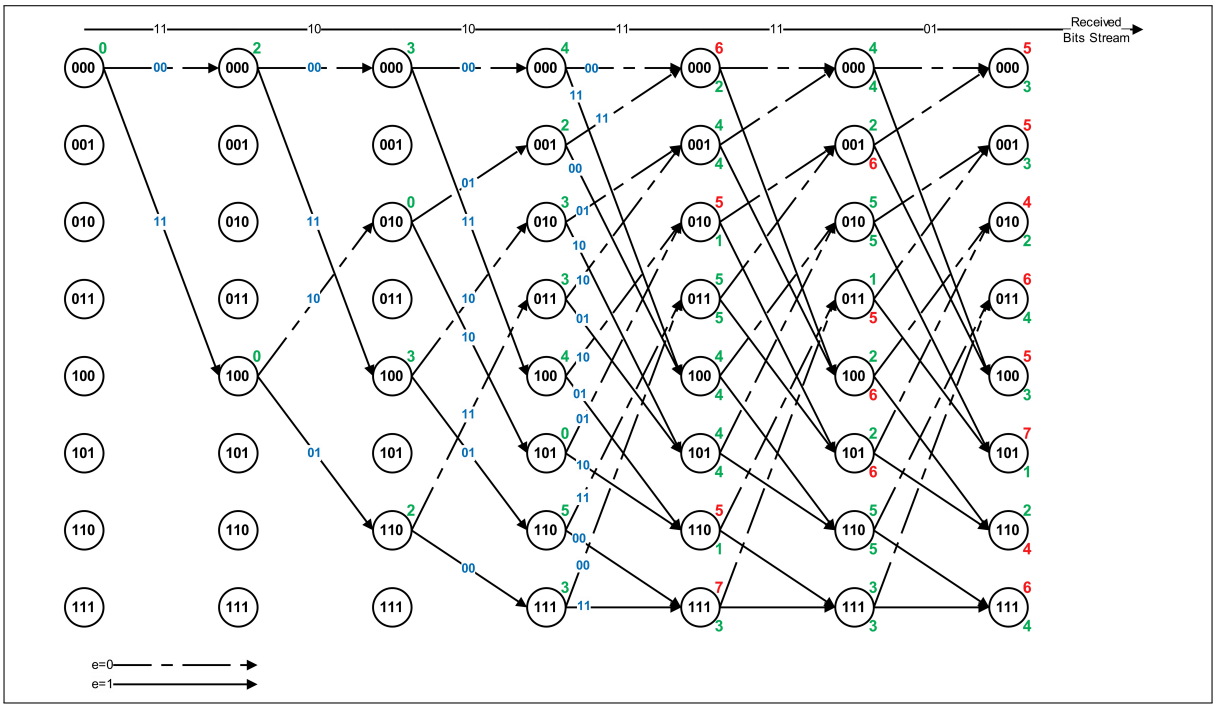

FIGURE 1.7: État du treillis à l'instant $i = 6$.

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 1.8 illustre cette opération.

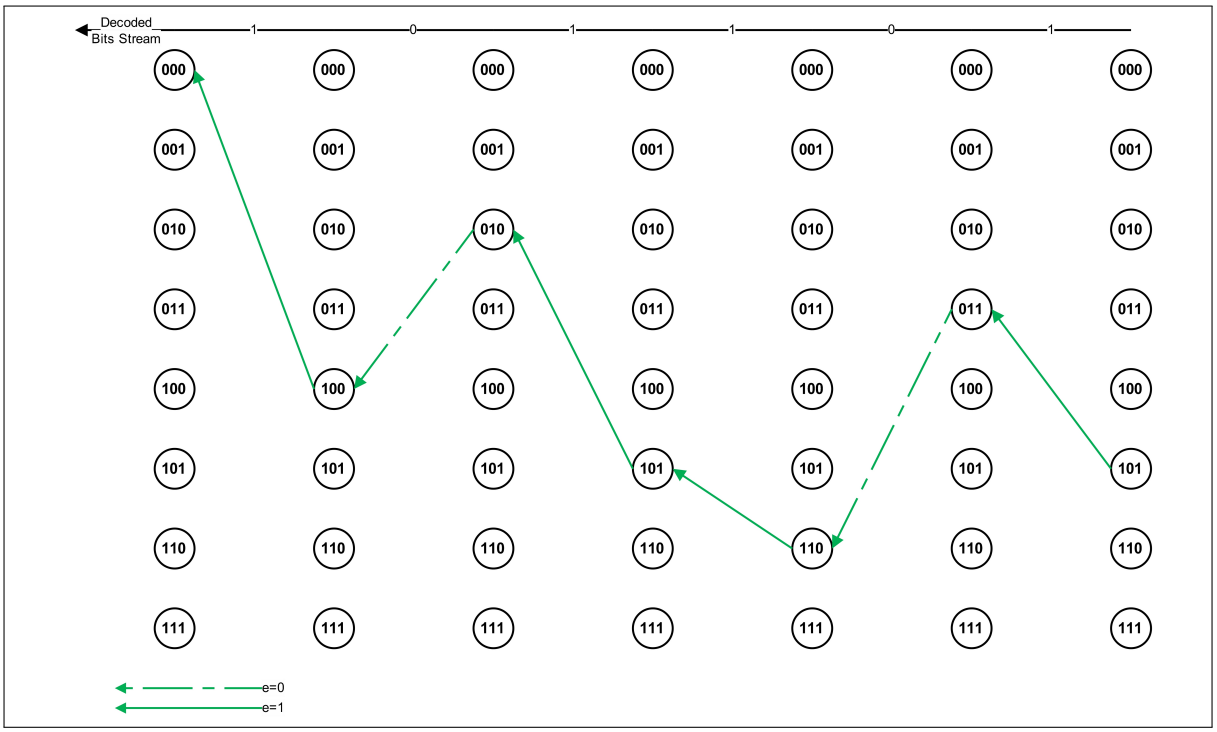

FIGURE 1.8: Décodage en remontant le treillis.

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.

1.3 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 1.9.

D'après la Figure 1.9 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

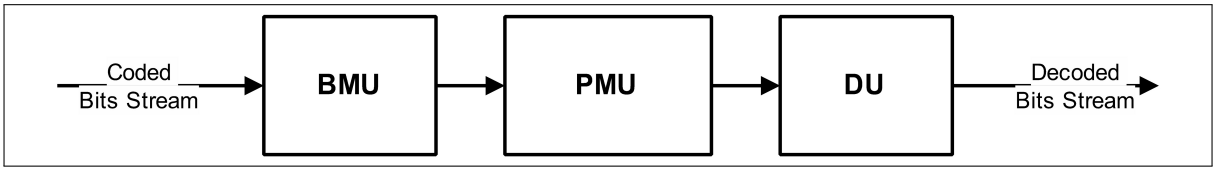

FIGURE 1.9: Illustration des différents modules du VD.

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.

1.3.1 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 1.10 illustre le schéma fonctionnel de la BMU pour $n = 2$.

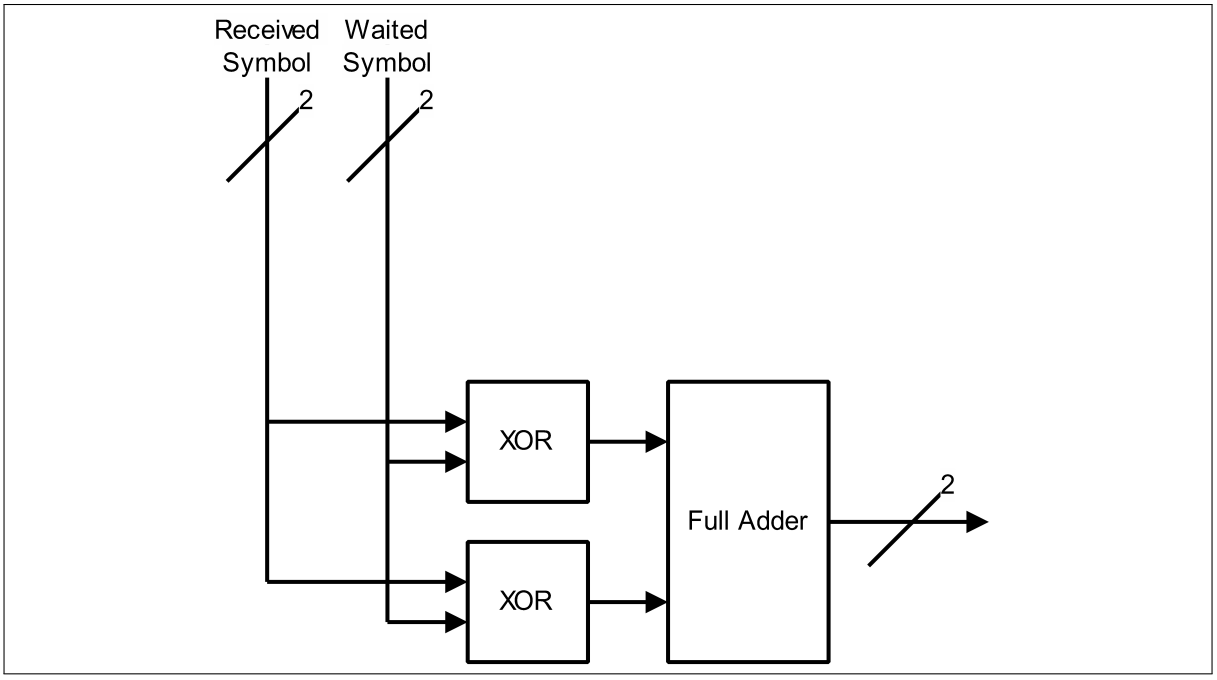

FIGURE 1.10: Illustration de la BMU.

1.3.2 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, aussi ces bits permettent d'échanger les information entre les Registres qui contiennent les bits décodés. 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 1.11 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 1.12 illustre le regroupement des papillons pour un treillis de $N = 2^3 = 8$, et un $R = 1/2$.

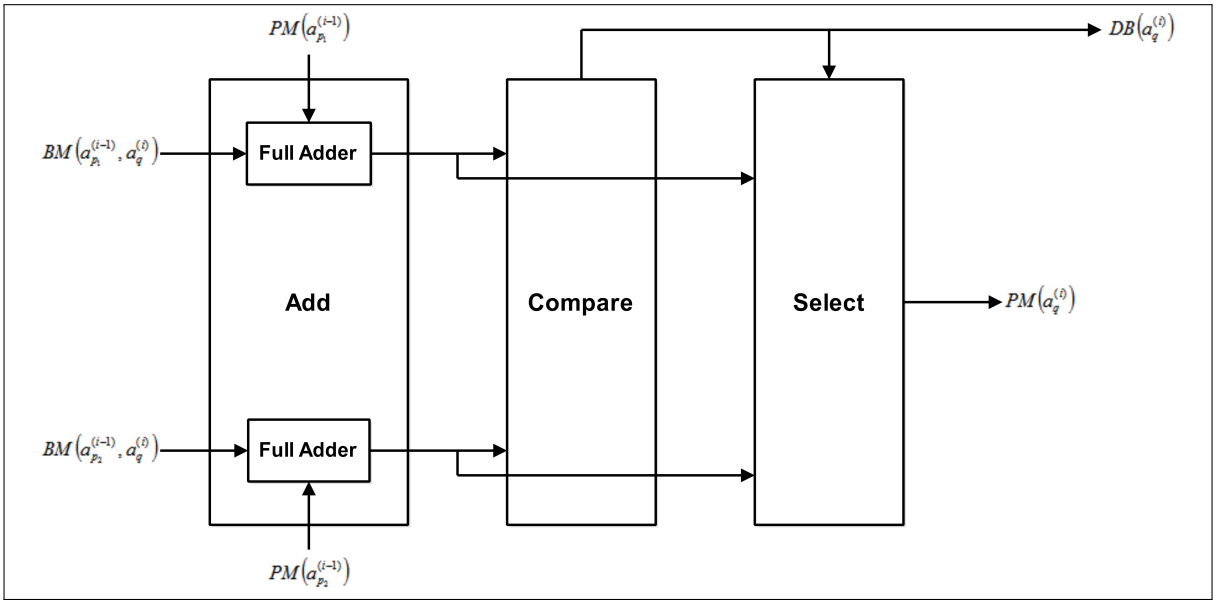

FIGURE 1.11: Schéma de l'ACS pour $R = 1/n$.

1.3.3 Unité de Décision

Dans cette unité on génère la séquence décodée. Dans notre travail, pour cette unité on a choisie l'approche *Register Exchange*. Cette méthode est la méthode la plus simple pour le décodage, puisque le décodage est inclus dans la structure du treillis. En effet chaque état possède un Registre ou il stocke les bits décodés à chaque instant i. La séquence estimée se trouve dans le registre qui correspond à l'état qui possède la métrique de chemin la plus petite. Donc pour réaliser la DU (ou REU) on doit implémenter une fonction qui recherche la position du minimum et donne en sortie l'état correspondant, et un autre module qui prend en entrée la position du minimum, et délivre en sortie une séquence de bits, c'est la séquence décodée. La Figure 1.13 illustre le schéma bloc de cette unité.

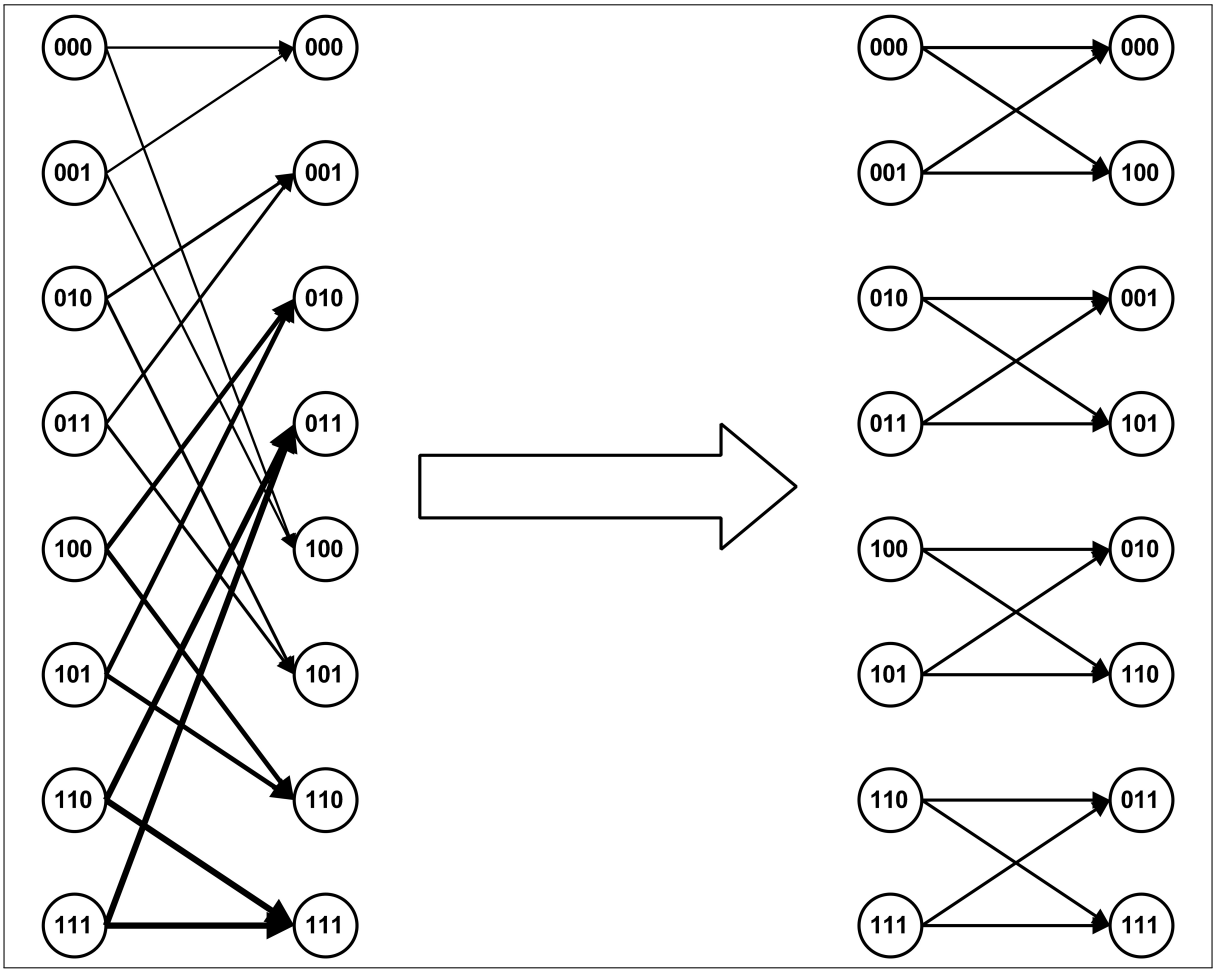

FIGURE 1.12: Regroupement des papillons pour un treillis $N = 8$ et $R = 1/2$.

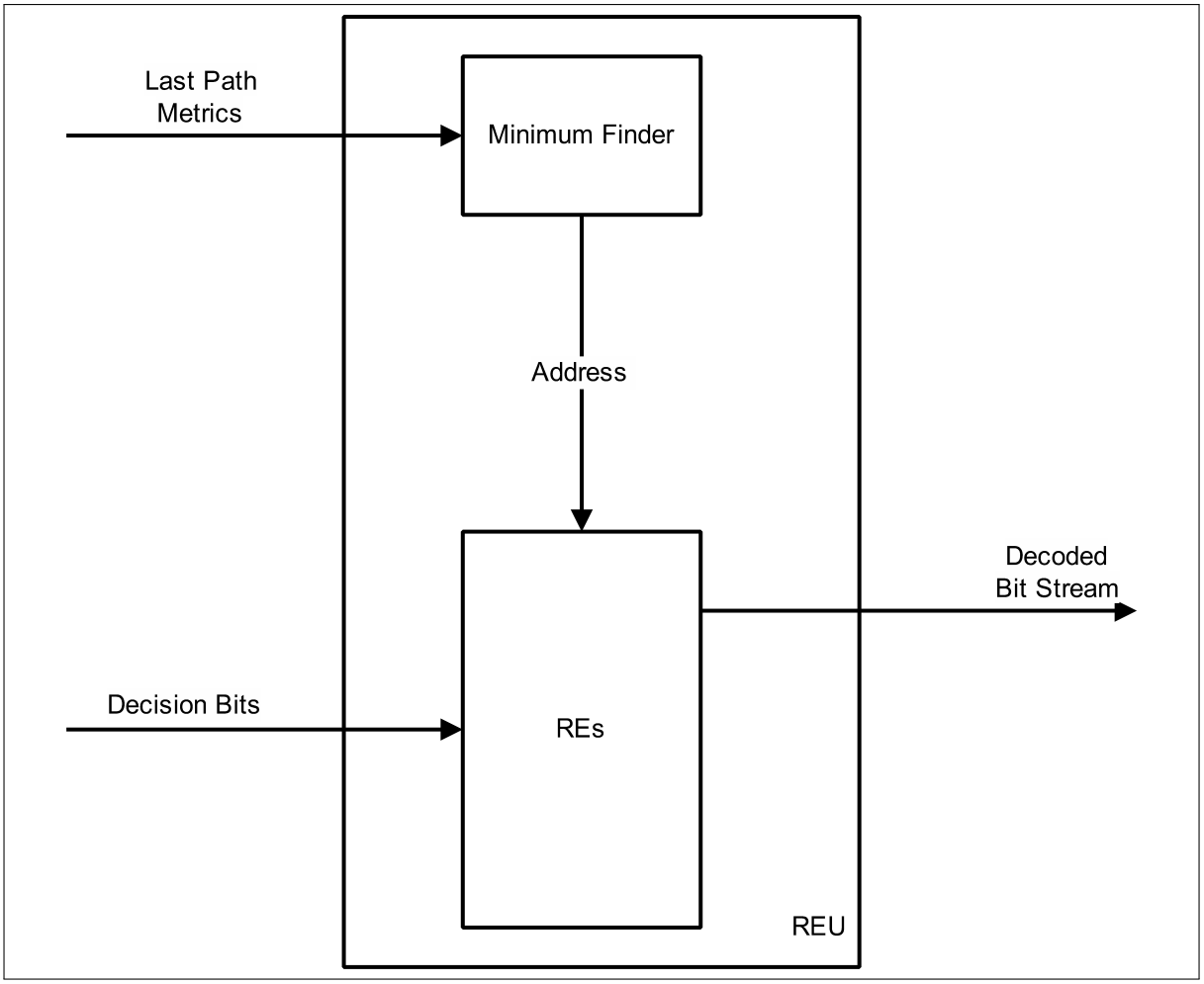

FIGURE 1.13: Illustration de la DU.

Chapitre 2

Architecture proposée

2.1 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 $g_2 = 13_8$.

2.2 Schéma proposé du décodeur

L'architecture en bloc du décodeur de VITERBI est souvent celle montrée à la Figure 1.9. 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 2.1 illustre le schéma en bloc de notre décodeur.

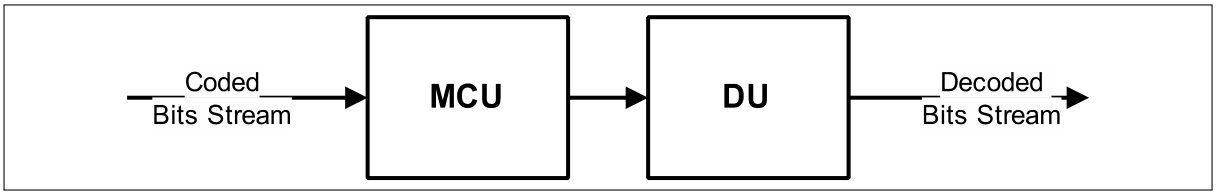

FIGURE 2.1: Schéma bloc du décodeur.

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.

2.3 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.

2.3.1 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 :

$$NPM_1\left(a_q^{(i)}\right) = PM\left(a_{p_1}^{(i-1)}\right) + BM\left(a_{p_1}^{(i-1)},\ a_q^{(i)}\right) \tag{2.1}$$

$$NPM_2\left(a_q^{(i)}\right) = PM\left(a_{p_2}^{(i-1)}\right) + BM\left(a_{p_2}^{(i-1)},\ a_q^{(i)}\right) \tag{2.2}$$

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 2.2 est l'organigramme appliqué par l'ACS pour effectuer les trois opérations. Le schéma détaillé de l'ACS est illustré dans la Figure 2.3.

2.3.2 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 2.4 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

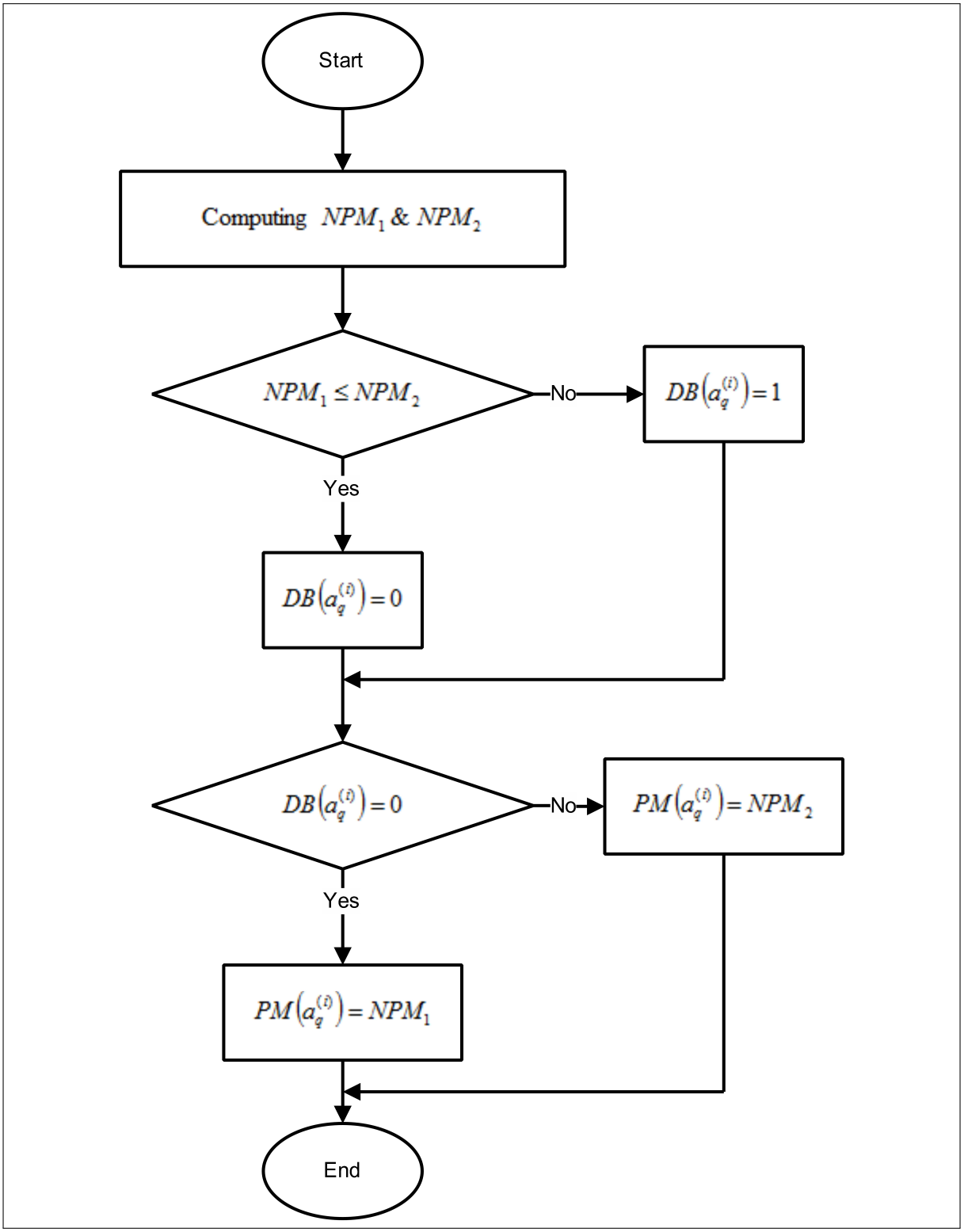

FIGURE 2.2: Organigramme appliqué par l'ACS.

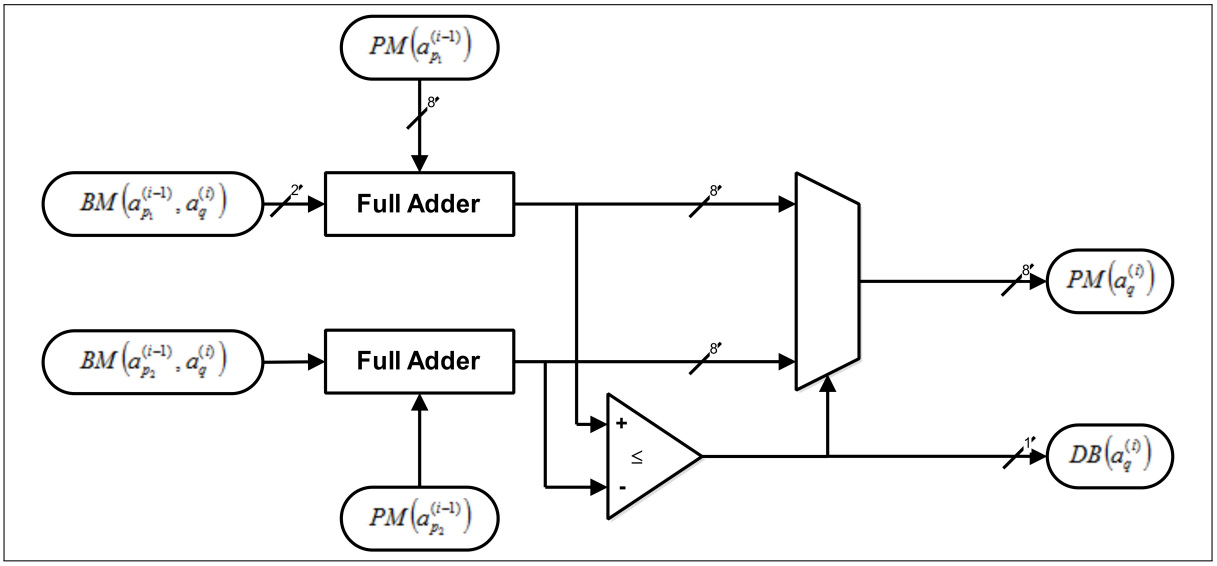

FIGURE 2.3: Schéma détaillé de l'ACS.

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 2.5 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.

2.3.3 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 2.5 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 2.1 résume toutes

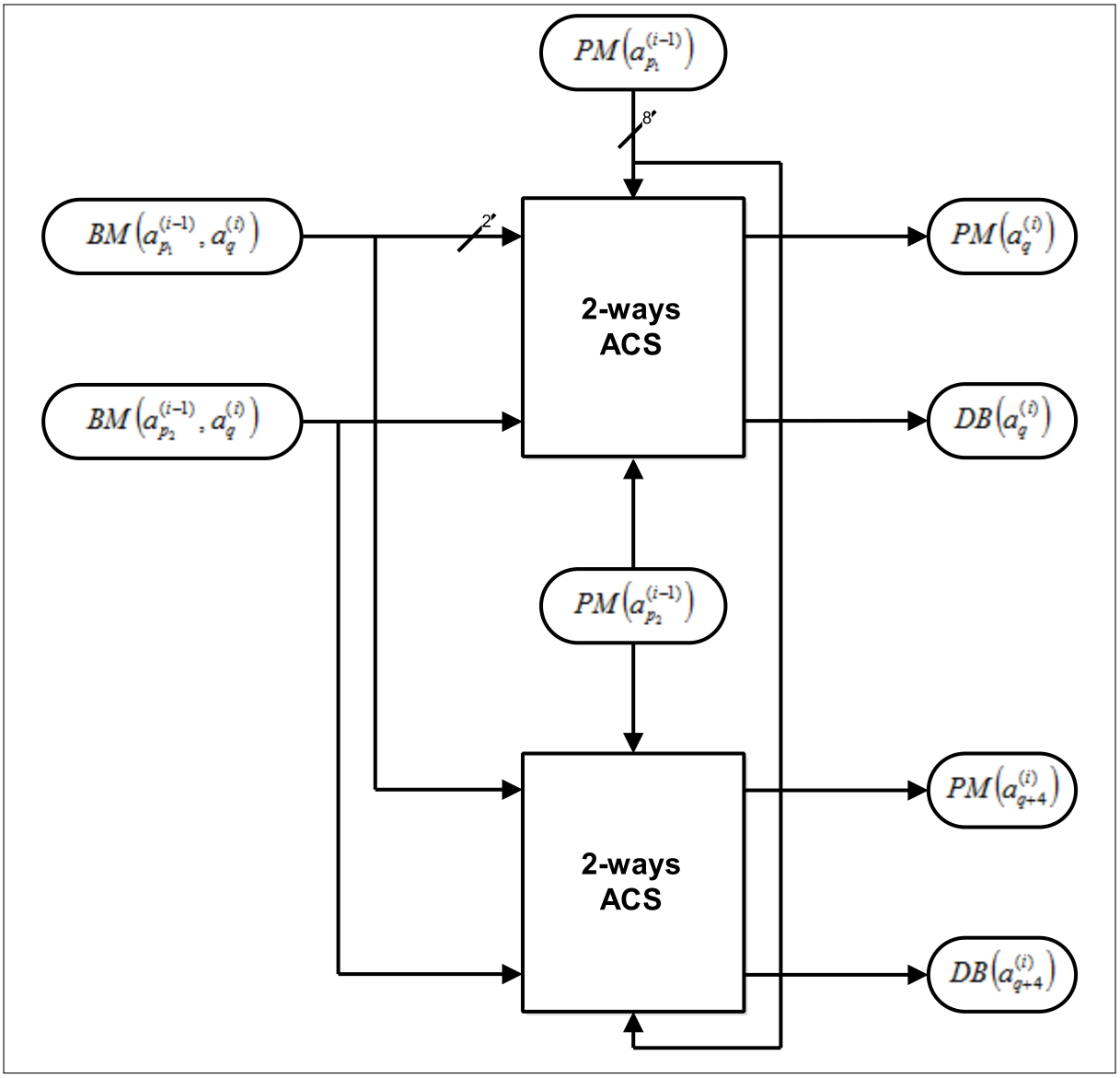

FIGURE 2.4: Architecture d'un papillon.

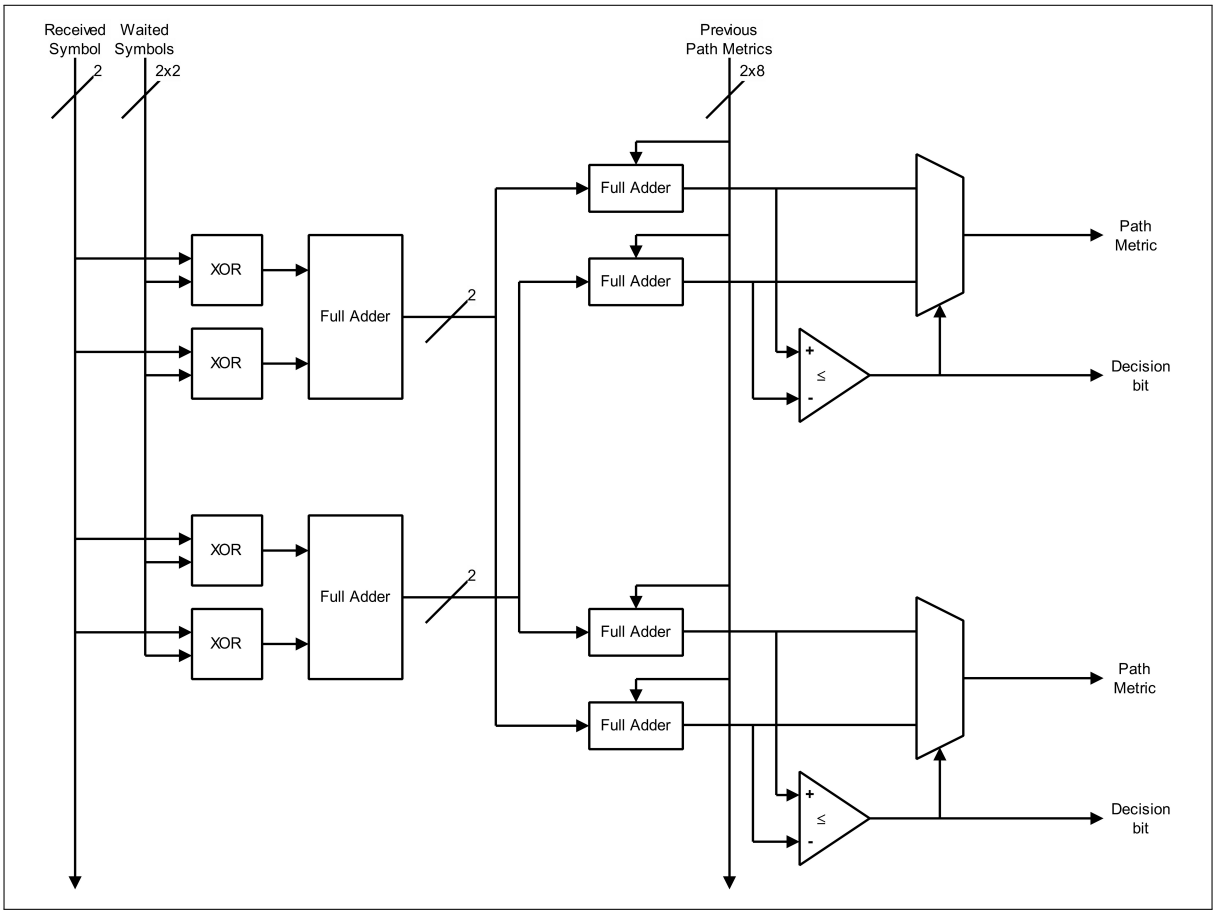

FIGURE 2.5: Architecture détaillée du papillon.

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 2.1: 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 ;

$$NPM_1\left(a_q^{(i)}\right) = PM\left(a_{p_1}^{(i-1)}\right) + BM\left(a_{p_1}^{(i-1)},\ a_q^{(i)}\right) \tag{2.3}$$

NPM_2 : New Path Metric 2 ;

$$NPM_2\left(a_q^{(i)}\right) = PM\left(a_{p_2}^{(i-1)}\right) + BM\left(a_{p_2}^{(i-1)},\ a_q^{(i)}\right) \tag{2.4}$$

$DB\left(a_q^{(i)}\right)$: 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 2.6 illustre toutes les modifications apportées à l'unité de calcul des métriques pour un seul éléments ACS.

2.3.4 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 2.7.

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.

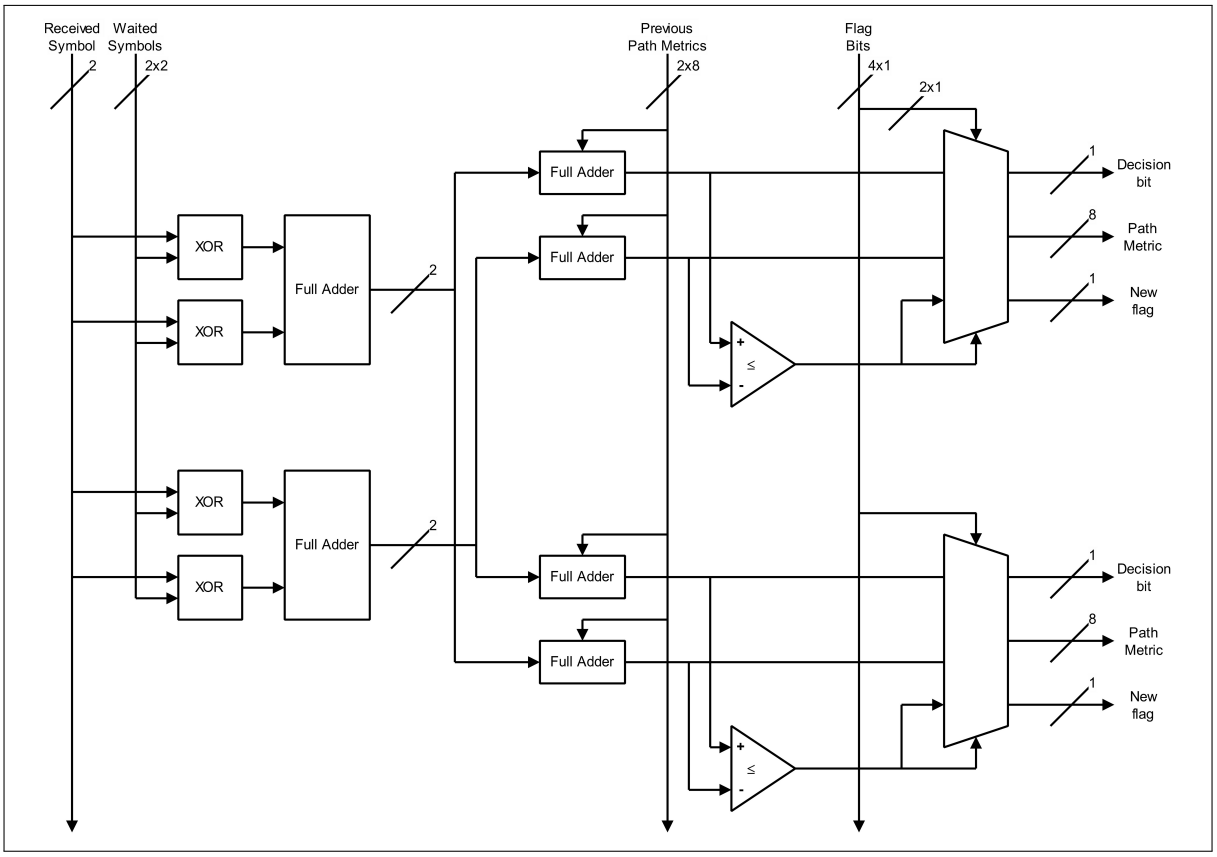

FIGURE 2.6: Unité de calcul des métriques modifiée (illustration pour un seul papillon).

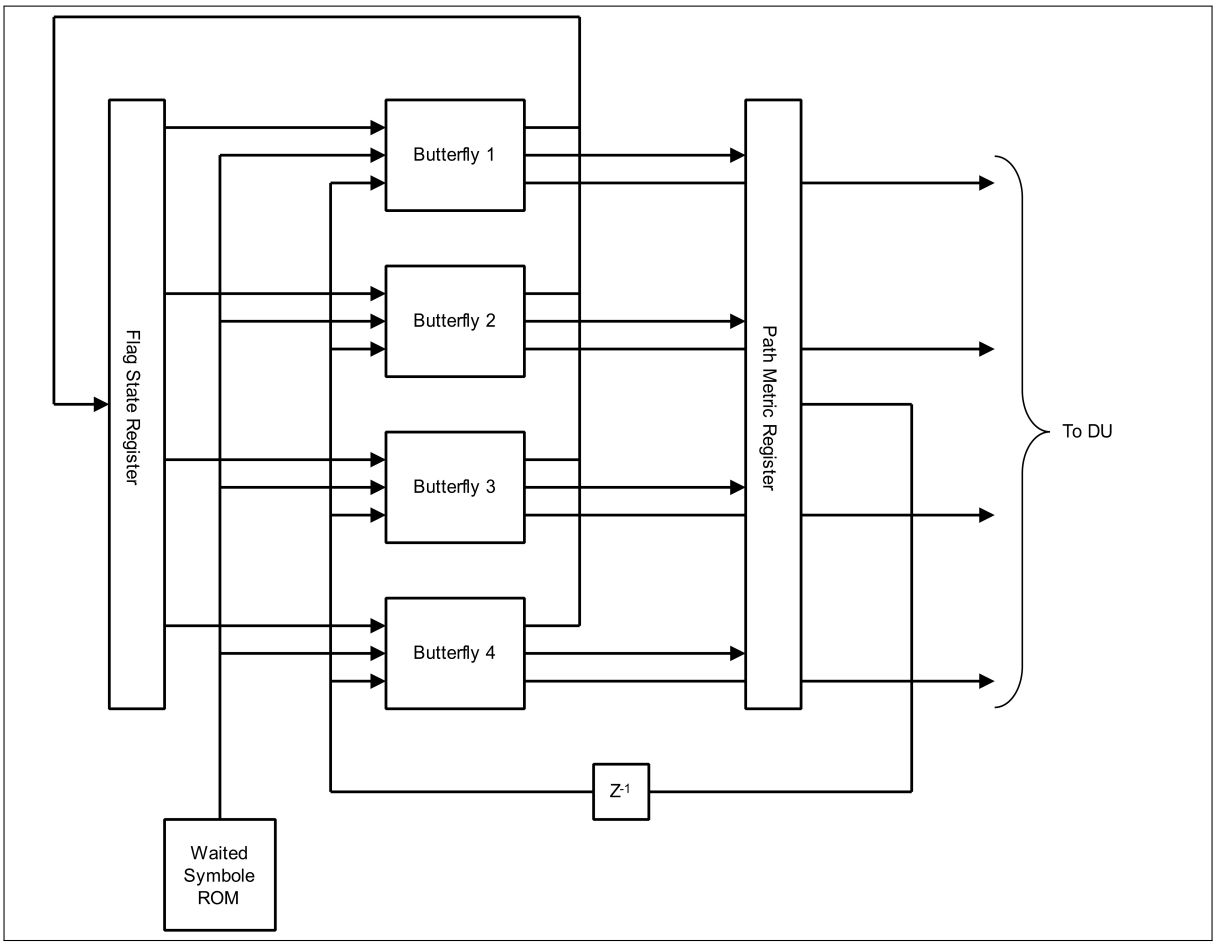

FIGURE 2.7: Architecture finale de la MCU.

2.4 Unité de décision

Cette unité utilise les bits de décision générés par l'unité de calcul des métriques pour décoder la séquence reçue.

2.4.1 Organisation des REs

Dans notre cas on possède de 8 Registre qui contiennent les bits décodés. L'échange des bits décodés entre les différents registres se fait grâce au bits de décisions et à l'aide de multiplexeur (Voir Figure 2.8).

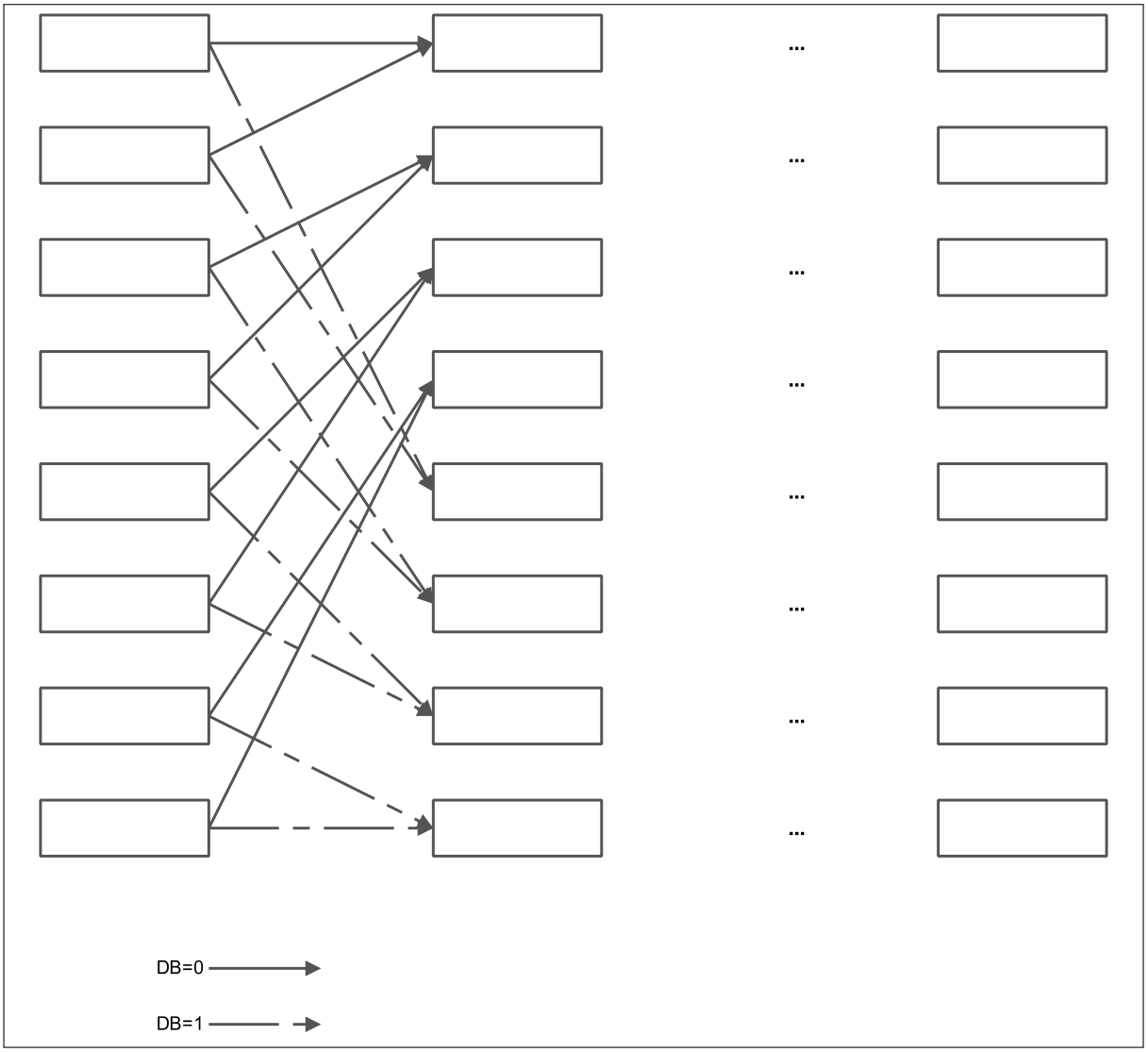

FIGURE 2.8: Échange des bits décodés entre les REs.

2.4.2 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 2.9.

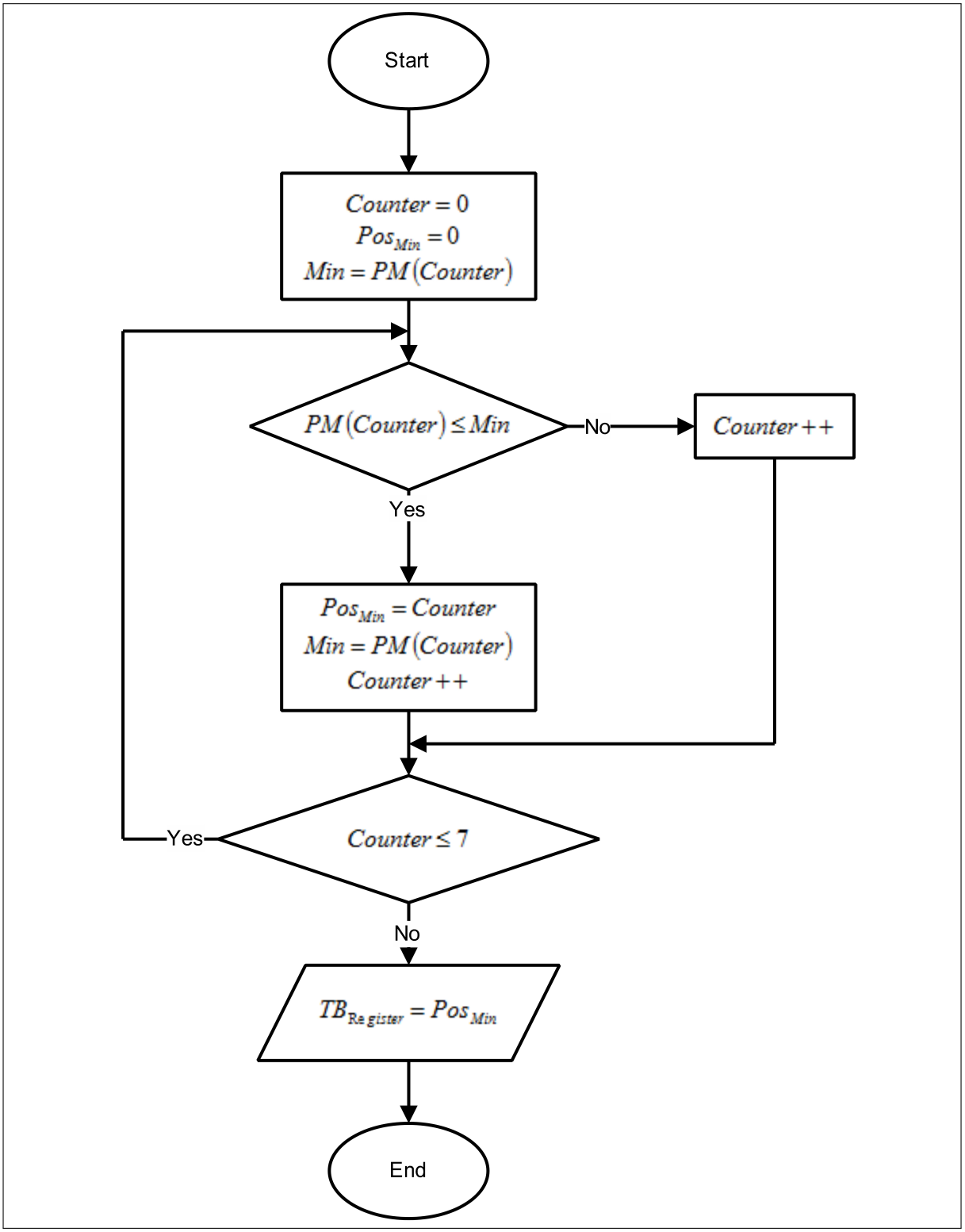

FIGURE 2.9: Organigramme de la fonction de recherche du minimum.

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.