N° d'ordre : XXX

Thèse

présentée à

TELECOM Bretagne

en habilitation conjointe avec l'Université de Bretagne-Sud

pour obtenir le grade de

Docteur de TELECOM Bretagne

Mention : *Sciences pour l'Ingénieur*

par

Haisheng Liu

Contributions à la maîtrise de la consommation dans des turbo-décodeurs

---

soutenue le 1 juillet 2009 devant la commission d'examen :

Composition du Jury :

Président :	? ? ?
Directeur de thèse :	? ? ?
Rapporteur :	? ? ?
Examinateur :	? ? ?
Invité :	

Table des matières

Liste des figures

Liste des tableaux

Introduction

Théorie de l'information et turbocodes

Au cours du XX$^{\text{ème}}$ siècle, de nombreuses innovations technologiques ont changé radicalement les sociétés et leur manière d'interagir. Les technologies de communication sont en perpétuelle évolution et la quantité d'information à transmettre devient de plus en plus importante. Pour répondre à ces exigences, la communauté scientifique tente d'établir des systèmes de communication toujours plus performants et innovants. Ainsi, des innovations technologiques sont proposées pour augmenter la quantité d'information transmise, la fiabilité des systèmes et diminuer leur consommation. Cependant, les fondements de la transmission d'information reste identique et les travaux de recherche sont basés sur des modèles établis.

En 1948, la théorie de l'information a été établie par Claude Shannon [Shannon 48] sous la forme d'une modélisation mathématique. A la même époque, l'invention du transistor a rendu possible la transmission numérique en utilisant l'état du composant semi-conducteur (*ouvert* et *fermé*) pour transmettre les informations « 0 » et « 1 ». Cette invention est à l'origine de l'avènement de l'ère de la transmission numérique. Sans donner le type de transmission, le théorème de Shannon indique d'une part la quantité de ressources nécessaires pour transmettre un message et d'autre part la quantité d'information qu'il est possible de transmettre. Depuis lors, la question « *comment atteindre ces capacités théoriques ?* » constitue un véritable défi pour les chercheurs.

La recherche de codes correcteurs d'erreurs est l'une des voies explorées. Dans les années 1950, les théoriciens ont proposé les premières familles de codes correcteurs d'erreurs à savoir les codes BCH, de Reed-Solomon, de Reed-Muller et les codes convolutifs [Elias 54]. Le décodage des codes BCH posa néanmoins un problème de la mise en œuvre. En effet, l'application du critère de maximum de vraisemblance pour ce type de code implique une croissance exponentielle du nombre du calcul. Malgré les algorithmes développés par Berlekamp [Berlekamp 84] et Massey [Massey 69] qui favorise la diminution de la complexité calculatoire, leur réalisation n'était pas possible avec les technologies de l'époque. Quant aux codes convolutifs, l'algorithme de Viterbi proposé en 1967 [Viterbi 67] est devenu l'algorithme de référence pour le décodage. Cet algorithme est basé sur le critère du maximum de vraisemblance *a posteriori*. Puis, le principe des codes concaténés est apparu pour diminuer l'encombrement du décodeur à la réception. Mais il fallut attendre Claude Berrou en 1993 [Berrou 93] pour aboutir à l'invention des turbocodes, qui atteignent des performances à 0.7dB de la limite théorique de Shannon. Cette famille de code combine à la fois une concaténation parallèle et un décodage itératif. L'invention des turbocodes a permis de franchir une étape importante et d'aboutir à des codes correcteurs d'erreurs très performants. Le principe turbo fut étendu aux codes en blocs par Ramesh Pyndiah [Pyndiah 98]. De plus, les

caractéristiques des turbocodes ont favorisé la redécouverte des codes LDPC (Low Density Parity Code) par MacKay [Gallager 62] [MacKay 99]. Par ailleurs, il faut aussi mentionner les innovations technologiques qui ont permis de progressivement intégrer des dizaines puis des centaines de millions de transistors sur une puce de silicium. Grâce à ces innovations, il est maintenant possible d'intégrer des algorithmes de décodage toujours plus complexes dans un circuit intégré. Des performances de décodage quasi-optimales sont atteintes à ce jour. Les problématiques s'orientent donc vers la pertinence des implantations logicielles et matérielles d'algorithmes de décodage.

Conception de circuits électroniques et consommation

L'invention de la diode à oscillations par John Ambrose Fleming en 1904, a marqué le point de départ de l'électronique, qui était considérée comme une science fondamentale. La découverte du transistor [Shockley 48] a permis de franchir une étape importante dans le domaine de l'électronique. Les premiers circuits numériques d'Intel, qui intégraient 2,300 transistors (connus sous le nom *processeur 4004* [Faggin 72]) ont été développés en 1971. En 2008, le processeur Quad-Core processeur Intel Xelon 5400 [Intel 08] contient 820 millions de transistors dans une technologie 45nm. Selon les prédictions des experts, l'évolution technologique permettra l'intégration de 3 milliards de transistors par mm^2 en 2015. Cette évolution de la densité d'intégration suit la fameuse loi de Moore introduite en 1965 [Moore 65], qui dit que le nombre de transistors dans un circuit intégré double tous les 18 mois.

Face à une telle quantité de transistors intégrable dans un circuit, deux défis majeurs sont à relever par les concepteur de circuit. Le premier défi concerne purement l'avancement technologique pour que la loi de Moore puisse être vérifiée le plus long temps possible. Les limites de la microélectronique vont bientôt être atteintes. Actuellement, nous atteignons les technologies nanométriques comme pour le processeur Intel Xelon [Intel 08]. Les méthodologies de conception constitue alors le second défi. En effet, des méthodes de conception et de développement favorisant l'intégration dans un temps raisonnable sont indispensables.

La consommation d'un circuit intégré ou d'une mémoire ne cesse d'augmenter. Par exemple, la puissance consommée par un processeur Intel 4004 est moins de 0.11W. Mais cette puissance atteint 150W pour un processeur de type Intel Quad-Core Xelon 5400. La consommation devient alors un critère discriminant crucial en terme de performance.

Le problème de la consommation se pose en particulier pour des systèmes de type énergie limitée, tels que la téléphonie mobile, Wifi, ou Wimax. La maîtrise de la consommation permet d'une part de réduire le coût matériel du système de refroidissement, et d'autre part de prolonger la vie de batteries. Cela implique donc non seulement un enjeu économique considérable sur le produit terminal, mais aussi un facteur écologique. Le succès des futurs produits électroniques dépend à la fois de sa faible consommation et d'un prix modeste. Il s'avère donc primordial de maîtriser la consommation des systèmes numériques. Dans ce contexte, l'optimisation de la consommation d'un circuit doit être prise en considération tout au long du flot de conception. Au niveau conception, il s'avère plus efficace d'optimiser le système au niveau algorithmique et architectural qu'au niveau logique ou transistor.

Problématique et contributions

Nous situons nos travaux dans la problématique de la consommation d'un turbo-décodeur à travers l'exploration de solutions algorithmiques et architecturales. Les codes correcteurs d'erreurs, en particulier les turbocodes, sont indispensables pour de nombreuses applications mobiles, telles que UMTS, LTE, Wifi et Wimax. Ce type d'application dispose d'une propriété commune : l'énergie est fournie par des batteries et elle est donc limitée dans le temps. Par conséquent, il est nécessaire de considérer la problématique consommation afin d'améliorer les capacités de ces systèmes de communication.

Dans nos travaux, nous tentons de maîtriser d'une part la consommation statique et d'autre part la consommation dynamique. Nous montrons qu'il est possible de réduire la surface du circuit en réduisant la taille de la mémoire dans un turbo-décodeur sans dégrader les performances en terme de TEB (Taux d'Erreur Binaire). Concernant la consommation dynamique, nous explorons l'activité de l'algorithme de turbo-décodage en introduisant un codage à la réception. Nous définissons ensuite différentes métriques pour caractériser l'impact sur la consommation. Des simulations sont alors effectuées et des courbes sont présentées pour mettre en évidence l'impact au niveau consommation de nos approches.

Au niveau de la mise en œuvre, nous proposons les différentes architectures adaptées à nos approches qui favorisent la consommation. Les solutions architecturales sont ensuite intégrées sur une plateforme de prototypage à base de circuit FPGA. Nous démontrons ainsi le gain apporté en terme de consommation par des mesures réelles.

Organisation du mémoire

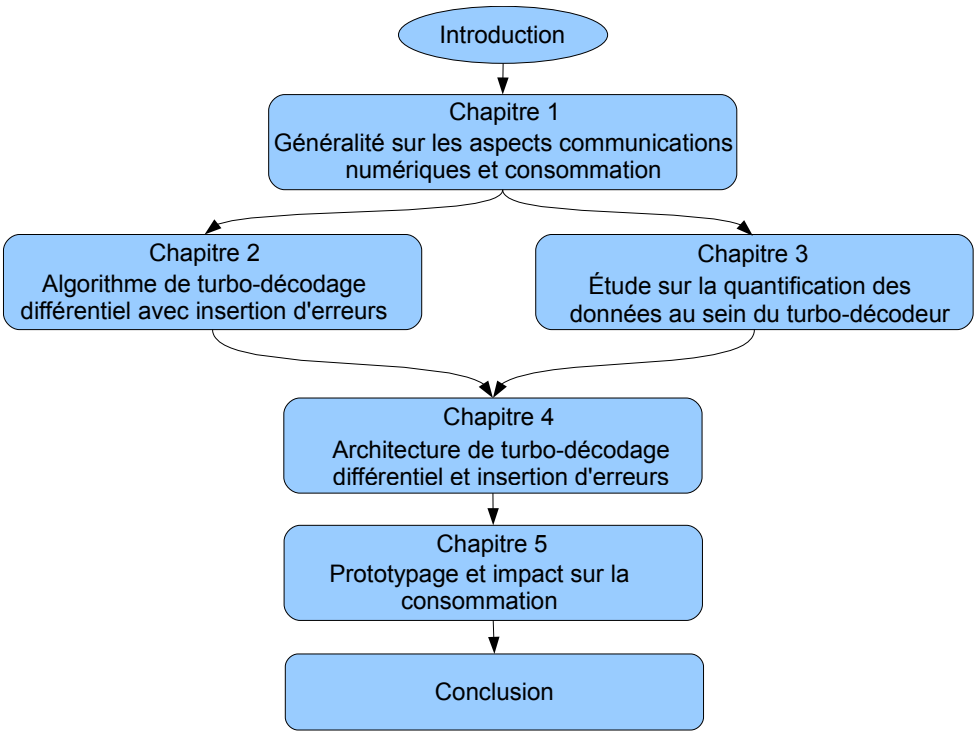


Figure 1 — Organisation du mémoire.

L'organisation du mémoire est résumée dans la Figure 1. Elle permet de situer les différents chapitres dans la structure globale du document. Notre projet s'inscrit dans un contexte de la contribution à la maîtrise de consommation d'un turbo-décodeur. Le mémoire est organisé de la manière suivante.

Deux aspects sont d'abord traités dans le chapitre 1. Nous présentons d'une part les notions de base sur la chaîne de communications numériques et en particulier sur le codage de canal et les turbocodes. Nous donnons ensuite les grands principes des algorithmes de turbo-décodage pour bien comprehendre la suite de l'étude. D'autre part, nous situons la problématique de la consommation pour le turbo-décodage. Une investigation complète des techniques existantes favorisant la consommation est alors présentée à la fois au niveau algorithmique mais aussi au niveau architectural.

Dans le chapitre 2, nous expliquons le principe du codage différentiel ainsi que son application sur les algorithmes de décodage des codes convolutifs. Les métriques théoriques caractérisant l'activité de ce type de système sont clairement définies. Puis, nous proposons différentes approches pour tenter de diminue l'activité d'un système de turbo-décodage. Les performances du turbo-décodage différentiel sont comparées avec celles du turbo-décodage classique.

Le chapitre 3 exploite la maîtrise de la consommation statique d'un système de turbo-décodage. A partir d'un turbo-décodeur classique, nous tentons de diminuer la surface de l'architecture en diminuant la taille de mémoire nécessaire dans un turbo-décodeur. La quantification des données est réexaminée afin d'aboutir à cet objectif. Les études ont montré que nos approches permettent de réduire la taille de la mémoire de façon significative, sans dégrader les performances en terme de TEB (Taux d'Erreur Binaire). Nous obtenons 43% de diminution de la taille de mémoire par rapport à un turbo-décodeur classique.

Dans le chapitre 4, nous détaillons l'architecture d'un décodeur élémentaire souple ou SISO (Soft Input Soft Output). L'organisation des plans mémoires au niveau turbo-décodage est également présentée. A partir d'une architecture classique, nous expliquons les différents composants nécessaires au niveau architectural pour implémenter nos différentes approches. Des solutions architecturales sont apportées pour tenter de réduire à la fois la complexité et la consommation.

Dans le dernier chapitre, nous décrivons l'environnement et les caractéristiques des différents prototypes. Nous avons implémenté une chaîne de communications numériques sur un circuit FPGA. Les différents résultats de synthèse correspondant aux architectures retenues sont données. Les performances de turbo-décodage d'après le prototypage sont relevées. Des mesures de consommation nous permettent d'évaluer l'impact de nos approches.

CHAPITRE

1 Généralité sur les aspects communications numériques et consommations

CE chapitre donne un aperçu général du cadre classique d'un système de communications numériques. Il combine à la fois la théorie de l'information et les communications numériques. Les notions de base concernant le traitement numérique de l'information sont abordées dans ce chapitre. Une description plus exhaustive se trouve dans l'ouvrage [Glavieux 96].

Dans un premier temps le fonctionnement d'un système de communications numériques, constitué de l'émission, de la transmission et de la réception est décrit. Ces différents éléments sont ensuite successivement développés. Deux modèles de canaux, canal symétrique et canal Additif à Bruit Blanc Gaussien (ABBG), sont alors définis dans la section 1.2. La section 1.3 expose la partie codage de canal, y compris les représentations diverses de code convolutif.

À la réception, les algorithmes dédiés au décodage de codes convolutifs, comme les algorithmes de Viterbi et MAP, sont détaillés dans la section 1.4. Afin de faciliter l'implémentation de l'algorithme MAP, l'algorithme sous-optimal Max-Log-MAP est présenté dans la même section. A partir de l'information extrinsèque engendrée par un décodeur élémentaire MAP, le principe de codage et de turbo-décodage s'établit dans la section 1.5. Pour plus d'information, l'ouvrage [Berrou 07] détaille l'ensemble de ces aspects.

La deuxième partie (section 1.6) de ce chapitre est consacrée à la consommation d'un système de communications numériques et en particulier du turbo-décodeur. Une introduction générale sur la consommation dans un circuit CMOS est tout d'abord abordée. Ensuite, nous évoquons spécifiquement la problématique de consommation dans un turbo-décodeur. Dans cette thèse, seul les turbocodes à concaténation parallèle de codes convolutifs sont considérés. Puis une étude concernant l'impact de la consommation du turbo-décodeur est détaillée dans la section 1.6. Cette étude résume les divers travaux de recherche favorisant la maîtrise de cette consommation à la fois aux niveaux algorithmiques et architecturaux.

1.1 Introduction

Un système de communications numériques est principalement constitué d'un bloc d'émission, d'un milieu de transmission et d'un bloc de réception (voir la figure 1.1). Nous nous limitons aux communications radios dans cette thèse. Les sources initiales sont généralement représentées sous forme de signaux analogiques à spectre limité. Ces dernières sont ensuite échantillonnées avec une fréquence d'échantillonnage au moins deux fois supérieure au spectre le plus élevé du signal. Après échantillonnage, les sources se transforment en signaux discrets et numériques, qui sont composés d'une suite d'éléments pouvant prendre ses valeurs parmi les $\mathcal{Q}$ valeurs possibles, représentées par a_k dans la figure 1.1 dont k est l'indice de temps. L'ensemble des valeurs possibles est appelé « alphabet », noté A. Les éléments, dits $\mathcal{Q}$-aires, peuvent être considérés comme des variables aléatoires définies dans un intervalle limité. Lorsque l'alphabet est égal à $A = \{0, 1\}$, son élément est appelé l'élément binaire. Dans la théorie des communications numériques, un élément $\mathcal{Q}$-aire peut être modélisé par $\mathcal{P}$ éléments binaires par l'expression :

$$\mathcal{P} = \lceil \log_2 \mathcal{Q} \rceil, \tag{1.1}$$

où $\lceil x \rceil$ désigne le plus petit entier supérieur ou égal à x.

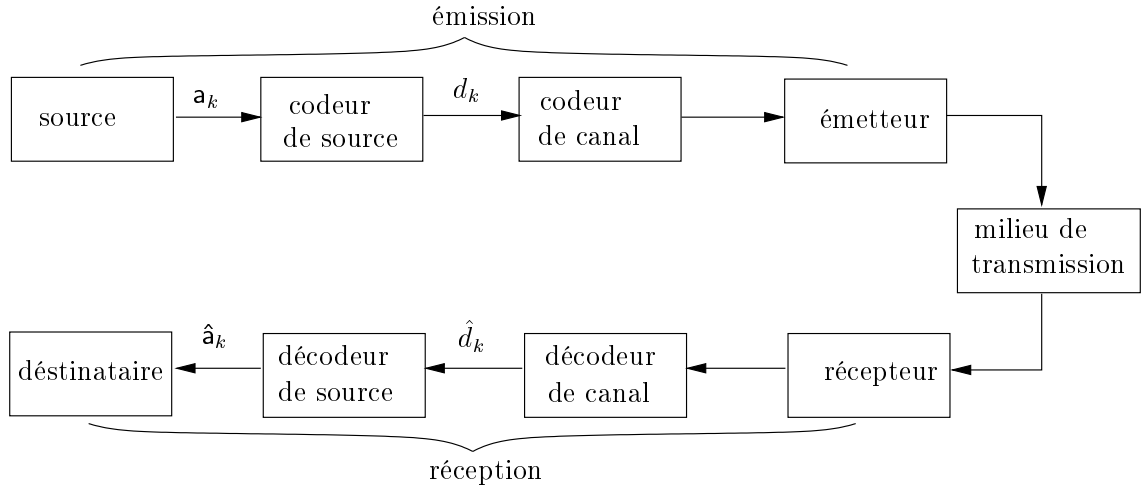


Figure 1.1 — Schéma d'une chaîne de transmissions numériques

Pour réduire la quantité d'information à transmettre, afin de réduire le coût de la transmission, il est nécessaire de compresser les messages numériques avant de les envoyer aux destinataires. La compression de message, dite *codage de source*, permet de rendre le message plus concis et donc de réduire le débit de transmission. Le message après codage de source est noté d_k dans la figure 1.1. Pour lutter contre le bruit introduit par le canal de transmission, le codeur de canal ajoute une partie de redondance. Cette redondance est associée au message à transmettre et forme un mot de code. La règle de codage doit respecter une loi fixée *a priori*, connue à la réception. Cet ajout de redondance, en outre, permet de détecter voire de corriger les erreurs de transmission à l'aide de l'étape de décodage. Le mot de code est classiquement représenté par une suite de symboles sous forme discrète.

L'émetteur, quant à lui, transforme le message numérique en un signal électrique dont les caractéristiques sont adaptées aux propriétés du canal. Dans un environnement radio mobile, l'émetteur a en charge l'opération de modulation qui associe un ou plusieurs éléments binaires

à un symbole caractérisé par son amplitude et sa phase.

Réciproquement, le récepteur (figure 1.1), qui a pour fonction de reconstituer le message émis, effectue une démodulation pour distinguer le mot de code du signal reçu. Nous introduisons deux types de décision : décision dure et décision souple. La décision dure signifie que toutes les valeurs sont prises dans un alphabet binaire $A = \{0, 1\}$. Plus précisément, une valeur est égale à 0 si elle est négative, 1 sinon. Pour la décision souple, il s'agit tout d'abord d'une quantification sur le message reçu. Chaque valeur prise est donc composée d'une suite des éléments binaires. La quantification des données, abordée dans le chapitre 3, peut engendrer une perte d'information pour le décodeur.

D'après la nature du canal, un circuit de décision prend soit une décision dure, soit une décision souple qui alimente donc le décodeur de canal. Lorsque le canal est discret, le récepteur fournit une suite des symboles numériques binaires d'après le message reçu. Dans ce cas, le récepteur est sous-optimal puisque la prise de décision dure par canal provoque une perte irréversible d'information. A l'inverse, lorsque le canal et le démodulateur fournissent une représentation souple du symbole, le décodeur délivre alors le mot le plus vraisemblable par rapport au mot reçu, noté $\hat{d}_k$, en vérifiant si la loi *a priori* est bien respectée. La démodulation avec décision souple permet alors d'améliorer de façon significative les performances du système. Dans notre travail, un canal sans effet mémoire est considéré, c'est pourquoi les symboles sont indépendants les uns des autres.

Correspondant au codage de source, le décodeur de source s'occupe de reconstituer la source initiale a_k à partir du résultat $\hat{d}_k$. Son résultat de reconstitution $\hat{\mathsf{a}}_k$ est ensuite transmis aux destinataires en transformant le signal numérique en signal analogique.

1.2 Canal de transmission

Dans la théorie de communications numériques, le canal de transmission correspond aux éléments entre la sortie du traitement en bande de base de l'émetteur et l'entrée du récepteur en bande de base. Notons que l'entrée du canal de transmission traite des éléments binaires, tandis que la sortie produit une information probabiliste associée aux éléments binaires. Cette information est dite souple et le canal de transmission est dit canal à bruit continu. Si le récepteur prend une décision ferme sur les éléments binaires, alors le canal est dit canal discret et la décision est dite décision dure. La prise de décision dure provoque des pertes d'information irréversibles. Ainsi, le traitement de l'information souple a de meilleure performance par rapport au traitement de l'information dure. Mais ce premier traitement accroît la complexité algorithmique et matérielle du système.

1.2.1 Le canal symétrique

Probabilité d'erreur de transmission

Pour un canal binaire, la probabilité d'un symbole de sortie Y_k par rapport à celle d'entrée X_k est constante, et dite *probabilité d'erreur de transition*. Le canal est sans mémoire si le symbole reçu Y_k ne dépend que de l'élément X_k. Lorsque la probabilité d'erreur est constante et indépendante du symbole X_k, le canal est appelé *canal symétrique binaire*, comme montré sur la figure 1.2, où p représente la probabilité d'erreur par élément binaire. La probabilité

d'erreur est alors symmétrique et peut être écrite de la manière suivante :

$$\begin{cases} p_{01} = p_{10} = p \\ p_{00} = p_{11} = 1 - p \end{cases} \tag{1.2}$$

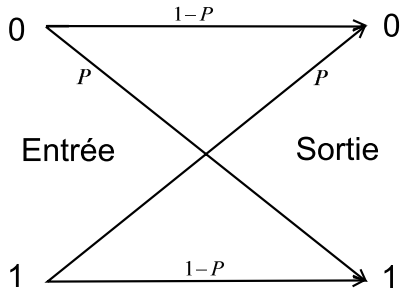


Figure 1.2 — Canal symétrique binaire de probabilité d'erreur p

1.2.2 Le canal additif à bruit blanc gaussien

Le canal binaire symétrique se focalise sur le codeur et le décodeur de canal. Le modulateur et le démodulateur sont supposés avoir un fonctionnement idéal et nous ne nous préoccupons pas de la forme d'onde choisie. Un autre modèle de canal élémentaire est celui qui au contraire permet de prendre en compte les caractéristiques du modulateur et du démodulateur. Classiquement, le canal additif à bruit blanc gaussien (ABBG) ou AWGN (Additive White Gaussian Noise en anglais correspondant) est le plus utilisé.

Pour ce canal, l'entrée X_k est discrète binaire, à valeur dans l'alphabet $\{0, 1\}$ et la sortie est continue, constituée d'échantillons analogiques perturbés par un bruit b_k, additif, blanc, gaussien, stationnaire, centré et indépendant des éléments X_k à transmettre. Ce modèle de canal correspond à la transmission d'un message continu, sans interférence entre symboles (sans mémoire) dans un milieu à atténuation constante avec réception cohérente et décision pondérée. Le canal est représenté symboliquement par la figure 1.3.

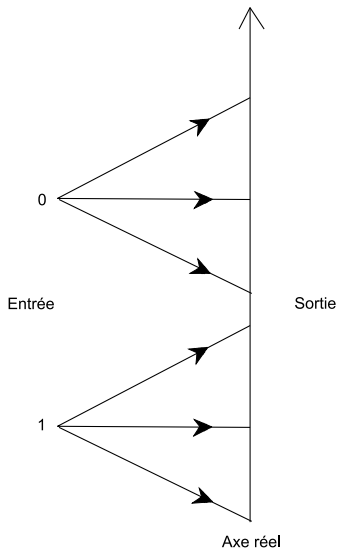


Figure 1.3 — Canal additif à bruit blanc gaussien

Le Bruit Additif Blanc Gaussien (BABG) dispose les propriétés suivantes. Sa densité spectrale de puissance est constante sur une bande de fréquence donnée, dit *bruit blanc*. Les échantillons du bruit représentent une distribution gaussienne avec une variance σ^2. La sortie du canal est la somme de l'entrée discrète et d'un échantillon du bruit. C'est alors que la mesure physique à la sortie du canal à l'instant donné est une valeur réelle et donc continue (voir l'axe réel de la figure 1.3).

Considérons que le BABG est une variable aléatoire gaussienne B de moyenne nulle (centré) et de variance σ^2 et b est une réalisation de B. Après filtrage adapté et échantillonnage périodique, l'amplitude des échantillons représente une densité de probabilité $P_B(b)$, exprimée de la manière suivante :

$$P_B(b) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{b^2}{2\sigma^2}\right), \tag{1.3}$$

où $\sigma^2 = N_0/2$ dont N_0 est la densité spectrale de puissance monolatérale constante.

1.3 Codage de canal

1.3.1 Introduction

Nous avons vu que la probabilité d'erreur d'une transmission numérique est fonction du rapport signal à bruit E_b/N_0. Pour améliorer la qualité de transmission, il est donc possible d'améliorer ce rapport en augmentant la puissance d'émission ou diminuant le bruit du récepteur. Cependant, cela n'est pas toujours une solution convenable puisque la puissance émise est généralement limitée par l'équipement, par les standards, ou simplement par le prix matériel des composants. Une autre solution pour lutter contre les erreurs de transmission consiste à répéter la transmission, dite *transmission redondante*, pour obtenir une meilleure qualité. Bien que cette solution conduise à un accroissement de la complexité des équipements de transmission, elle constitue une approche possible.

Le deuxième théorème de Shannon [Shannon 48] indique qu'il est possible d'avoir une transmission statistiquement fiable avec un codage approprié. Jusqu'à présent, il existe essentiellement deux familles de code : les codes en blocs et les codes convolutifs. Les codes en blocs n'étaient pas abordés dans notre étude, nous nous contentons d'une présentation des codes convolutifs tels que définis dans la norme [3GPP 99]. Le lecteur pourra trouver plus d'information sur la famille des codes en blocs dans [Glavieux 05][Berrou 07].

1.3.2 Code correcteur d'erreurs

Soit K la longueur d'un bloc d'information à l'entrée du codeur, N la longueur du mot correspondant après codage, $K \leq N$, le rendement de code (taux de codage) R est déterminé par le rapport :

$$R = \frac{K}{N} \tag{1.4}$$

Pour une transmission ayant un débit binaire d'information $\mathcal{D}$, le débit de transmission à travers ce canal devient :

$$\mathcal{D}_c = \frac{\mathcal{D}}{R} \text{ bits/s.} \tag{1.5}$$

La relation 1.5 montre que $\mathcal{D}_c$ doit être supérieur à $\mathcal{D}$ car $R \leq 1$. En effet, la transmission de la redondance exige une augmentation du débit par rapport au système non codé. De ce fait,

il est nécessaire d'élargir la bande passante. Ainsi, le principal inconvénient introduit par la fonction de codage réside dans le fait que l'efficacité spectrale est diminuée.

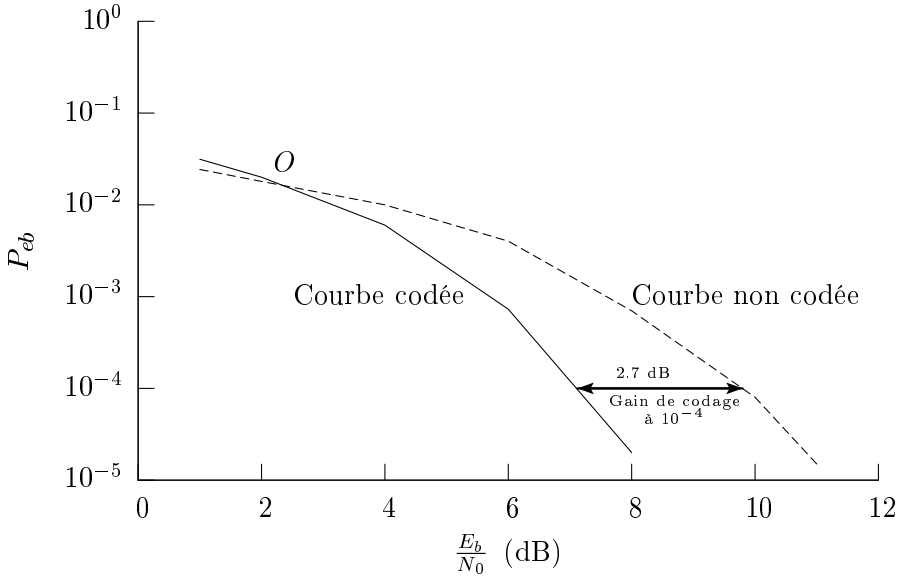


Figure 1.4 — Illustration du gain de codage pour une $P_{eb} = 10^{-4}$

Soit $\mathbf{P}_c$ la puissance moyenne du signal modulé, E_c l'énergie binaire par symbole, E_b l'énergie binaire d'information, nous pouvons établir la relation :

$$\mathbf{P}_c = \mathcal{D}_c \cdot E_c = \frac{\mathcal{D}}{R} \cdot E_c. \tag{1.6}$$

L'énergie reçue par élément binaire transmis E_c est égale à :

$$E_c = R \cdot E_b. \tag{1.7}$$

Généralement, le canal est qualifié par le rapport signal à bruit E_b/N_0, exprimé en décibel(dB) :

$$(\frac{E_b}{N_0})_{\text{dB}} = (\frac{E_c}{N_0})_{\text{dB}} + 10\log(\frac{1}{R}). \tag{1.8}$$

Dans le cas d'une transmission sans codage ($R = 1$), le rapport signal à bruit E_b/N_0) vaut E_c/N_0. En revanche, ce rapport E_b/N_0 est augmenté d'un terme $10\log(\frac{1}{R})$ lors du codage ($R < 1$) si le même type de modulation est considéré. Une énergie supplémentaire est alors requise pour transmettre les bits de redondance.

Les performances du codage de canal sont représentées par la probabilité d'erreur par bit P_{eb}, dite *Taux d'Erreur Binaire (TEB)* après décodage, en fonction du rapport E_b/N_0. Le gain apporté par le codage, appelé *gain de codage*, correspond à la différence des rapports E_b/N_0 existante entre la courbe de taux d'erreur binaire sans codage et celle avec codage pour un taux d'erreur fixé. A titre d'exemple, la figure 1.4 illustre un gain de codage de 2.7 dB pour une probabilité d'erreur à 10^{-4}. Lorsque la probabilité d'erreur binaire est supérieure au point de chevauchement O de la figure 1.4, le codage n'apporte pas d'interêt. En revanche, les gains de codage accroissent en fonction du rapport E_b/N_0 jusqu'à obtenir une valeur asymptotique à partir du point O. En conséquence, le codage de canal n'est pas avantageux lorsque la qualité du canal est mauvaise, c'est-à-dire, un rapport E_b/N_0 faible.

1.3.3 Code convolutif

Les codes convolutifs, introduits en 1955 par Elias [Elias 55], forment une classe efficace de codes correcteurs d'erreurs. Ce sont les codes les plus utilisés dans le système de télécommunications fixes et mobiles. Théoriquement, ils ont les mêmes caractéristiques que les codes en blocs [Berrou 07] sauf pour la valeur de leur dimension et leur longueur. Les codes convolutifs s'appliquent sur une séquence infinie de symboles d'information et génèrent une séquence infinie de symboles codés.

1.3.3.1 Représentation générique d'un codeur convolutif

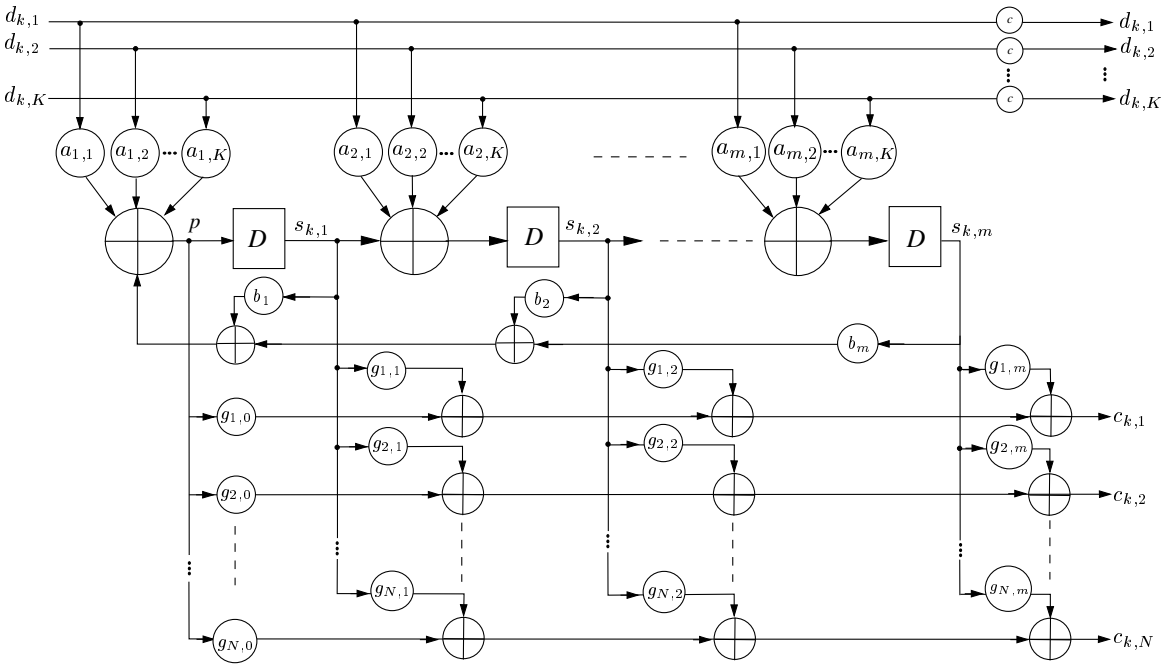


Figure 1.5 — Représentation générique d'un codeur convolutif.

Pour un codeur convolutif, le symbole de sortie dépend non seulement de celui d'entrée, mais aussi des m symboles précédents. Il existe donc un effet de mémorisation sur le codeur convolutif. La quantité $v = m + 1$ est appelée la *longueur de contrainte* du code. La figure 1.5 représente un codeur convolutif suffisamment générique. L'état du codeur est représenté par le registre à décalage, composé des m bascules $(s_{k,1}, \cdots, s_{k,m})$. Il y a donc 2^m états possibles.

A l'instant k, un vecteur $\mathbf{d}_k$ de K bits $\mathbf{d}_k = (d_{k,1}, \cdots, d_{k,K})$ est envoyé à l'entrée du codeur. Le code est alors appelé *code K-binaire*. A l'aide d'un additionneur binaire, soit une porte ou-exclusif, le contenu d'une bascule est égal à la somme des éléments d_k, sélectionnés respectivement par les coefficients correspondants non nuls $a_{i,j}$ et le contenu de la bascule précédente, sauf le cas de la première bascule. Cette dernière reçoit la somme des éléments $s_{i,j}$ sélectionnées par les coefficients non nuls b_i et de l'entrée sélectionnée. Dans ce cas, le code est dit *code récursif*. Si tous les coefficients b_i sont nuls, le code est alors dit *code non récursif*. Ainsi, l'état du codeur à l'instant k, souvent noté en binaire ou en décimal, dépend de l'état précédent $\mathbf{e}_{k-1}$ et de l'entrée en cours $\mathbf{d}_k$.

Lorsque $c = 1$, dit *code systématique*, le vecteur $\mathbf{d}_k$ est directement transmis vers la sortie.

Le mot de code **c** s'exprime alors de la manière suivante :

$$\mathbf{c}_k = (d_{k,1}, \cdots, d_{k,K-1}, d_{k,K}, c_{k,1}, \cdots, c_{k,N-1}, c_{k,N}). \tag{1.9}$$

Son rendement de code est égal à $K/(K+N)$. Lorsque $c \neq 1$, dit *code non systématique*, la partie redondante $(c_{k,1}, \cdots, c_{k,N})$ est construite à partir de l'entrée **d** et l'état du codeur **s**. Les résultats c_k sont obtenus en additionnant les contenus des bascules et la valeur en position p, sélectionnés par les coefficients non nuls $g_{i,j}$. Dans ce cas, le rendement de code est égal à K/N.

1.3.3.2 Générateur de code

Considérons un code convolutif ayant les paramètres : $m = 2$, $K = 1$, $N = 2$ et $R = 1/2$, décrit sur la figure 1.6. Chaque sortie du codeur à l'instant k est le produit de convolution entre la suite binaire d_k présentée à l'entrée du codeur et sa réponse impulsionnelle du codeur définie par ses séquences génératrices. Nous avons alors :

$$c_{k,i} = \sum_{\substack{j=0,\cdots,m \\ i=1,\cdots,N}} g_{i,j} d_{k-j} \quad g_{i,j} \in \{0,1\}, \tag{1.10}$$

où j est l'index de la longueur de contrainte et i l'élément du code. Le terme $g_{i,j}$ est le générateur du code qui spécifie les propriétés du codeur. Dans l'exemple de la figure 1.6,

$$g_1 = [g_{1,1}\, g_{1,2}\, g_{1,3}] = [1\,1\,1] = 7_{(\text{octal})} \text{ et } g_2 = [g_{2,1}\, g_{2,2}\, g_{2,3}] = [1\,0\,1] = 5_{(\text{octal})}. \tag{1.11}$$

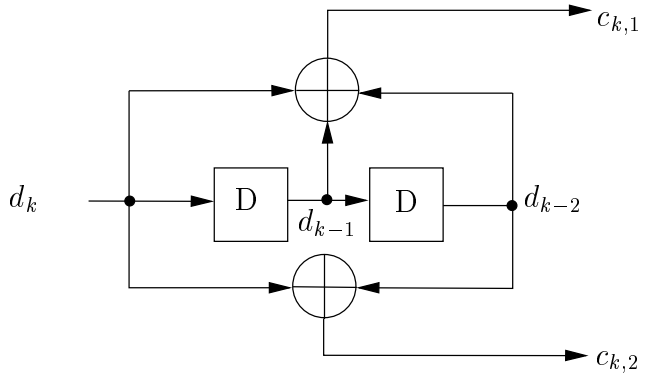


Figure 1.6 — Schéma d'un codeur convolutif pour $m = 2$, $K = 1$ et $R = 1/2$

Les codes convolutifs peuvent aussi être définis à partir des polynômes générateurs. Nous évoquons une variable D (*délai*) équivalent à la variable z^{-1} de la transformée en z. Ainsi, les polynômes générateurs ont pour expression :

$$\begin{cases} g_1 & \rightarrow & G_1 = g_{1,1} + g_{1,2}D + g_{1,3}D^2 \\ g_2 & \rightarrow & G_2 = g_{2,1} + g_{2,2}D + g_{2,3}D^2, \end{cases} \tag{1.12}$$

soit pour l'exemple de la figure 1.6 :

$$\begin{cases} G_1(D) = 1 + D + D^2 \\ G_2(D) = 1 + D^2. \end{cases} \tag{1.13}$$

En tenant compte de l'expression 1.13, le code $C(D)$ en fonction de D peut s'exprimer comme :

$$\begin{cases} C_1(D) = G_1(D)d(D) \\ C_2(D) = G_2(D)d(D). \end{cases} \tag{1.14}$$

Cette représentation par polynômes générateurs peut être appliquée aux codes convolutifs de rendement et de longueur de contrainte quelconque.

1.3.3.3 Représentation des codes convolutifs

Le polynôme générateur caractérise explicitement la nature du codeur. Mais l'introduction de l'effet de mémoire complique la représentation des états à partir des polynômes générateurs. Pour mieux appréhender l'algorithme de décodage, il est néanmoins nécessaire de connaître l'évolution des états au cours du (dé)codage. C'est pourquoi nous décrivons une représentation graphique sous forme d'arbre, de treillis ou encore de diagramme d'états dans cette section. Cette représentation la plus répandue dans la littérature fait apparaître l'évolution des états du codeur. De plus, nous utilisons le code introduit dans la sous-section 1.3.3.2 pour illustrer notre propos.

Diagramme d'états

Le nombre d'états est donné par 2^m, 4 états possibles dans notre cas. Comme dans la représentation d'un automate à machine d'états, l'état est marqué par une valeur décimale de 0 à 3, à laquelle est associée une représentation binaire sur 2 bits, représentés par deux bascules ($3 = (11)_{\text{b}},\ 1 = (01)_{\text{b}},\ \cdots$) dans la figure 1.7. Les entrées 0 et 1 sont respectivement indiquées par le trait pointillé et le trait continu. Les valeurs sortantes (c_k^1, c_k^2) du codeur sont associées à chaque transition du diagramme. Une flèche indique le sens des transitions au niveau des états. Nous constatons que l'état 0 est suivi par l'état 2 si et seulement si l'entrée est égale à 1. Cette représentation montre ainsi l'évolution des états en fonction de l'entrée et de l'état courant.

Diagramme en treillis

Pour connaître l'évolution des états en fonction de la trame entrante du codeur, la figure 1.8 décrit un diagramme en treillis ayant l'état 0 comme l'état initial à l'instant $k = 0$. Les états et leur représentation binaire sont indiqués sur la gauche du treillis.

Pour caractériser un code convolutif, nous introduisons les notions de chemin et de distance libre. Un *chemin* est constitué d'une suite de branches, correspondant à l'évolution des états au fil du temps. Deux chemins se croisent à une intersection appelée *nœud*. En général, il y a 2^m nœuds possibles à un instant donné si m est la longueur de mémoire du codeur.

La *distance libre*, notée d_f, est égale à la Distance Minimale de Hamming (DMH) qui existe entre deux chemins qui divergent puis convergent de nouveau. Nous pouvons suivre un chemin du code en parcourant le treillis dans le temps. Lorsque les valeurs de la séquence à coder ne sont que des 0, l'état du codeur reste alors à l'état 0 et le mot sortant est toujours égal à (00). Dans ce cas, le chemin correspondant est appelé le *chemin Tout A Zéro (TAZ)*. En revanche, supposons que la séquence à coder est $\mathbf{d} = (d_0 d_1 d_2 d_3) = (1111)$, l'évolution des états suit alors le chemin $(0 \rightarrow 2 \rightarrow 3 \rightarrow 3)$ en délivrant les mots binaires en coupe $(11 \rightarrow 01 \rightarrow 10 \rightarrow 10)$.

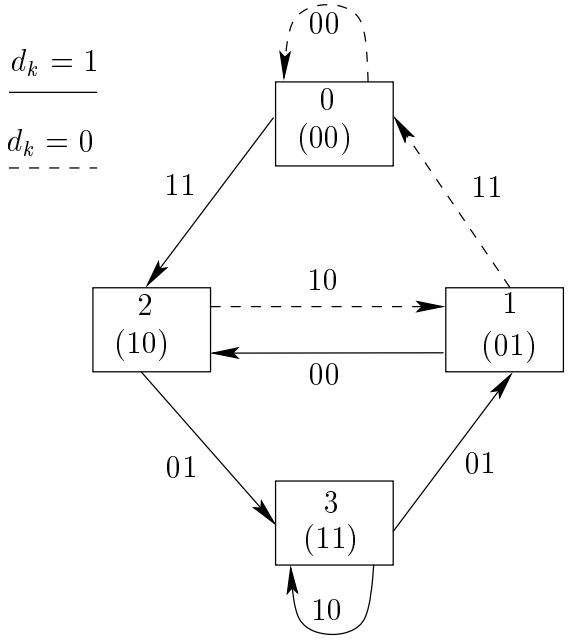


Figure 1.7 — Diagramme des états

Pour un code convolutif de rendement K/N, il existe 2^K branches à partir d'un nœud dans le treillis. Le nombre de branches augmente donc de manière exponentielle. Lorsque K est grand, la complexité du treillis associé devient importante, ce qui rend le décodage plus compliqué.

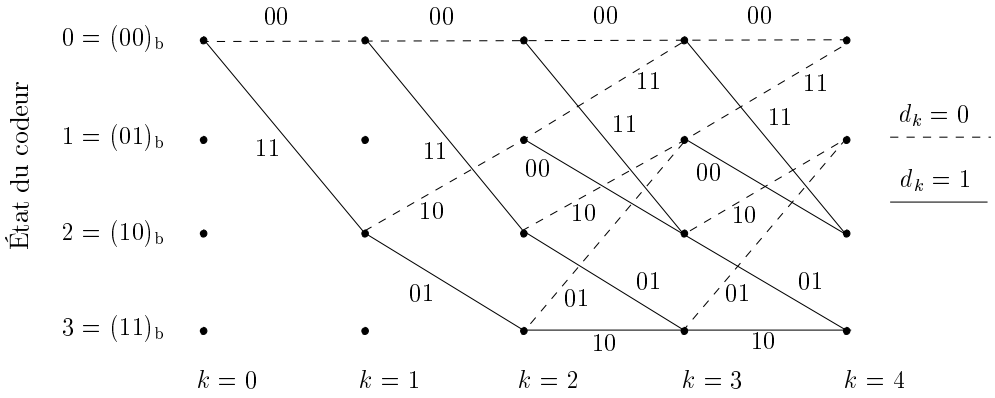


Figure 1.8 — Diagramme en treillis

Diagramme en arbre

La figure 1.9 représente un diagramme en arbre associé au code convolutif de la figure 1.5. Sur ce diagramme, les conventions suivantes sont adoptées :

- le temps s'écoule de la gauche à la droite ;
- lorsque l'entrée binaire est égale à 0 (respectivement 1), le couple binaire en sortie (noté en dessous de la branche) et l'état du codeur sont portés par une branche montante (respectivement descendante) du diagramme en arbre. Les branches montantes et descendantes se séparent d'un point commun (nœud). Chaque nœud dirige 2^K branches vers les nœuds suivants.

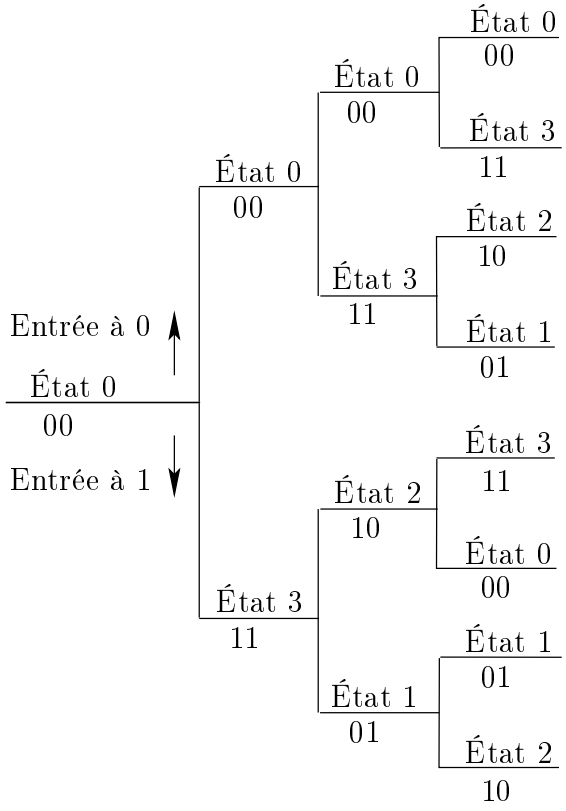


Figure 1.9 — Diagramme en arbre

Comme pour le diagramme en treillis, pour une séquence donnée à l'entrée du codeur, les mots codés sont représentés par un *chemin* constitué d'une suite de branches correspondantes. Le nombre de branches par nœud est déterminé par 2^K, K étant la taille du bloc à l'entrée.

1.3.3.4 Code convolutif poinçonné

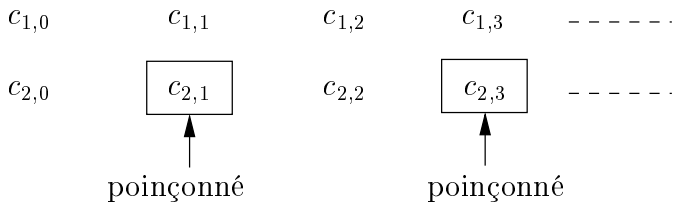


Figure 1.10 — Code poinçonné à partir d'un code convolutif de rendement 1/2

Il est possible de construire un code de rendement plus élevé à partir d'un code de rendement 1/2. Une solution consiste à poinçonner périodiquement le mot codé à la sortie du codeur de manière à diminuer le nombre de symboles de redondance à transmettre. La figure 1.10 représente un code convolutif ayant un rendement 2/3, soit 3 symboles codés pour 2 symboles d'information. Le mot codé en couple à l'instant k est composé des symboles binaires $(c_{1,k}, c_{2,k})$. Ce code est obtenu par un poinçonnage tel qu'un symbole de redondance sur deux soit supprimé (élément encadré dans la figure 1.10). La règle de poinçonnage est représentée

par un masque M, qui est défini par l'expression 1.15 dans notre exemple :

$$M = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}. \tag{1.15}$$

L'élément de M égal à 0 indique que le symbole correspondant est poinçonné. Ce masque correspond donc au cas présenté par la figure 1.10.

1.3.3.5 Code convolutif récursif

Pour un code convolutif de rendement 1/2 ayant une mémoire de longueur m (longueur de contrainte $v = m + 1$), nous avons :

$$\begin{aligned} C_1(D) &= G_1(D)d(D) \\ C_2(D) &= G_2(D)d(D) \end{aligned} \tag{1.16}$$

Lorsque le code est normalisé par rapport à G_1, la nouvelle représentation de code est :

$$\begin{aligned} \tilde{C}_1(D) &= d(D) \\ \tilde{C}_2(D) &= (G_2(D)/G_1(D))d(D) \end{aligned} \tag{1.17}$$

Les relations 1.17 s'écrivent aussi :

$$\begin{aligned} \tilde{C}_1(D) &= G_1(D)A(D) \\ \tilde{C}_2(D) &= G_2(D)A(D) \end{aligned} \tag{1.18}$$

avec

$$A(D) = d(D)/G_1(D), \tag{1.19}$$

d'où

$$d(D) = A(D)G_1(D). \tag{1.20}$$

Rappelons que les opérations sont considérées comme les opérations modulo-2. Considérons la variable de délai D comme la variable de transformation z^{-1}, l'équation 1.20 s'écrit dans le domaine temporel de la manière suivante (figure 1.11) :

$$d_k = a_k + \sum_{j=1}^{m} g_{1,j} a_{k-j}, \tag{1.21}$$

d'où :

$$a_k = d_k + \sum_{j=1}^{m} g_{1,j} a_{k-j}. \tag{1.22}$$

La figure 1.11 représente un Code Convolutif Récursif Systématique (CCRS) construit à partir des polynômes générateurs $G[1, (1 + D^2)/(1 + D + D^2)]$. La partie systématique $c_{k,1}$ est reliée directement à l'entrée d_k, ce qui garantit l'égalité $c_{k,1} = d_k$. La redondance $c_{k,2}$ est générée à partir des éléments a_k, a_{k-1} et a_{k-2}, représentés par 2 bascules successives. De plus, la valeur a_k est égale à la somme des valeurs a_{k-1} et a_{k-2} ainsi que la donnée entrante d_k, d'où le calcul récursif.

Les relations 1.16 et 1.18 sont semblables et elles correspondent à un même code convolutif classique (code non systématique et non récursif) de polynômes générateurs $G_1(D)$ et $G_2(D)$ sur deux séquences différentes d_k et a_k. Ainsi, les deux codes de polynômes générateurs

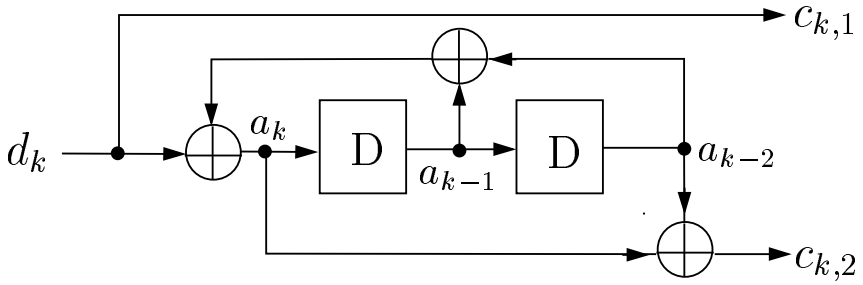


Figure 1.11 — Codeur convolutif systématique récursif construit à partir des polynômes générateurs $G[1,(1+D^2)/(1+D+D^2)]$

$[G_1(D), G_2(D)]$ (code non récursif non systématique) et $[1, G_1(D)/G_2(D)]$ (code récursif et systématique) possèdent un diagramme en treillis ayant la même structure. Les séquences codées délivrées par les deux codeurs sont identiques et par conséquent, les codes ont alors la même distance libre. Une même séquence codée correspond à des séquences d_k et a_k identiques. Mais les deux codeurs ne délivrent pas les mêmes séquences codées à partir de la même séquence d'information d_k. L'intérêt de ce code réside dans le fait qu'il est systématique tout en conservant les propriétés de distance des codes non systématiques. Il est donc particulièrement mieux adapté à la concaténation parallèle (voir la section 1.5).

1.4 Décodage des codes convolutifs

1.4.1 Introduction

Il existe plusieurs algorithmes de décodage pour le code convolutif, parmi lesquels les plus connus sont l'algorithme de Viterbi et l'algorithme MAP (Maximum *a priori*). L'algorithme de Viterbi minimise la probabilité d'erreur sur la séquence des blocs décodés, tandis que l'algorithme MAP travaille sur la probabilité d'erreur à chaque bloc considéré. Rappelons qu'un bloc entrant est constitué de K symboles binaires, $K = 1$ pour le cas binaire (notre cas). Considérons le décodeur reçoit une séquence bruitée et démodulée, ayant une longueur L, l'algorithme de Viterbi estime la vraisemblance à partir de la séquence reçue en fournissant la séquence la plus vraisemblable. Cette approche peut être réalisée de manière assez simple pour une complexité raisonnable (section 1.4.2). En revanche, l'algorithme MAP recherche la solution optimale de décodage en calculant la probabilité d'erreur à chaque bloc considéré. Mais l'implémentation de l'algorithme MAP est difficile en raison de sa complexité de mise en œuvre (section 1.4.3).

1.4.2 Algorithme de Viterbi

Soient
- $\mathbf{d} = (d_0 \cdots d_k \cdots d_{L-1})$ la séquence d'information ;
- $\mathbf{c} = (c_0 \cdots c_k \cdots c_{L-1})$ la séquence codée ;
- $\mathbf{r} = (r_0 \cdots r_k \cdots r_{L-1})$ la séquence reçue par le décodeur (observation).

L'algorithme de Viterbi [Viterbi 67] consiste à chercher la séquence $\hat{\mathbf{d}}$ telle que

$$\hat{\mathbf{d}} = \mathrm{Arg}\max_{\mathbf{d}} p(\mathbf{d}/\mathbf{r}) = \mathrm{Arg}\max_{\mathbf{d}} p(\mathbf{d},\mathbf{r})/p(\mathbf{r}) \ \ \forall \mathbf{d} \tag{1.23}$$

où Arg désigne l'argument $\mathbf{d}$ de la probabilité conditionnelle $p(\mathbf{d}/\mathbf{r})$. Sachant que la probabilité $p(\mathbf{r})$ est indépendante de l'information $\mathbf{d}$. L'algorithme de Viterbi cherche la valeur $\hat{\mathbf{d}}$ maximisant la probabilité $p(\mathbf{d}, \mathbf{r})$. Importons une suite d'états $\mathbf{S} = (S_0 \cdots S_k \cdots S_{\mathcal{M}})$ tel que $S_k = (s_k, s_{k-1}, \cdots, s_{k-v+2})$. La connaissance de la suite $\mathbf{S}$ permet de retrouver l'information $\mathbf{d}$. La suite $\mathbf{S}$ est par conséquent une image de $\mathbf{d}$. De plus, la transition $S_{k-1} \rightarrow S_k$ dépend seulement du bloc d'entrée d_k. L'algorithme de Viterbi revient donc à chercher $\hat{\mathbf{S}}$ tel que

$$\hat{\mathbf{S}} = \text{Arg} \max_{\mathbf{S}} p(\mathbf{S}, \mathbf{r})/p(\mathbf{r}) \ \ \forall \mathbf{S} \tag{1.24}$$

De ce fait, il suffit de maximiser le terme $p(\mathbf{S}, \mathbf{r})$, puisque $p(\mathbf{r})$ est indépendant de l'entrée du codeur $\mathbf{d}$.

$$p(\mathbf{S}, \mathbf{r}) = p(S_0) \prod_{k=1,L} p((r_k, S_k)/S_{k-1}), \tag{1.25}$$

où $p(S_0)$ représente la distribution de l'état initial. Si l'état du codeur est initialisé à l'état 0, la distribution $p(S_0)$ est égale à 1 pour $S_0 = (0 \cdots 0 \cdots 0)$, 0 pour le reste.

Introduisons la métrique de branche γ_k définie par l'équation 1.26 :

$$\gamma_k(s', s) = -\log p((r_k, S_k = s')/S_{k-1} = s). \tag{1.26}$$

En exprimant les expressions 1.24 et 1.25 dans le domaine logarithmique, nous obtenons :

$$\hat{\mathbf{S}} = \text{Arg} \min_{\mathbf{S}} (-\log p(S_0) + \sum_{k=1}^{L} \gamma_k(S_{k-1}, S_k)). \tag{1.27}$$

Le terme $\gamma_k(S_{k-1}, S_k)$ est appelé *métrique de branche* à l'instant k. Elle correspond à la distance de Hamming entre l'observation r_k et le mot porté par le treillis c_k lors d'un canal binaire symétrique, soit $\gamma_k(S_{k-1}, S_k) = D_h(r_k, c_k)$. Pour un canal gaussien, la métrique de branche est égale au carré de la distance euclidienne entre l'observation r_k et le mot c_k, $\gamma_k(S_{k-1}, S_k) = ||r_k - c_k||^2$. Si aucune transition n'est possible entre les états s' et s à l'instant donné, la métrique $\gamma_k(s', s)$ est égale à $\gamma_k(s', s) = +\infty$. De plus, il y a au total 2^N métriques de branche pour une longueur de mot codé N. L'expression 1.27 s'appuie sur la recherche d'un chemin le plus court, dit *chemin survivant*, si la métrique $\gamma_k(s', s)$ est considérée comme la distance associée entre les états $S_{k-1} = s'$ et $S_k = s$. Ce chemin est composé des transitions d'état $S_{k-1} \rightarrow S_k$ associées au symbole d_k.

La métrique de nœud $M_{s,k}^N$ pour l'état s à l'instant k s'exprime par l'expression 1.28 :

$$M_{s,k}^N = \min_{s' \rightarrow s} (M_{s',k-1}^N + \gamma_k(s', s)) \ \ \forall s'. \tag{1.28}$$

Elle signifie le maximum de vraisemblance partant du nœud S_0 au nœud $S_k = s$.

Au niveau de la mise en œuvre, le décodeur de Viterbi calcule d'abord les métriques de branche γ_k à partir du mot reçu. Pour un décodage binaire de rendement 1/2 et $N = 2$, il y a donc 4 métriques de branche mesurant respectivement la vraisemblance d'observation r_k par rapport au mot codé c_k. Ensuite, un module ACS (Add Compare Select en anglais) sélectionne la métrique de nœud minimale $M_{s,k}^N$ pour chaque nœud $S_k = s$ en additionnant la métrique de branche appropriée $\gamma_k(s', s)$ à la métrique de nœud précédente $M_{s',k-1}^N$. Les résultats de sélection, 2^{v-1} chemins au total, sont ensuite stockés, ce qui permet de remonter la séquence décodée lors d'un parcours inverse du treillis.

Dans sa version de base, le décodeur de Viterbi ne donne le résultat qu'après la réception totale de la séquence considérée. En effet, les chemins survivants convergent vers un nœud de l'instant $k+n$ à l'instant k, c'est-à-dire, tous les chemins survivants à partir de l'instant k partent du même nœud. Ainsi, le décodeur peut sortir les résultats de l'instant 0 à l'instant k puisque le décodage suivant n'influencera pas les résultats précédents. En générale, la valeur de n vaut $n = 5v$ ou $6v$ pour $R = 1/2$. Chaque nœud reçoit 2^K branches. A chaque étape, il y a 2^N calculs de métrique de branche et 2^{K+v-1} additions dans les modules ACS. La complexité de calcul accroît de manière exponentielle. C'est pourquoi l'implémentation de l'algorithme de Viterbi est limitée à des petites valeurs de K et v. Typiquement, $v \leq 8$. Au-delà de cette limite, l'algorithme de décodage séquentiel comme celui de Fano [Fano 63] est adopté en raison de sa complexité réduite.

Bien que l'algorithme de Viterbi soit une mesure de la probabilité d'erreur sur une séquence décodée, il conduit aussi, avec une bonne approximation, à un minimum de la probabilité d'erreur sur les blocs d'information lorsque le rapport signal à bruit dépasse quelques décibels.

1.4.3 Algorithme MAP

L'algorithme MAP, abrégé de Maximum *a posteriori*, appelé aussi l'algorithme BCJR [Bahl 74], est une solution optimale pour le décodage des codes convolutifs. Il fournit la fiabilité associée à chaque bloc décodé. Pour ce faire, le décodeur cherche tous les blocs d'information $k = 0, \cdots, L-1$ de manière exhaustive tel que :

$$p(\hat{d_k}/\mathbf{r}) \geq p(d_k/\mathbf{r}) \quad \forall d_k. \tag{1.29}$$

Trois termes doivent être définis avant de détailler les caractéristiques de l'algorithme MAP :

- $\bar{\alpha}_k(s)$: métrique de nœud dans le sens *Aller* représentant la vraisemblance conjointe des observations $(r_0, \cdots, r_k)$ et de l'état $S_k = s$, soit $\bar{\alpha}_k(s) = p(r_0, \cdots, r_k, S_k = s)$;
- $\bar{\beta}_k(s)$: métrique de nœud dans le sens *Retour* représentant la vraisemblance conditionnelle à l'état S_k, soit $\bar{\beta}_k(s) = p(r_{k+1}, \cdots, r_{L-1}/S_k = s)$;
- $\bar{\psi}_k(s, s')$: la vraisemblance conjointe de r_k, S_k et S_{k-1}. $\bar{\psi}_k(s, s') \propto p(S_{k-1} = s, S_k = s'/r_k)$ où le symbole $\propto$ est égal à un facteur de proportionnalité près.

L'algorithme MAP se décompose de la manière suivante :

1. Calculer la métrique de nœud $\bar{\alpha}_k(s)$
   Si $\bar{\alpha}_0(s) = p(S_0 = s)$, alors la définition de $\bar{\alpha}_k(s)$ est :

$$\bar{\alpha}_k(s) = \sum_{s'/s' \to s} p(r_0, \cdots, r_k, S_{k-1}, S_k). \tag{1.30}$$

   Nous en déduisons :

$$\bar{\alpha}_k(s) = \sum_{s'/s' \to s} \bar{\alpha}_{k-1}(s') \exp(-\gamma_{k-1}(s', s)) \tag{1.31}$$

   où le terme $s'/s' \to s$ porte l'ensemble de s' tel qu'il existe une transition de l'état s' à l'état s.

2. Calculer la métrique de nœud $\bar{\beta}_k(s)$

Par analogie, nous obtenons :

$$\begin{aligned} \bar{\beta}_k(s) &= \sum_{s'/s' \to s} p(r_{k+1}, \cdots, y_{L-1}, S_k = s'/S_k = s) \\ &= \sum_{s'/s' \to s} \bar{\beta}_{k+1}(s') \exp(-\gamma_k(s', s)); \end{aligned} \tag{1.32}$$

3. Calculer la vraisemblance conjointe $\bar{\psi}_k(s, s')$ d'après l'expression suivante :

$$\bar{\psi}_k(s', s) = \bar{\alpha}_{k-1}(s) \exp(-\gamma_k(s', s)) \bar{\beta}_k(s'), \tag{1.33}$$

où la métrique de branche γ_k est antérieurement définie par l'équation 1.26 ;

4. Prendre la décision $\hat{d_k}$ d'après le critère MAP à partir de la vraisemblance conjointe $\bar{\psi}_k$:

$$\hat{d_k} = \text{Arg} \max_{d_k} \sum_{(s,s')/d_k = d(s',s)} \bar{\psi}_k(s', s) \tag{1.34}$$

et calculer la fiabilité correspondante :

$$p(\hat{d_k}/\mathbf{r}) = p(\hat{d_k}, \mathbf{r})/p(\mathbf{r}) = \frac{\sum_{(s',s)/d_k = d(s',s)} \bar{\psi}_k(s', s)}{\sum_{(s',s)} \bar{\psi}_k(s', s)} \tag{1.35}$$

où le terme $(s', s)/d_k = d(s', s)$ signifie que la somme porte l'ensemble des branches $s' \to s$ qui correspondent à un bloc d'information d_k.

Comme le montre l'expression 1.35, le décodage selon le critère MAP représente un interêt majeur tel que chaque bloc décodé $\hat{d_k}$ est associé à une fiabilité sous forme de la vraisemblance conditionnellement à l'entrée $\mathbf{r}$. Cette information peut être utilisée pour le décodage pondéré du décodeur extérieur en cas de concaténation en série ou parallèle. C'est aussi sur la base de cette information que l'information extrinsèque est utilisée dans le décodage itératif comme turbo-décodage.

1.4.4 Algorithme Max-Log-MAP

Le décodage selon le critère MAP est difficile à intégrer dans un circuit dû à la complexité de calcul, imposée notamment par les multiplications, les divisions et les opérations exponentielles. En pratique, il existe plusieurs algorithmes sous-optimaux qui ont pour objectif de faciliter l'implémentation.

Pour contourner les opérations arithmétiques très complexes citées au-dessus, l'une des méthodes efficaces permettant de simplifier ces opérations est de transférer le problème dans un espace logarithmique. Cette modification permet de transformer respectivement les multiplications, les divisions et les opérations exponentielles en additions, soustractions et multiplications.

La transformation algorithmique conduit à la redéfinition des calculs de l'algorithme MAP de la manière suivante :

- logarithme de la métrique de nœud avant $\alpha_k(s)$: $\alpha_k(s) = -\log \bar{\alpha}_k(s)$;
- logarithme de la métrique de nœud arrière $\beta_k(s)$: $\beta_k(s) = -\log \bar{\beta}_k(s)$;

– logarithme de la vraisemblance conjointe $\psi_k(s, s')$: $\psi_k(s, s') = -\log \bar{\psi}_k(s, s')$.

A partir de la relation 1.31, nous obtenons :

$$\alpha_k(s) = \log \sum_{s'/s' \to s} \exp(-\alpha_{k-1}(s') - \gamma_k(s', s)). \tag{1.36}$$

Or

$$\log(\mathrm{e}^x + \mathrm{e}^y) = \max(x, y) + \log(1 + \mathrm{e}^{-|y-x|}) \tag{1.37}$$

et

$$\log(\mathrm{e}^x + \mathrm{e}^y) \simeq \max(x, y) \qquad \text{quand } |y - x| \gg 0. \tag{1.38}$$

En utilisant ces propriétés, l'expression 1.36 peut être simplifiée par :

$$\alpha_k(s) = \max_{s'/s' \to s} (-\alpha_{k-1}(s') - \gamma_{k-1}(s', s)). \tag{1.39}$$

En pratique, nous prenons le minimum défini par l'expression suivante :

$$\alpha_k(s) = \min_{s'/s' \to s} (\alpha_{k-1}(s') + \gamma_{k-1}(s', s)). \tag{1.40}$$

L'expression 1.39 est valide lorsque le rapport signal à bruit (SNR) de la transmission est suffisamment élevé, typiquement supérieur à quelques décibels.

De la même manière, nous déduisons la métrique $\beta_k(s)$ d'après la relation 1.32 :

$$\beta_k(s) \simeq \min_{s'/s' \to s} (\beta_{k+1}(s') + \gamma_k(s', s)). \tag{1.41}$$

D'après l'expression 1.33, le logarithme de vraisemblance conjointe ψ s'écrit alors :

$$\psi_k(s', s) = \alpha_k + \gamma_k(s', s) + \beta_{k+1}(s'). \tag{1.42}$$

Le logarithme de vraisemblance Φ_k par rapport au d_k vaut :

$$\begin{aligned} \Phi_k(d_k) &= -\log p(d_k, \mathbf{r}) \\ &= -\max_{(s',s)/d_k = d(s',s)} (-\psi_k(s', s)) \\ &= \min_{(s',s)/d_k = d(s',s)} (\psi_k(s', s)) \end{aligned} \tag{1.43}$$

Cette version de l'algorithme MAP est connue sous le nom algorithme Max-Log-MAP, qui conduit à la décision suivante [Robertson 95] :

$$\hat{d}_k = \mathrm{Arg} \min_{d_k} \Phi_k(d_k). \tag{1.44}$$

1.4.5 Performances des codes convolutifs

Les performances d'un code sont caractérisées par la probabilité d'erreur binaire après décodage d'une transmission de code bruitée. L'ouvrage [Berrou 07] a détaillé différentes bornes supérieures pour les performances d'un code convolutif. Nous nous contentons ici de présenter les bornes générales d'un code convolutif pour deux canaux : canal gaussien et canal symétrique.

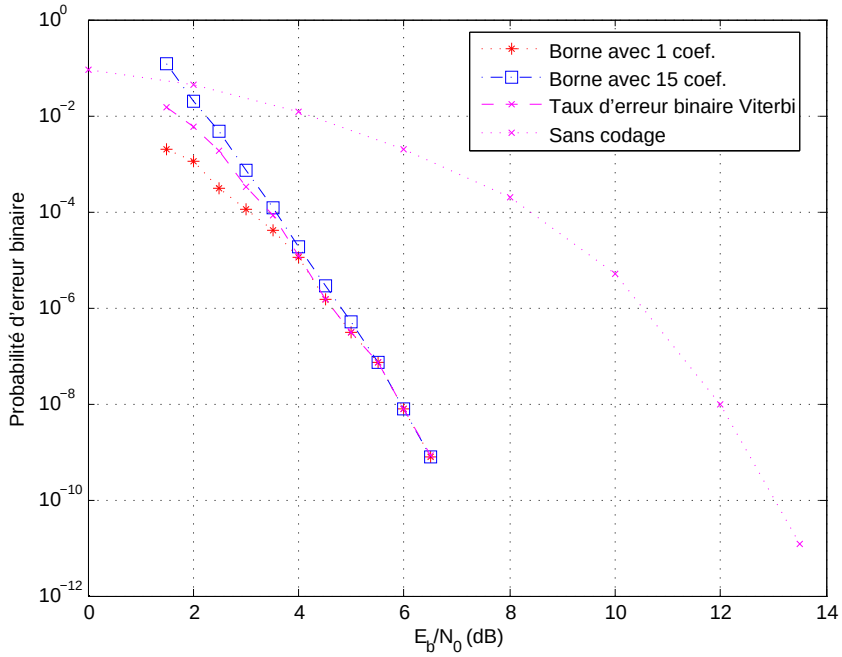


Figure 1.12 — Probabilité d'erreur par élément binaire avec décodage pondéré pour le code $G(133, 171)_O$, extraite de l'ouvrage [Berrou 07](page 214). Le rendement de codage est égal à $R = 1/2$.

Pour un code convolutif de rendement $R = K/N$, la probabilité d'erreur P_{eb} peut être bornée supérieurement par (si la longueur d'information L est grande) :

$$P_{eb} \leq \frac{1}{K} \sum_{d=d_f}^{\infty} w(d) P(d), \tag{1.45}$$

où le symbole d_f décrit la distance libre du code et le terme $w(d)$ est appelé le *spectre de code*. Leurs valeurs sont listées dans l'ouvrage [Berrou 07] pour divers codes convolutifs.

A fort rapport E_b/N_0 sur un canal gaussien, l'expression 1.45 peut se limiter au premier terme :

$$P_{eb} \simeq \frac{1}{2K} \text{erfc} \sqrt{R d_f \frac{E_b}{N_0}} \text{ pour } \quad \frac{E_b}{N_0} \gg 1, \tag{1.46}$$

où le terme erfc(x) note la fonction d'erreur complémentaire sur x.

Sur un canal symétrique, la probabilité $P(d)$ de l'expression 1.45 a pour expression :

$$P(d) < (\sqrt{4p(1-p)})^d. \tag{1.47}$$

Il est encore possible de raffiner la borne sur ce même canal par l'expression suivante :

$$P_{eb} \leq \sum_{d=d_f}^{\infty} w(d) (\sqrt{4p(1-p)})^d. \tag{1.48}$$

Si nous considérons une modulation de phase à deux ou quatre états avec démodulation

cohérente, la probabilité p peut s'écrire :

$$p = \frac{1}{2}\text{erfc}\sqrt{\frac{E_b}{N_0}} < \frac{1}{2}\exp(-\frac{E_b}{N_0}). \tag{1.49}$$

L'expression 1.48 devient alors :

$$P_e < \exp(-5\frac{E_b}{N_0}). \tag{1.50}$$

En résumé, le gain de codage pour un canal gaussien est de 4 dB pour un code convolutif ayant les paramètres : $R = 1/2$, $d_f = 5$ et $w(d_f) = 1$. De plus, Ce gain de codage est de 3 dB supérieur au celui sur le canal symétrique [Berrou 07]. Sur la figure 1.12, les bornes des performances sont illustrées avec 1 coefficient puis 15 coefficients de l'expression 1.45. Les performances d'un décodeur de Viterbi y sont également présentées. Le code considéré a pour polynômes générateurs $G[133, 171]$ exprimés en octet. Son rendement de codage est égal à $1/2$.

1.5 Turbo-décodage dédié aux codes convolutifs

1.5.1 Turbocodes : Codes concaténés en parallèle

Les turbocodes [Berrou 93] constituent une famille de code correcteur d'erreurs construite, dans leur version originale, à partir d'une concaténation en parallèle de deux codes CCRS identiques, reliés par un entrelaceur $\prod$ (figure 1.13). La séquence de données d est codée d'abord dans l'ordre naturel par le codeur 1 produisant la première séquence de redondance y_1. En parallèle, le codeur 2 délivre la seconde séquence de redondance y_2 à partir de la séquence d entrelacée. Le codage en parallèle ajoute alors deux symboles redondants y_1, y_2 pour un symbole codé x et donc améliore la diversité du code. En effet, cette structure permet de distribuer l'énergie disponible à l'émission entre différents symboles redondants de telle sorte que le décodeur correspondant puisse tirer le meilleur profit d'un effet de diversité. En outre, il est possible d'effectuer un poinçonnage éventuel pour obtenir différents rendements. Plus faible est le rendement de codage, plus important est l'effet de diversité.

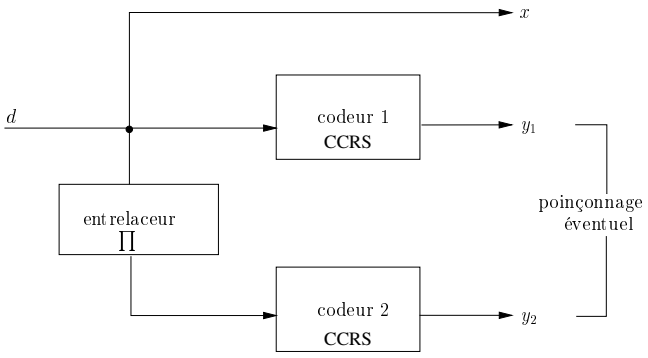


Figure 1.13 — Concaténation parallèle de codes convolutifs

1.5.1.1 Entrelacement

L'entrelacement consiste à disperser la séquence des données dans le temps. Cette technique rend de grands services en communications numériques pour réduire l'effet d'évanouisse-

ment. Plus particulièrement, ce mécanisme est très efficace pour lutter contre les perturbations des symboles consécutifs. Dans le cadre des turbocodes, l'entrelacement est une opération primordiale. Il existe une littérature riche sur le sujet et le lecteur intéressé trouvera un point d'entrée dans la lecture de [Berrou 07].

Un exemple d'entrelaceur simple à réaliser est présenté par la relation suivante :

$$i = \prod(j) = \mathfrak{P}j + i_0 \qquad \text{mod. } L, \tag{1.51}$$

où j et i sont respectivement les positions avant et après l'entrelacement, $\mathfrak{P}$ est un entier premier avec L et i_0 est l'indice du départ.

Ce type d'entrelacement est appelé *entrelacement régulier*. Il joue bien un rôle de dispersion des données entre l'ordre naturel et l'ordre entrelacé, mais ne garantit pas une DMH (Distance Minimale de Hamming) élevée du turbocode. Il apparaît alors des motifs réguliers d'erreurs lors du décodage. Pour lutter contre ces motifs et augmenter la DMH, il faut introduire une dose d'irrégularité. Pour ce faire, une des méthodes efficaces est d'introduire un décalage $\mathfrak{Q}(j)$ en fonction de l'indice j, dans l'équation 1.51. Nous obtenons :

$$i = \prod(j) = \mathfrak{P}j + \mathfrak{Q}(j) + i_0 \qquad \text{mod. } L, \tag{1.52}$$

où $\mathfrak{Q}(j) = \mathfrak{C}(\mathfrak{a}(j) \cdot \mathfrak{P} + \mathfrak{b}(j))$. Les entiers positifs $\mathfrak{a}(j)$ et $\mathfrak{b}(j)$, le plus souvent de valeurs 0 à 8, sont périodiques et leur période $\mathfrak{C}$ est un diviseur de L.

Un exemple pour les valeurs de $\mathfrak{Q}(j)$ est donné à la suite en supposant que $\mathfrak{a} = 0$ ou 1 et $\mathfrak{C} = 4$. De plus, l'une des valeurs $\mathfrak{Q}(j)$ peut être systématiquement 0.

$$\begin{array}{ll} \text{si} j = 0 \text{ mod. } 4, & \text{alors } \mathfrak{Q} = 0 \\ \text{si} j = 1 \text{ mod. } 4, & \text{alors } \mathfrak{Q} = 4\mathfrak{P} + 4\mathfrak{b}_1 \\ \text{si} j = 2 \text{ mod. } 4, & \text{alors } \mathfrak{Q} = 4\mathfrak{b}_2 \\ \text{si} j = 3 \text{ mod. } 4, & \text{alors } \mathfrak{Q} = 4\mathfrak{P} + 4\mathfrak{b}_3 \end{array} \tag{1.53}$$

$$\begin{array}{ll} \text{si} j = 0 \text{ mod. } 4, & \text{alors } \mathfrak{Q} = 0 \\ \text{si} j = 1 \text{ mod. } 4, & \text{alors } \mathfrak{Q} = 4\mathfrak{b}_1 \\ \text{si} j = 2 \text{ mod. } 4, & \text{alors } \mathfrak{Q} = 4\mathfrak{P} + 4\mathfrak{b}_2 \\ \text{si} j = 3 \text{ mod. } 4, & \text{alors } \mathfrak{Q} = 4\mathfrak{P} + 4\mathfrak{b}_3 \end{array} \tag{1.54}$$

Les paramètres $\mathfrak{P}, \mathfrak{b}_1, \mathfrak{b}_2, \mathfrak{b}_3$ peuvent être déterminés par la méthode proposée dans [C.Berrou 04]. L'entrelacement défini dans les normes DVB-RCS [ETSI 00], DVB-RCT [C.Berrou 01] et WiMax [802.16a 03] est inspiré des expressions 1.53 et 1.54.

1.5.1.2 Fermeture de treillis

Le décodage de code convolutif dépend des informations antérieures et des informations postérieures. Or, pour décoder une séquence de mot reçu, les deux extrémités ne bénéficient pas des informations antérieures et postérieures. Par conséquent, les performances de décodage sont dégradées puisque les deux extrémités sont moins bien protégées. Le problème peut être contourné lorsque l'état final du codeur est connu par le décodeur. Pour ce faire, une solution consiste à transmettre l'état final au récepteur. Mais cette solution diminue le rendement du codage et donc l'efficacité spectrale. Il est aussi possible d'imposer au codeur un état connu par le décodeur, par exemple, l'état 0. Ces différentes techniques sont connues sous le terme de « fermeture de treillis ».

Pour les turbocodes, la fermeture de deux treillis est à considérer et il peut y avoir plusieurs solutions envisageables.

- Ne rien faire, dans ce cas, les informations situées à la fin de séquence sont moins bien protégées autant dans l'ordre naturel que dans l'ordre permuté. Ceci conduit à une diminution du gain asymptotique mais peut être compatible avec certaines applications. Il est aussi à noter que la non-fermeture des treillis pénalise plus fortement le TEP (Taux d'Erreur par Paquet) que le TEB (Taux d'Erreur Binaire).
- Fermer le treillis d'un ou de deux codeurs élémentaires, comme proposent les standards CCSDS [CCSDS 98] et UMTS [3GPP 99]. Les bits de fermeture sont indépendants d'un codeur à l'autre. Ces bits ne subissent donc pas de l'effet turbo. Il en résulte que les performances du décodage sont légèrement dégradées.
- Utiliser un entrelaceur permettant de fermer automatiquement le treillis sans ajouter des bits de fermeture. Cette solution ne diminue pas le rendement de codage mais impose certaines contraintes à l'entrelaceur, qui sont parfois difficiles à respecter.
- Adopter un code circulaire. Cette technique retenue dans les standards DVB-RCS [ETSI 00] et DVB-RCT [C.Berrou 01] assure l'égalité entre l'état initial et l'état final. Le treillis est considéré comme infini par le décodeur correspondant. Cette technique est connue sous le nom de *tail biting* pour les codes non-récursifs [Weiss 01] [Weiss 98].

1.5.2 Code convolutif circulaire

L'état initial et l'état final d'un code circulaire sont identique. Cet état initial est appelé *l'état circulaire*. Le treillis est implicitement considéré infini par le décodeur [Weiss 98] [Weiss 01].

Pour un codeur convolutif, Il est possible d'établir une relation entre l'état $\mathbf{s}_{k+1}$ du codeur à un instant $k+1$, son état $\mathbf{s}_k$ et le vecteur $\mathbf{w}_k^N$ à un instant k :

$$\mathbf{s}_{k+1} = \mathbf{A} \cdot \mathbf{s}_k^N + \mathbf{w}_k^N \tag{1.55}$$

Pour un code convolutif de générateurs polynôme $G[1, (1+D^2)/(1+D+D^2)]$ (figure 1.11), les paramètres $\mathbf{s}$, $\mathbf{A}$ et $\mathbf{w}$ ont pour valeur :

$$\mathbf{s} = \begin{pmatrix} s_1 \\ s_2 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \text{ et } \mathbf{w}_k^N = \begin{pmatrix} d_k \\ 0 \end{pmatrix}.$$

Si l'état du codeur est initialisé à l'état 0, à partir de l'équation 1.55, l'état final après codage de L données s'écrit :

$$\mathbf{s}_L^0 = \sum_{i=1}^{L} \mathbf{A}^{L-1} \cdot \mathbf{w}_{i-1}. \tag{1.56}$$

Lorsque l'état initial est à l'état $\mathbf{s_c}$, l'état final s'exprime de la manière suivante :

$$\mathbf{s'}_L = \mathbf{A}^L \cdot \mathbf{s}_c + \sum_{i=1}^{L} \mathbf{A}^{L-1} \cdot \mathbf{w}_{i-1} \tag{1.57}$$

En posant $\mathbf{s'}_L = \mathbf{s_c}$, nous obtenons :

$$(\mathbf{I} - \mathbf{A}^L)\mathbf{s_c} = \sum_{i=1}^{L} \mathbf{A}^{L-1} \cdot \mathbf{w}_{i-1}. \tag{1.58}$$

où la matrice d'identité $\mathbf{I}$ a pour taille $(v-1 \times v-1)$. Ainsi, en introduisant l'état $\mathbf{s}_L^0$ après initialisation à 0, l'état circulaire $\mathbf{s_c}$ est égal à :

$$\mathbf{s}_c = (\mathbf{I} - \mathbf{A}^L)^{-1}\mathbf{s}_L^0. \tag{1.59}$$

L'équation est valide si et seulement si la matrice $(\mathbf{I} - \mathbf{A}^L)$ est *inversible*.

En pratique, la recherche d'état circulaire se décompose en deux étapes :
- l'obtention de l'état de terminaison $\mathbf{s}_L^0$ après codage d'une trame de longueur L à partir de l'état initial 0 ;
- la détermination de l'état circulaire $\mathbf{s_c}$ à partir de l'état $\mathbf{s}_L^0$. Les états $\mathbf{s_c}$ sont préalablement calculés et stockés dans une table d'après $(\mathbf{I} - \mathbf{A}^L)$.

Pour des séquences du type non-RTZ, le poids de Hamming de la parité est très grand. Le codage du code circulaire ne contribue donc pas à la DMH du code considéré. Mais, aucun bit supplémentaire n'est ajouté. Le rendement de codage n'est donc pas diminué. Comme tous les bits sont protégés de la même manière, il n'existe pas de l'effet de bord pour les codes circulaires.

1.5.3 Algorithme de turbo-décodage

1.5.3.1 Principe

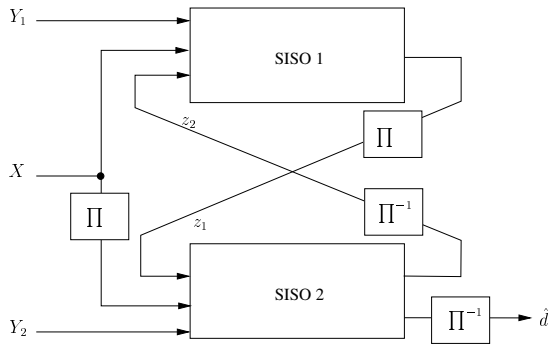


Figure 1.14 — Schéma de principe d'un turbo-décodeur

L'algorithme MAP effectué sur un code symétrique fournit une bonne estimation du LRV (Logarithme de Rapport de Vraisemblance) relatif à la donnée d_k codé à l'instant k. Cette estimation peut être considérée comme la somme deux sources. La première, l'information intrinsèque, est issue du canal de transmission, directement liée à la donnée d_k. Cette information intrinsèque est généralement disponible avant le décodage. La seconde, l'information extrinsèque, est apportée par le décodage du lien entre la donnée d_k et les autres symboles du mot de code (convolution, parité etc).

Au cours du décodage itératif, l'information intrinsèque est déjà exploitée par chacun des processeurs. Alors, elle ne doit pas être utilisée une deuxième fois comme information nouvelle sous peine de générer un effet d'auto-confirmation. Cette dernière peut provoquer l'instabilité du système. Ainsi, c'est l'information extrinsèque, sous forme de la probabilité ou du LRV, doit être échangée. Grâce à cet échange, le système converge avec une grande probabilité vers le mot de code le plus proche à partir du mot reçu.

Grâce à l'énergie apportée par la partie redondante, le rôle du décodeur SISO est d'essayer d'augmenter le rapport signal à bruit en assimilant l'information extrinsèque fournie par le

décodeur concurrent. Symboliquement, le LRV peut s'écrire :

$$LRV_{\text{sortie}}(d) = LRV_{\text{entrée}}(d) + z(d), \tag{1.60}$$

où $z(d)$ est l'information extrinsèque relative à d. Le LRV sortant est amélioré si l'information z est négative si $d = 0$, ou positive si $d = 1$. Ainsi, l'information extrinsèque $z(d)$ peut être extraite du LRV sortant en soustrayant le LRV entrant.

La figure 1.14 représente le schéma de principe d'un turbo-décodeur. Il est composé de deux décodeurs élémentaires SISO qui produisent respectivement l'information extrinsèque z_1 et z_2. L'entrelaceur et le désentrelaceur sont respectivement représentés par les symboles $\prod$ et $\prod^{-1}$. A partir du symbole reçu X, de l'information extrinsèque z_2 désentrelacée venant du décodeur DEC 2 et de la redondance correspondante Y_1, le décodeur DEC 1 fournit une probabilité de vraisemblance z_1 pour chaque élément décodé. Le décodeur, quant à lui, travaille conjointement à partir de l'information z_1 entrelacée, du symbole X entrelacé et de la redondance correspondante Y_2. Il produit sa propre information extrinsèque z_2. Ce processus d'échange répète N_{it} itérations. Une fois que les N_{it} itérations sont terminées, les informations extrinsèques du DEC 2 et les informations *a priori* sont sommées et une décision est prise sur la valeur du bit décodé.

Lorsque les deux décodeurs fournissent pour chaque bit considéré une probabilité similaire, le décodage a alors convergé. Dans ce cas, les itérations suivantes n'apportent pas de gain aux performances du système. Or, le décodage itératif est coûteux au niveau du temps et de la consommation de puissance. Il est alors nécessaire d'insérer un critère d'arrêt pour que le processus itératif se termine dès que le décodage converge. Ce sujet est abordé dans la section 1.6.3.

1.5.3.2 Performances du turbo-décodage

Les turbocodes forment une famille de codes correcteurs d'erreurs qui fait objet de nombreuses investigations depuis leur découverte. De plus, ils ont été adoptés par plusieurs normes comme UMTS [3GPP 99] et DVBTS. Le système UMTS est donc visé dans ce mémoire. La figure 1.15 donne les courbes des performances d'un turbocode binaire, tiré de la norme UMTS, au niveau de Taux d'Erreur par Paquet (TEP). Les performances sont obtenues avec une longueur de trame $L = 640$ bits et un rendement de code $R = 1/3$ sur canal ABBG. Le nombre d'itérations effectuées est égal à 6. L'algorithme Max-Log-MAP est utilisé pour le décodeur SISO (Soft Input Soft Output). Nous constatons une décroissance du Taux d'Erreurs par Paquet (TEP), tout près de la limite théorique de Shannon. D'ailleurs, la distance minimale du code n'est pas suffisante, $d_{min} = 26$ dans notre cas pour un rendement $R = 1/3$. C'est la raison pour laquelle le changement de pente est prononcé.

1.6 Problème de la consommation dans les systèmes de communications numériques

L'implémentation classique de l'algorithme MAX-Log-MAP nécessite une mémoire importante pour sauvegarder les métriques de nœud lors du premier parcours du treillis. Une étude dans [Schurgers 01] montre que la consommation de mémoire atteint jusqu'à 50% de la consommation globale dans un turbo-décodeur. Pour attaquer la problématique de consommation dans un turbo-décodeur, le modèle de consommation est présenté dans un premier

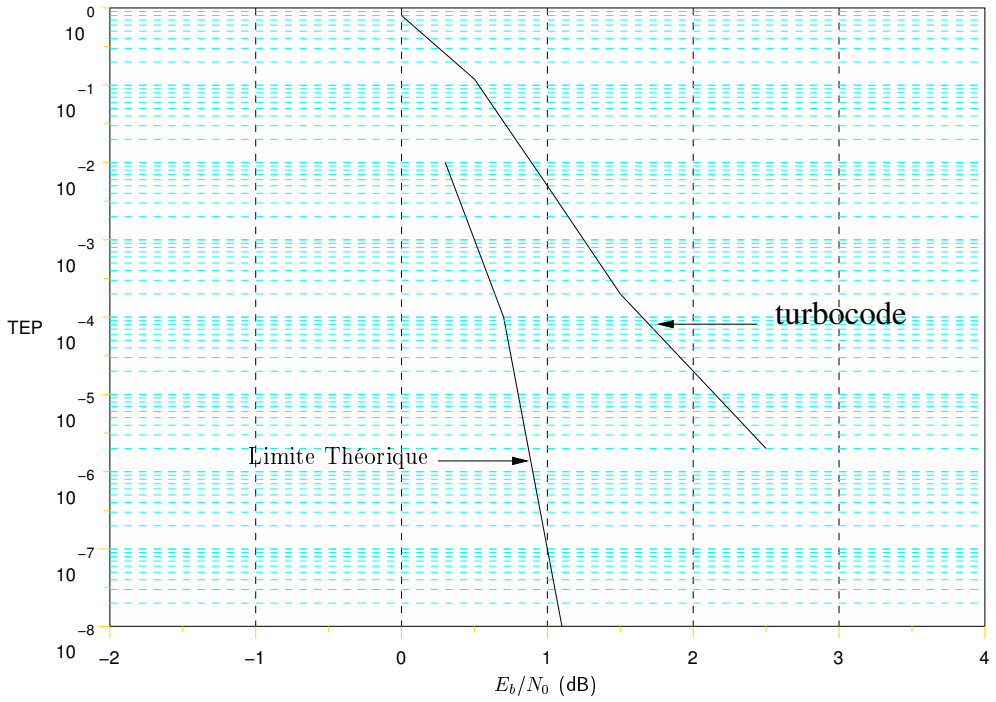


Figure 1.15 — Performances en Taux d'Erreurs par Parquet (TEP) du turbocode de la norme UMTS, extraites de l'ouvrage [Berrou 07].

temps dans la section 1.6.1. Ensuite, une investigation sur la maîtrise de consommation à l'aspect algorithmique est menée dans la section 1.6.2. De plus, les travaux de recherche sur l'aspect architectural sont synthétisés dans la section 1.6.3.

1.6.1 Modèle de consommation

Dans un système embarqué, la consommation du système est un critère primordial. La puissance dynamique P_{dyn} et la puissance statique (leakage en anglais) P_{lkg} sont les deux sources majeures de consommation. Elles peuvent s'exprimer par l'équation 1.61, où V_{DD}, f, α, C_L et I_{lkg} représentent respectivement la tension d'alimentation, la fréquence, le facteur d'activité, la capacité parasite et le courant statique. La consommation statique est proportionnelle à la surface du circuit. Traditionnellement, la puissance dynamique a été considérée comme une source dominante sur la consommation d'un système numérique. Cependant, en raison de l'évolution de la technologie, la densité de transistor dans un circuit a considérablement augmenté ces dernières années. Par conséquent, la puissance statique devient une source de consommation de plus en plus significative. L'ITRS (abrégé de International Technology Roadmap for Semiconductors) prévoit une consommation statique de 50% de la consommation globale dans les futurs circuits employant la technologie 45nm [ITRS]. Pour maîtriser la consommation du système, il est donc pertinent de considérer tous les éléments de l'expression 1.61 :

$$P_{dyn} = V_{DD}^2 \cdot f \cdot \alpha \cdot C_L, \quad P_{lkg} = V_{DD} \cdot I_{lkg} \tag{1.61}$$

Dans un système mobile de communications numériques, le budget énergétique d'un turbo-décodeur peut atteindre jusqu'à 50% de la consommation globale du récepteur dédié aux signaux mixtes [Bougard 06]. Cela veut dire que le décodage de canal est le problème majeur au niveau consommation. Pour réduire la consommation, il est possible de diminuer la sur-

face du circuit et baisser l'activité des mémoires. De nombreuses études algorithmiques et architecturales sont menées pour tenter d'apporter des solutions à ce problème.

1.6.2 Etat de l'art sur la maîtrise de consommation d'un turbo-décodeur : aspect algorithmique

La puissance dynamique prend effet lors des lectures et des écritures de mémoire, tandis que la puissance statique est proportionnelle à la taille de mémoire et accroît de façon exponentielle selon la technologie employée. Cette constatation est contraignante pour l'aspect consommation car il a été montré [Schurgers 01] que les accès mémoires jouent un rôle essentiel sur la consommation d'un circuit turbo-décodage. C'est pourquoi des études algorithmiques sont menées pour tenter d'apporter des solutions à cette contrainte.

Parmi elles, le calcul inverse des métriques de nœud proposé initialement dans [Wu 00] est détaillé dans cette section. Son objectif est d'éliminer au maximum la mémorisation des métriques de nœud lors du premier parcours du treillis. Dans ce cas, le calcul du LRV ne nécessite pas la récupération d'une métrique de nœud en mémoire. Choi.H montre que la taille de la mémoire peut diminuer jusqu'à 50% [Choi 06]. La consommation du système est alors réduite d'environ 35% [Atluri 03][Choi 06] par rapport à l'algorithme classique.

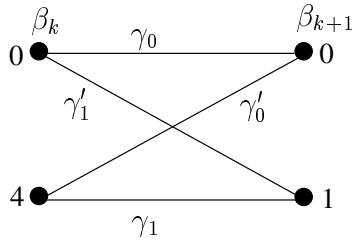


Figure 1.16 — Papillon du treillis associé à un codeur convolutif $G[13, 15]_0$ du système UMTS. Les chiffres 0,1,4 représentent l'état considéré.

Pour un papillon de treillis donné(figure 1.16), il existe quatre métriques de branche, notées respectivement γ_0, γ_1, γ'_0 et γ'_1 si le code CCRS est bien défini [Wu 00]. Considérons le treillis à l'instant k durant un parcours retour du treillis, le calcul des métriques de nœud est :

$$\begin{aligned} \beta_k(0) &= -\log(\mathrm{e}^{-(\beta_{k+1}(0)+\gamma_0)} + \mathrm{e}^{-(\beta_{k+1}(1)+\gamma'_1)}) \\ \beta_k(4) &= -\log(\mathrm{e}^{-(\beta_k(0)+\gamma'_0)} + \mathrm{e}^{(-\beta_{k+1}(1)+\gamma_1)}) \end{aligned} \tag{1.62}$$

Le calcul inverse des métriques de nœud permet d'obtenir d'après les explications données dans [Atluri 03])[Lee 05] :

$$\begin{aligned} \beta_{k+1}(0) &= -\log(\mathrm{e}^{-(\beta_k(0)+\gamma_0)} - \mathrm{e}^{(\beta_k(4)+\gamma'_1)}) - \log|C| \\ \beta_{k+1}(1) &= -\log(\mathrm{e}^{-(\beta_k(4)+\gamma_1)} - \mathrm{e}^{(\beta_k(0)+\gamma'_0)}) - \log|C| \end{aligned} \tag{1.63}$$

où la constante $\log|C|$ est égale à $-\log(\mathrm{e}^{(\gamma_0+\gamma_1)} + \mathrm{e}^{(\gamma'_0+\gamma'_1)})$. L'expression 1.63 montre que les métriques de nœud β_{k+1} à l'instant $k+1$ sont calculées à partir des métriques de nœud β_k à l'instant k.

Sachant que

$$\log(|\mathrm{e}^x - \mathrm{e}^y|) = \min(x, y) + \log(\mathrm{e}^{|x-y|} - 1), \tag{1.64}$$

le terme $\log(e^{|x-y|}-1)$ est approximativement égal à :

$$\log(e^{|x-y|}-1) \simeq |x-y| \quad \text{si } |x-y| \gg 0. \tag{1.65}$$

Ainsi, il est encore possible de simplifier l'expression 1.63 si les conditions sont satisfaites pour 1.65 :

$$\begin{array}{l} \beta_{k+1}(0) = \max(\beta_k(0)+\gamma_0, \beta_k(4)+\gamma'_1) + \max(\gamma_0+\gamma_1, \gamma'_0+\gamma'_1) + C' \\ \beta_{k+1}(1) = \max(\beta_k(4)+\gamma_1, \beta_k(0)+\gamma'_0) + \max(\gamma_0+\gamma_1, \gamma'_0+\gamma'_1) + C', \end{array} \tag{1.66}$$

où la constante C' dépend des valeurs $\beta_k(0)$, $\beta_k(4)$, γ_0, γ_1, γ'_0 et γ'_1. Pour calculer les autres métriques de nœud, l'expression 1.66 est aussi vraie en prenant les métriques correspondantes de β_k, γ et γ' sur le papillon considéré.

Similaire au calcul des métriques α_k à partir de l'expression 1.39, les métriques de noeud β peuvent être calculées de manière récursive durant le second parcours du treillis (sens *Aller*) au sein du décodage. Ainsi, les calculs des métriques α_k et β_k dans les expressions 1.42 et 1.44 peuvent être effectués en parallèle. Dès lors, si la condition 1.65 est toujours vraie, il n'est pas nécessaire de stocker les métriques de noeud β_k pour calculer le rapport LRV durant le premier parcours du treillis (sens *Retour*). De plus, en supposant que les données sont traitées simultanément en sous-blocs, la solution basée sur l'expression 1.66 donne une bonne réduction de mémoire [Atluri 03] puisque les métriques de noeud ne sont stockées qu'au début de chaque sous-bloc. La consommation est par conséquent réduite grâce à une diminution significative de la taille des mémoires. Cependant, ce gain est obtenu au prix d'une dégradation des performances comme indiquée dans [Atluri 03].

Pour remédier à ce problème, une autre approche basée sur l'expression 1.64, proposée dans [Lee 05], consiste à limiter le nombre des métriques de noeud mémorisées au lieu de diminuer la taille des mémoires. Cette approche prend en compte la condition 1.65 à chaque étape. Lorsque la condition n'est pas satisfaite, la métrique de nœud β est stockée en attendant la métrique α_k correspondante pour calculer le rapport LRV. Au cas contraire, les métriques β ne sont pas stockées et le processus continue. Or, Il est impossible de déterminer préalablement les positions où la métrique β doit être stockée. Dans le cas pire, il peut arriver que toutes les métriques β_k doivent être stockées. Il est donc nécessaire d'avoir la même taille des mémoires que l'algorithme classique. Dans ce cas, son intérêt se situe au niveau des accès au stockage, qui sont considérés comme un aspect critique au niveau de la consommation. Des résultats expérimentaux dans [Lee 05] montrent que la diminution du nombre des accès peut atteindre 90% des cas par rapport à l'algorithme Max-Log-MAP classique.

De plus, une solution permettant à la fois de diminuer la taille des mémoires et le nombre des accès mémoires, a été proposée récemment dans [Choi 06]. D'après la relation 1.41, nous obtenons au premier parcours du treillis (sens *Retour*) (figure 1.16) :

$$\begin{cases} \beta_k(0) = \min(\beta_{k+1}(1)+\gamma'_1, \beta_{k+1}(0)+\gamma_0) \\ \beta_k(4) = \min(\beta_{k+1}(0)+\gamma'_0, \beta_{k+1}(1)+\gamma_1) \end{cases} \tag{1.67}$$

Nous faisons une comparaison entre les termes $(\beta_k(0)-\gamma_0)$ et $(\beta_k(4)-\gamma'_0)$. En cas d'égalité, la valeur $\beta_{k+1}(1)$ qui n'a jamais gagné l'expression 1.67 est stockée dans la mémoire. Dans ce cas, sur le sens *Retour*, nous avons :

$$\begin{cases} \beta_{k+1}(0) = \beta_k(0) - \gamma_0 \\ \beta_{k+1}(1) = \beta_{k+1}(1) \qquad \text{(stocké)} \end{cases} \tag{1.68}$$

De plus, une autre comparaison est procédée en parallèle entre les termes $(\beta_k(0) - \gamma'_1)$ et $(\beta_k(4) - \gamma_1)$. Si ces deux termes sont égaux, nous obtenons :

$$\begin{cases} \beta_{k+1}(1) = \beta_k(0) - \gamma'_1 \\ \beta_{k+1}(0) = \beta_{k+1}(0) \qquad \text{(stocké)} \end{cases} \tag{1.69}$$

Sinon, les métriques β_{k+1} s'expriment par :

$$\begin{cases} \beta_{k+1}(0) = \max(\beta_k(0) - \gamma_0, \beta_k(4) - \gamma'_0) \\ \beta_{k+1}(4) = \max(\beta_k(0) - \gamma'_1, \beta_k(4) - \gamma_1) \end{cases} \tag{1.70}$$

L'expression 1.70 montre que les valeurs β_{k+1} peuvent être calculées à partir des valeurs β_k. Ainsi, il n'est pas nécessaire de stocker toutes les valeurs de β_{k+1}. Au pire cas, exprimé par les relations 1.68 et 1.69, la moitié des valeurs β_{k+1} est stockée à chaque étape, d'où la taille des mémoires est réduite de 50%. De plus, la diminution des accès mémoires peut atteindre 80% selon les expérimentations menées dans [Choi 06].

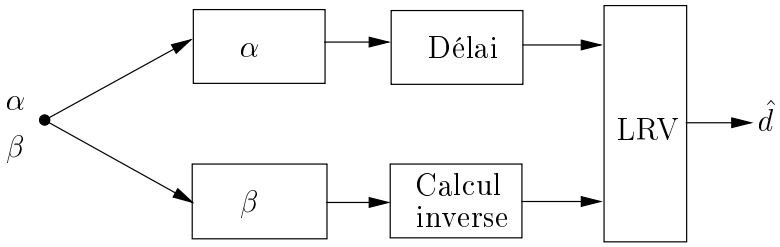


Figure 1.17 — Une architecture en bloc associée à l'algorithme Max-Log-MAP appliquant le calcul inverse des métriques de nœud

La figure 1.17 illustre une architecture en bloc associée à l'algorithme Max-Log-MAP appliquant le calcul inverse des métriques de nœud β. Nous supposons que le décodage se déroule d'abord dans le sens *Retour*, puis dans le sens *Aller* (figure 1.17). Leurs métriques α et β sont respectivement calculées dans les blocs correspondants. Au niveau du calcul LRV, au lieu d'accéder directement aux mémoires lors du second parcours du treillis, ce qui est généralement demandé dans un système classique de l'algorithme Max-Log-MAP, les opérations nécessaires sur le calcul inverse sont réalisées dans le bloc *calcul inverse*. De ce fait, la complexité du système est augmentée. C'est la raison pour laquelle ces opérations supplémentaires peuvent avoir l'impact sur la consommation globale d'un système. De plus, un délai temporel permet de synchroniser les données à l'entrée du bloc « LRV » fournissant la fiabilité du symbole décodé.

1.6.3 Etat de l'art sur la maîtrise de consommation d'un turbo-décodeur : aspect architectural

La standardisation des turbocodes dans des applications mobiles et l'émergence des systèmes embarqués nécessitent le développement de solutions architecturales ayant une faible consommation. Dans cette thèse, nous nous focalisons sur le problème de la consommation des architectures de turbo-décodage. Deux aspects doivent alors être considérés au niveau consommation : la partie traitement et la partie mémorisation. Cette dernière joue désormais un rôle primordial.

1.6.3.1 Optimisations architecturales au niveau des traitements

Au niveau du turbo-décodeur, des études concernent l'insertion d'un critère d'arrêt et la définition d'entrelaceurs performants. Le principe du critère d'arrêt [Hagenauer 96] est lié à l'aspect itératif de l'algorithme de turbo-décodage. En effet, un nombre a priori d'itérations est fixé pour les architectures de turbo-décodeurs. Ce nombre est lié au contexte d'utilisation du code correcteur d'erreurs. Cependant, l'ensemble des données traitées par le turbo-décodeur n'est pas affecté de la même manière. En effet, un grand nombre d'erreurs sont corrigées lors des premières itérations. Dans un souci de maîtrise de la consommation, il apparaît donc pertinent d'insérer un critère d'arrêt à l'architecture de turbo-décodage. Son rôle est d'ajuster le nombre d'itérations adéquat au cours du traitement des données codées et bruitées par le décodeur.

De même, la définition des entrelaceurs a un impact sur la complexité de la solution architecturale [Garrett 01]. Par exemple, le standard de turbo-décodage retenu pour l'UMTS impose la génération d'adresses complexes pour l'entrelacement entre les deux décodeurs élémentaires. Le plus simple est de réaliser une table d'adressage contenant toutes les adresses possibles. Cependant, une meilleure solution, du point de vue de la consommation, est de construire un générateur d'adresses. De plus, une structure bien conçue et adaptée d'entrelacement et de désentrelacement prenant en compte les contraintes matérielles apporte à coup sûr des gains significatifs [Masera 99] [Elassal 03] [Gnaëdig 03].

Au niveau du décodeur élémentaire SISO, des études traitent de la quantification de l'information reçue via le canal, de l'information extrinsèque et des métriques de nœud. Il est ainsi possible de borner les valeurs des informations extrinsèques sans apporter de dégradation en terme de performance comme le montre [Garrett 01]. De même, la précision des métriques de nœud qui découle de celle des informations reçues par le décodeur élémentaire a un impact direct sur la complexité des chemins de données du décodeur [Boutillon 03b]. Il est également important de noter l'importance de la quantification sur la taille de la mémoire et par conséquent sur la taille des transferts de données. Enfin, un soin particulier doit être apporté aux composants combinatoires de base. Ainsi, pour l'algorithme Max-Log-MAP, le composant ACS (Add Compare Select) est fondamental [Mansour 03].

1.6.3.2 Optimisations architecturales au niveau de la mémorisation

Comme nous l'avons précédemment indiqué, la gestion de la mémoire est fondamentale pour la maîtrise de la consommation des architectures dédiées au turbo-décodage. En effet, des études expérimentales ont montré que pour l'algorithme Max-Log-MAP, la consommation d'énergie est répartie de la manière suivante : 72% pour le stockage des métriques cumulées et 20% pour le stockage des données et des informations extrinsèques [Schurgers 01]. La minimisation de la mémoire nécessaire à cet effet passe tout d'abord par la réduction de la fenêtre de traitement sur laquelle est réalisé le décodage. Dès 1996, le principe de la fenêtre glissante a été introduit dans [Benedetto 97]. Cette méthode consiste à remplacer le premier traitement complet du treillis par une succession de traitements partiels. Dans ce cas, les performances sont similaires si un prologue servant à estimer les valeurs initiales des métriques de nœud est effectué pour chaque traitement partiel. Enfin, le choix de la longueur de la fenêtre glissante a un impact non seulement sur la taille de la mémoire mais aussi sur la complexité combinatoire comme le montre [Dingninou 01].

La mémorisation des métriques de nœud, proprement dit, peut être optimisée en appliquant le calcul inverse explicité dans la première partie. Le principal intérêt est alors de

sauvegarder partiellement les métriques de nœud lors du premier parcours du treillis au cours du décodage. Ce procédé permet d'éviter une grande partie des accès mémoires qui sont une source de consommation importante comme nous l'avons préalablement indiqué. D'ailleurs, il est possible de mémoriser les métriques de branche lors du premier parcours du treillis, comme proposé dans les travaux de [Tsai 05] et de [Lopez-Vallejo 02]. Cette approche permet de supprimer les accès aux mémoires principales des données systématiques et des informations extrinsèques lors du second parcours du treillis. En effet, le calcul des métriques de branche est le résultat de combinaison $\pm(x \times u) \pm (y \times v)$ où les symboles x et y correspondent à l'information reçue, et u et v les mots associés du treillis. C'est pourquoi il est possible de sauvegarder uniquement les deux résultats des polynômes $|x \times u| + (y \times v)$ et $|x \times u| - (y \times v)$ pour les métriques de branche quelque soit le nombre d'états choisis. En contrepartie, cette solution nécessite d'avoir une mémoire dédiée à la sauvegarde des métriques de branche. La taille de cette mémoire peut être importante selon la longueur de la trame, d'où une source non négligeable de la consommation.

En pratique, les métriques de branche sont recalculées lors du second parcours du treillis. Dans ce cas, nous n'avons pas besoin de stocker les métriques de branche. D'un point de vue consommation, la solution sans mémoriser les métriques de branche est alors retenue dans la suite de la thèse. Enfin, pour mieux gérer les conflits des accès mémoires, une solution consiste à rassembler plusieurs données dans un seul mot afin de diminuer le nombre total des accès à la mémoire. Mais un accès mémoire d'un long mot consomme plus d'énergie qu'un accès pour un mot court. Le bilan n'entraîne donc pas forcement une diminution de la consommation globale. Cependant, une expérimentation proposée dans [Maessen 00] montre une réduction de la consommation globale par un facteur 2.

1.7 Conclusion

Dans ce premier chapitre, des notions de base sur les transmissions numériques sont introduites. Après une description succincte d'une chaîne de transmissions numériques, ses différents composants sont évoqués. Pour le canal de transmission, nous considérons uniquement le canal symétrique et le canal ABBG. Leurs propriétés sont également discutées. A partir de la base théorique de l'information, la nécessité de redondance est justifiée pour lutter contre les bruits de transmission. Pour mieux protéger les informations à transmettre, une méthode efficace, connue sous le nom *codage de canal*, consiste à ajouter de la redondance aux informations en formant un mot de code.

Nous nous intéressons particulièrement aux codes convolutifs et plus précisément aux turbocodes. Les diverses représentations des codes convolutifs sont exposées dans ce chapitre. Leurs capacités de correction sont également discutées. Au niveau du décodage, les algorithmes de Viterbi et MAP sont présentés. L'implémentation de l'algorithme de Viterbi n'est pas complexe. Mais ses performances du décodage sont moins bonnes que celles de l'algorithme MAP. Quant à l'algorithme MAP, il représente une solution optimale du décodage au prix d'une complexité importante dû à ses opérations arithmétiques. En prenant un compromis entre les performances et la complexité, une version simplifiée permettant notamment de diminuer la complexité matérielle est donnée sous le nom de *Max-Log-MAP*. Puis, le principe des turbocodes à concaténation parallèle de deux codeurs convolutifs est évoqué. Nous avons montré leurs performances très proches de la limite théorique. Mais ces performances sont atteintes au prix d'un décodage itératif. Ceci signifie que le turbo-décodage implique des contraintes temporelles (débit) et des contraintes matérielles en particulier au niveau de la

consommation.

En ce qui concerne la consommation d'un système de communications numériques, ses caractéristiques et sa tendance sont dans un premier temps introduites. En particulier pour le turbo-décodeur, la consommation de mémoire atteint jusqu'à 50% de consommation globale, ce qui nous pousse de tenter de proposer des solutions architecturales pour diminuer à la fois le nombre d'accès mémoire (consommation dynamique) et la taille de mémoire (consommation statique).

Deux constats sont très prometteurs. Le turbo-décodage est très robuste contre les erreurs résiduelles grâce à sa propriété itérative. Il est donc envisageable de diminuer la quantification des données pour réduire la taille des mémoires, sans dégrader les performances de manière significative. De plus, Pour un processus itératif, le nombre des erreurs corrigées est généralement très limité sur les dernières itérations. Si le décodage ne dépend que des vecteurs des erreurs, l'activité du turbo-décodeur, définie par le nombre des transitions entre les valeurs binaires 0 et 1, peut être considérablement diminuée lorsque peu d'erreurs sont présentes. C'est ainsi que le turbo-décodage différentiel permet de distinguer les vecteurs d'erreurs de transmission à partir des symboles reçus bruités avant de procéder au *vrai* décodage. Par conséquent, le cœur du décodage fonctionne sur les vecteurs d'erreurs. Dans le prochain chapitre, nous présentons le principe du décodage différentiel ainsi que ses performances.

CHAPITRE

2 Algorithme de Turbo-Décodage Différentiel avec Insertion d'Erreurs

TRADITIONNELLEMENT , l'étape de turbo-décodage s'applique directement sur les mots de codes bruités. Pour maîtriser la consommation d'un système dans une chaîne de communications numériques, une méthode efficace consiste à diminuer son activité. Dans ce chapitre, nous détaillons un système différentiel de turbo-décodage, qui s'appuie sur les vecteurs obtenus à partir des erreurs de transmission. Ce système, appelé *système différentiel*, permet de retirer les vecteurs d'erreur à partir du message reçu avant de procéder au décodage proprement dit. Il est ainsi possible de stabiliser le chemin survivant sur le chemin TAZ (Tout A Zéro) lors du parcours d'un treillis. Dans ce cas, le système différentiel a une activité moins importante qu'un système classique lorsque la quantité des erreurs n'est pas significative. Ainsi, lors des dernières itérations de l'algorithme de turbo-décodage, une diminution significative de l'activité du décodeur peut être obtenue.

Dans la section 2.1.2, nous décrivons le principe général du codage différentiel. Pour ce faire, nous définissons le taux d'occupation sur le chemin TAZ comme une métrique de mesure de l'activité du décodage différentiel. Dans ce chapitre, nous définissons l'activité **a** du système par le nombre des transitions binaires entre 0 et 1 par l'unité de temps sur les métriques de nœud sauvegardées. Dans un premier temps, le codage différentiel est appliqué à un décodeur de Viterbi pour un canal symétrique. Le taux d'occupation sur le chemin TAZ et les performances du décodeur en terme de Taux d'Erreur Binaire (TEB) sont ensuite analysés dans la section 2.1.2.3. Dans un second temps, l'extension du codage différentiel à l'algorithme Max-Log-MAP est développée dans la section 2.1.3. Les performances d'un turbo-décodeur appliquant le codage différentiel sont en particulier présentées dans la section 2.1.3.2.

Cette étude démontre que lors du parcours d'un treillis pour un décodage différentiel, des motifs périodiques peuvent apparaître pour le chemin survivant plutôt qu'une stabilisation sur le chemin TAZ. Ce motif est néfaste au niveau de diminution d'activité puisqu'il diverge le chemin survivant du chemin TAZ lors du parcours d'un treillis. Pour limiter cet effet, une méthode d'insertion de dummy-errors est proposée dans la section 2.2. Cette méthode permet d'éliminer ces motifs répétitifs en insérant des erreurs supplémentaires au cours du codage différentiel. Grâce à cette insertion, le taux d'occupation sur le chemin TAZ est augmenté de manière significative lorsque le nombre d'erreurs est limité. Enfin, un calcul anticipé de l'information extrinsèque est mis en œuvre pour limiter la mémorisation des métriques de nœud et ainsi diminuer l'activité du turbo-décodeur (section 2.2.4). En effet, contrairement

à l'algorithme Max-Log-MAP, l'information extrinsèque est directement obtenue lors du parcours dans un sens sans connaître les valeurs des métriques de nœud dans l'autre sens. Cette approche modifie néanmoins les valeurs des informations extrinsèques. Les informations mutuelles et la corrélation entre l'information extrinsèque calculée pour une approche traditionnelle et l'information extrinsèque anticipée de notre approche sont détaillées respectivement dans les sections 2.3.5 et 2.3.7.

2.1 Algorithme de turbo-décodage différentiel

2.1.1 Introduction

Nous distinguons deux types de consommation dans un système électronique de communications numériques : consommation statique et consommation dynamique. Pour diminuer la consommation globale du décodeur, il est alors possible de diminuer sa consommation dynamique. Cette dernière dépend essentiellement de la fréquence de travail, de la tension d'alimentation et de l'activité. Dans la thèse, l'activité du décodeur est définie par le nombre de transitions binaires par l'unité de temps, en particulier sur la variation des métriques de nœud au cours du décodage. Au niveau de la consommation dynamique, il est par exemple possible d'ajuster la fréquence et la tension d'alimentation au cours du décodage de telle manière à adapter dynamiquement les caractéristiques du récepteur sur le débit souhaité, la longueur de trame et la qualité de transmission [Gilbert 01] [Leung 99]. A ce propos, nous proposons dans ce chapitre une nouvelle approche permettant de diminuer l'activité du décodeur.

Dans un cadre classique, l'information à transmettre suit une loi de distribution équiprobable. Après transmission d'un canal, l'information reçue est modélisée par l'expression suivante :

$$\mathbf{X} = \mathbf{x} + \mathbf{e} \tag{2.1}$$

où les éléments des vecteurs $\mathbf{X}$, $\mathbf{x}$ et $\mathbf{e}$ représentent respectivement l'information bruitée, l'information émise et le bruit associé. L'opération + est considérée comme une addition modulo à 2. Classiquement, le rôle du décodeur SISO est de retrouver le chemin le plus probable, dit *chemin survivant*, associé au vecteur $\mathbf{x}$ à partir de l'information reçue $\mathbf{X}$ dans le treillis correspondant. Comme les données transmises sont équiprobables, les chemins survivants suivent aussi une répartition équiprobable parmi les $2^{K+L+v-1}$ chemins possibles, où les symboles K, L et v représentent respectivement la taille du bloc à l'entrée du codeur, la longueur de trame et la longueur de contrainte du codeur.

Supposons que le vecteur transmis n'est composé que des éléments 0, dit *vecteur TAZ* (Tout A Zéro). Dans ce cas, seules les erreurs sont prises en compte par le décodeur. Lorsqu'il n'y a aucune erreur de transmission, le chemin survivant du décodage poursuit le chemin TAZ avec une probabilité maximale. Dans ce cas, nous disposons d'une éventuelle faible activité du décodage, puisque le chemin survivant ne diverge jamais du chemin TAZ au cours du décodage. Nous pouvons alors poser une question : est-il toujours vrai que le décodeur dispose une basse activité lors qu'il ne travaille que sur les vecteurs d'erreurs ? De plus, il est possible de connaître l'information émise à partir de l'information bruitée si le vecteur d'erreur est connu. Nous avons alors :

$$\mathbf{X} + \mathbf{e} = (\mathbf{x} + \mathbf{e}) + \mathbf{e} = \mathbf{x} \tag{2.2}$$

Le résultat du décodage s'appuie alors sur le vecteur d'erreurs $\mathbf{e}$ dues à la transmission.

En général, la répartition des chemins survivants dépend uniquement du rapport SNR si le vecteur d'erreurs est considéré pour le décodage. Le principe des turbocodes revient à traiter les données entrantes de manière à augmenter progressivement le rapport signal à bruit au cours d'un processus itératif. La plupart des erreurs sont corrigées durant les premières itérations, ce qui correspond au cas d'un faible SNR. Lors des dernières itérations, ce qui correspond au cas d'un fort SNR, les données à l'entrée du décodeur ne sont que peu bruitées par le canal. Suivant le même principe étendu aux turbocodes, la probabilité sur la valeur 0 est plus grande que celle sur la valeur 1 dans le chemin survivant lorsque le décodage sur le vecteur d'erreurs est considéré. Par conséquent, il y a moins de transitions entre les valeurs 0 et 1 sur

le chemin survivant lors du parcours du treillis. Cette propriété rend le décodeur moins actif par rapport à un décodeur classique, d'où une diminution potentielle de la consommation.

Dans le cas général, le vecteur $\mathbf{x}$ est bien sûr différent du vecteur TAZ. Nous proposons, dans cette thèse, d'effectuer un ré-encodage à partir de la séquence reçue avant de procéder au décodage proprement dit. Cette technique de ré-encodage a initialement été proposée dans [Welch 86] pour les codes en bloc. Puis, elle a été étendue aux codes Reed Solomon dans [Berlekamp 96]. Enfin, elle a été appliquée au décodage souple des codes RS [Gross 02]. Parallèlement, S. Kubota [Kubota 93] a intégré cette technique sous le nom *SST* (Scarce State Transition en anglais) pour permettre une diminution de la consommation dans des décodeurs dédiés aux codes convolutifs. L'algorithme de Viterbi est alors l'algorithme de décodage utilisé. Grâce à ce procédé, le système peut soustraire le vecteur d'erreurs de la séquence reçue bruitée, d'où l'appellation, *système différentiel*. Le décodeur différentiel travaille alors sur le vecteur d'erreurs et non plus le message reçu. Cette approche s'est avérée compatible avec des architectures de type RE (Registre d'Echange) [Kubota 93] et également avec l'algorithme T [Jin 07] [Lin 07], qui vise à diminuer la complexité du module de remontée à travers le treillis.

Y. Wang [Wang 00] a ensuite appliqué cette approche à un turbo-décodeur basé sur l'algorithme SOVA (Soft Output Viterbi Algorithm en anglais) [Hagenauer 89]. Similaire au décodeur de Viterbi, lorsque la valeur du SNR est élevée, la probabilité sur la valeur 0 est maximisée par rapport à la probabilité sur la valeur 1 dans le chemin survivant. Ainsi, le nombre des transitions entre 0 et 1 est diminué et l'activité du décodeur est ainsi réduite. Cependant, l'algorithme SOVA est une technique de décodage souple qui s'avère moins efficace en terme de performance que l'algorithme Max-Log-MAP pour les turbocodes. C'est pourquoi, dans ce mémoire, nous étudions cette approche pour l'algorithme Max-Log-MAP, qui fournit une information souple sur la probabilité du bit considéré. Contrairement à l'algorithme de Viterbi qui mémorise le chemin survivant, les métriques de nœud α sont mémorisées lors du premier parcours du treillis sur l'algorithme Max-Log-MAP.

Ce chapitre est écrit de la manière suivante. Nous présentons le principe du codage différentiel dans la section 2.1.2. Ensuite, nous appliquons cette technique à l'algorithme MAX-Log-MAP dans la section 2.1.3. L'activité d'un tubo-décodeur du système UMTS utilisant le codage différentiel est étudiée dans la section 2.1.4. De plus, nous proposons la méthode *insertion de dummy-errors* dans la section 2.2 pour éliminer les motifs répétitifs dus au codage différentiel. Enfin, en vue de diminuer l'accès mémoire, nous proposons une approche permettant de calculer l'information extrinsèque de façon anticipée dans la section 2.3.

2.1.2 Principe du codage différentiel

Comme évoqué précédemment, l'objectif du codage différentiel consiste à soustraire les vecteurs d'erreurs du message reçu avant d'effectuer le processus décodage. Dans un premier temps, nous expliquons le principe du CD (Codage Différentiel) pour l'algorithme de Viterbi avec les décisions fermes. Ensuite, l'application de cette technique est étendue à l'algorithme Max-Log-MAP.

2.1.2.1 Codage différentiel

Pour expliquer la construction de ce type de code, revenons aux équations fondamentales qui régissent le fonctionnement d'un codeur. Considérons que les vecteurs $\mathbf{w}_k^N$ et $\mathbf{u}_k^N$ sont composés des N éléments binaires et chaque élément est une combinaison arithmétique des

données entrantes du codeur. Le vecteur $\mathbf{v}_k^N$ est composé des N éléments binaires sortants du codeur. Il est possible d'établir une relation entre l'état $\mathbf{s}_{k+1}$ du codeur à l'instant $k+1$, son état $\mathbf{s}_k$ et le vecteur $\mathbf{w}_k^N$ à l'instant k :

$$\mathbf{s}_{k+1} = \mathbf{A} \cdot \mathbf{s}_k + \mathbf{w}_k^N \tag{2.3}$$

De même, une relation peut être établie entre les données sortantes du codeur $\mathbf{v}_k^N$, son état $\mathbf{s}_k$ et le vecteur $\mathbf{u}_k^N$ à l'instant k :

$$\mathbf{v}_k^N = \mathbf{C} \cdot \mathbf{s}_k + \mathbf{u}_k^N \tag{2.4}$$

Ces relations correspondent à une représentation dans l'espace d'états [Weiss 98].

Il est à noter que ces opérations matricielles sont basées sur les opérations arithmétiques modulo à 2. Pour illustrer notre propos, nous considérons un cas d'école à savoir le code systématique récursif de polynômes générateurs : $G[1, (1+D^2)/(1+D+D^2)]$. Ainsi, le nombre de sortie N est égal à 2. Son schéma de principe est présenté par la figure 2.1.

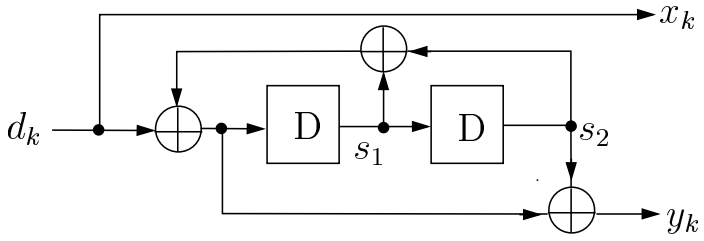


Figure 2.1 — Codeur convolutif systématique récursif ayant pour polynôme générateur $G[1, (1+D^2)/(1+D+D^2)]$

Les matrices ont alors pour valeur :

$$\mathbf{A} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \text{ et } \mathbf{C} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}$$

Dans la suite de cet exemple, nous mettons l'indice N égal à 2 pour alléger les notations. Les vecteurs entrants $\mathbf{u}$ et $\mathbf{w}$ du codeur, quant à eux, ont pour valeur :

$$\mathbf{u}_k = \begin{pmatrix} d_k \\ d_k \end{pmatrix} \quad \text{et } \mathbf{w}_k = \begin{pmatrix} d_k \\ 0 \end{pmatrix}$$

d'où

$$\mathbf{s}_{k+1} = \begin{pmatrix} s_{1,k+1} \\ s_{2,k+1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} s_{1,k} \\ s_{2,k} \end{pmatrix} + \begin{pmatrix} d_k \\ 0 \end{pmatrix} \tag{2.5}$$

et

$$\mathbf{v}_k = \begin{pmatrix} x_k \\ y_k \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} s_{1,k} \\ s_{2,k} \end{pmatrix} + \begin{pmatrix} d_k \\ d_k \end{pmatrix} \tag{2.6}$$

Appliquons ce formalisme à la technique de ré-encodage. Tout d'abord, le vecteur sortant du codeur est bruité par un vecteur d'erreurs correspondant $\mathbf{E}_k^N = \begin{pmatrix} E_{x,k} \\ E_{y,k} \end{pmatrix}$ dont les éléments $E_{x,k}$ et $E_{y,k}$ correspondent respectivement aux erreurs associées aux sorties x_k et

y_k du codeur :

$$\mathbf{V}_k^N = \mathbf{v}_k^N + \mathbf{E}_k^N \tag{2.7}$$

Nous pouvons développer l'expression 2.7 comme suit :

$$\mathbf{V}_k = \begin{pmatrix} X_k \\ Y_k \end{pmatrix} = \begin{pmatrix} x_k \\ y_k \end{pmatrix} + \begin{pmatrix} E_{x,k} \\ E_{y,k} \end{pmatrix} \tag{2.8}$$

Puis une étape de ré-encodage est appliquée à la composante systématique X_k de $\mathbf{V}_k$. En considérant

$$\mathbf{U}_k = \begin{pmatrix} X_k \\ X_k \end{pmatrix} \text{ et } \mathbf{W}_k = \begin{pmatrix} X_k \\ 0 \end{pmatrix} \tag{2.9}$$

l'état du ré-encodeur $\mathbf{S}_k$ et sa redondance correspondante $\mathbf{V}'_k$ peuvent s'exprimer de la manière suivante :

$$\mathbf{S}_{k+1} = \mathbf{A} \cdot \mathbf{S}_k + \mathbf{W}_k \tag{2.10}$$

$$\mathbf{V}'_k = \mathbf{C} \cdot \mathbf{S}_k + \mathbf{U}_k \tag{2.11}$$

Enfin, les informations issues du canal de transmission $\mathbf{V}_k$ et de l'étape de ré-encodage $\mathbf{V'}_k$ sont combinées :

$$\bar{\mathbf{V}}_k = \mathbf{V}_k + \mathbf{V'}_k \tag{2.12}$$

A partir des expressions 2.8 et 2.11, nous obtenons :

$$\bar{\mathbf{V}}_k = \begin{pmatrix} \bar{X}_k \\ \bar{Y}_k \end{pmatrix} = \begin{pmatrix} X_k \\ Y_k \end{pmatrix} + \begin{pmatrix} X_k \\ Y'_k \end{pmatrix} \tag{2.13}$$

Nous avons donc :

$$\bar{\mathbf{V}}_k^K = \begin{pmatrix} x_k + E_{x,k} \\ y_k + E_{y,k} \end{pmatrix} + \begin{pmatrix} x_k + E_{x,k} \\ Y'_k \end{pmatrix} = \begin{pmatrix} 0 \\ y_k + E_{y,k} + Y'_k \end{pmatrix} \tag{2.14}$$

A partir des expressions 2.3 et 2.4, l'état $\mathbf{s}_k$ du codeur à l'instant k peut s'exprimer de la manière suivante :

$$\mathbf{s}_k = \mathbf{A}^k \cdot \mathbf{s}_0 + \sum_{i=1}^{k} \mathbf{A}^{k-1} \cdot \mathbf{w}_{i-1} \tag{2.15}$$

Si le codeur est initialisé à l'état 0, $\mathbf{s}_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ l'expression 2.15 peut être simplifiée comme suit :

$$\mathbf{s}_k = \sum_{i=1}^{k} \mathbf{A}^{k-1} \cdot \mathbf{w}_{i-1} \tag{2.16}$$

De manière similaire, si le ré-encodeur est initialisé à l'état 0, l'état $\mathbf{S}_k$ peut s'exprimer par l'équation 2.17 :

$$\mathbf{S}_k = \sum_{i=1}^{k} \mathbf{A}^{k-1} \cdot \mathbf{W}_{i-1} \tag{2.17}$$

où

$$\mathbf{W}_{i-1} = \mathbf{w}_{i-1} + \epsilon_{i-1} \quad \text{et} \quad \epsilon_{i-1} = \begin{pmatrix} E_{x,i-1} \\ 0 \end{pmatrix} \tag{2.18}$$

L'état $\mathbf{S}_k$ est égal à la somme de l'état $\mathbf{s}_k$ et l'état $\hat{\mathbf{S}}_k$:

$$\mathbf{S}_k = \mathbf{s}_k + \hat{\mathbf{S}}_k \tag{2.19}$$

où l'état $\hat{\mathbf{S}}_k$ à l'instant k s'écrit à partir du vecteur d'erreurs ϵ :

$$\hat{\mathbf{S}}_k = \sum_{i=1}^{k} \mathbf{A}^{k-1} \epsilon_{i-1} = \begin{pmatrix} \hat{S}_{1,k} \\ \hat{S}_{2,k} \end{pmatrix} \tag{2.20}$$

L'état $\hat{\mathbf{S}}_k$ correspond au codage du vecteur $\mathbf{E}_x$ à partir de l'état $\mathbf{S}_0$. La redondance associée est notée $\hat{Y}_{e,k}$. Nous pouvons alors exprimer la redondance Y'_k par la somme des redondances y_k et $\hat{Y}_{e,k}$ comme suit :

$$Y'_k = y_k + \hat{Y}_{e,k} \tag{2.21}$$

Il est donc possible de reformuler l'expression 2.14 à partir des expressions 2.17 et 2.21 :

$$\begin{aligned} \bar{\mathbf{V}}_k^N &= \begin{pmatrix} 0 \\ y_k + E_{y,k} + Y'_k \end{pmatrix} = \begin{pmatrix} 0 \\ y_k + E_{y,k} + y_k + \hat{Y}_{e,k} \end{pmatrix} \\ &= \begin{pmatrix} 0 \\ E_{y,k} + \hat{Y}_{e,k} \end{pmatrix} \end{aligned} \tag{2.22}$$

En conclusion, les données combinées $\bar{\mathbf{V}}_k^N$ du ré-encodage et du mot bruité sont nulles pour la partie systématique. La partie redondance correspond à la somme du bruit $E_{y,k}$ à l'instant k avec la contribution de la redondance du bruit $\hat{Y}_k$ depuis l'état initial du ré-encodeur.

Remarque : la technique de ré-encodage n'a de sens que si le code utilisé est systématique. Dans le cas contraire, un décodage préliminaire doit être appliqué avant l'étape de ré-encodage pour extraire l'information initiale comme suggéré dans [Kubota 86].

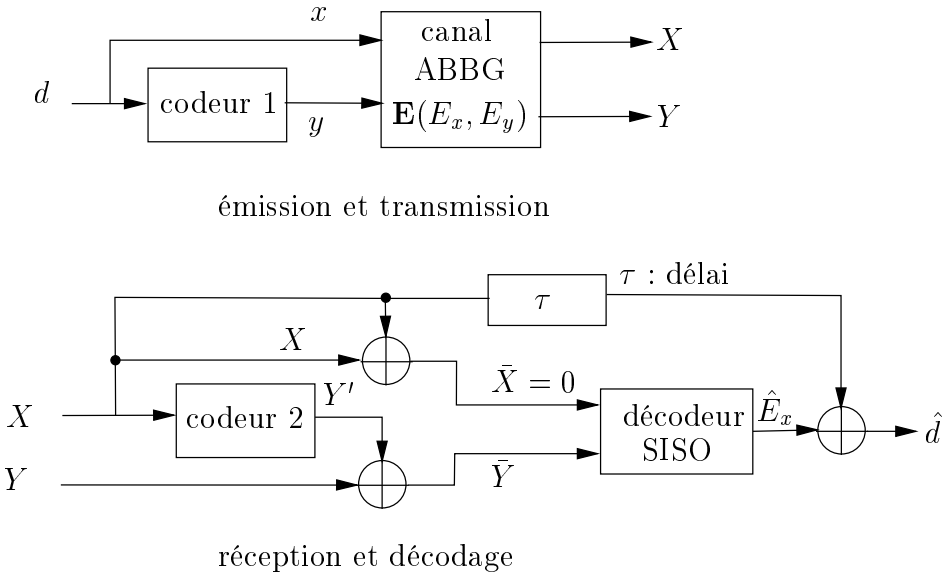


Figure 2.2 — Schéma de principe d'un système différentiel.

La figure 2.2 montre le schéma de principe d'un système différentiel dédié au Code Convolutif Récursif Systématique (CCRS). Les opérations sont définies comme des opérations modulo 2. Les valeurs du vecteur sont prises dans un alphabet $A = \{0, 1\}$, qui correspond aux valeurs

de la décision dure. A partir d'un message à transmettre d, le codeur convolutif 1 produit un mot de code $\mathbf{v}(x, y)$. La propriété du CCRS implique l'égalité $x = d$. Ce mot de code $\mathbf{v}(x, y)$ est ensuite transmis par un canal ABBG ayant une variance σ^2. Les erreurs de transmission correspondant aux valeurs x et y sont respectivement notées E_x et E_y, qui forment un couple d'erreurs $\mathbf{E}(E_x, E_y)$. Ainsi, nous obtenons le mot bruité en couple $\mathbf{V}(X, Y)$ tel que :

$$X = x + E_x \quad \text{et} \quad Y = y + E_y \tag{2.23}$$

Rappelons que le décodeur traite directement le mot bruité $\mathbf{V}(X, Y)$ dans un système classique. Par ailleurs, dans un système différentiel, l'étape de décodage est précédée d'un processus ré-encodage sur la partie systématique X. De plus, les codeurs 1 et 2 sont identiques et disposent donc des mêmes caractéristiques (figure 2.2). A la réception, le ré-encodage de l'entrée X donne alors un mot $\mathbf{V}'(X, Y')$ dont la redondance Y' est fonction de l'entrée X, tandis que la partie systématique est égale à X. Le mot combiné $\bar{\mathbf{V}}(\bar{X}, \bar{Y})$ est obtenu à partir du mot $\mathbf{V}'(X, Y')$ et du mot $\mathbf{V}(X, Y)$. Le décodeur SISO fournit alors une estimation de l'erreur $\hat{E}_x$. Un délai τ est compté pour récompenser le temps écoulé dans le décodeur SISO de la figure 2.2. Pour reconstituer le message initialement émis, il suffit alors d'effectuer une opération ou-exclusif entre $\hat{E}_x$ et X retardé d'un délai τ. Nous obtenons :

$$\hat{E}_x + X = \hat{E}_x + (x + E_x) = \hat{d} \tag{2.24}$$

Comme évoqué précédemment, le chemin survivant suit le chemin TAZ tout au long du décodage lorsque le vecteur $\mathbf{E}_x$ est nul. Cela correspond au cas du rapport signal à bruit élevé. La distribution de probabilité sur le chemin survivant au cours du décodage est alors repartie de manière à maximiser la probabilité sur le chemin TAZ. Comme le vecteur $\mathbf{E}_x$ dépend fortement de la valeur du rapport SNR, l'activité du système différentiel est donc fonction du rapport SNR, ce qui n'est pas le cas pour le système standard. Plus le rapport SNR est élevé, moins il y a d'erreurs à décoder, moins le système différentiel est actif.

2.1.2.2 Fermeture de treillis au cours du codage différentiel

La grande majorité des systèmes de transmissions numériques échangent l'information sous forme de trames indépendantes. Or, les codes convolutifs s'appliquent par défaut à des messages de longueur infinie. Appliquer ce type de code à des trames indépendantes s'avère moins efficace sur le début et la fin des trames par rapport à un code en bloc. Pour pallier ce problème, il est nécessaire de connaître l'état initial et l'état final du codeur lors du décodage d'une trame. Les techniques utilisées pour connaître ces états sont connues sous l'appellation *fermeture de treillis*. Classiquement, ces états pour le codeur sont forcés à une valeur connue par le décodeur (état 0 le plus souvent)[Berrou 07]. Nous allons dans la suite de cette section tenter d'étendre cette technique de fermeture de treillis à un codage différentiel.

Dans notre système différentiel, le codage commence à l'état zéro par convention pour les codeurs de l'émission et de la réception. La technique *tail biting* est appliquée pour fermer le treillis après codage des L bits du message à transmettre. Cette technique consiste à forcer l'état final du codeur à l'état zéro à la fin du codage en ajoutant des bits supplémentaires appelés *tail bits*. Le nombre maximum de *tail bits* est égal à $v - 1$, soit la longueur de mémoire du codeur. Par exemple, la figure 2.3 représente la technique de *tail biting* adoptée par la norme UMTS [3GPP 99]. Le rebouclage en pointillé est uniquement appliqué pendant la fermeture de treillis. La partie systématique est reliée à la valeur entrante d au cours du codage. Cela

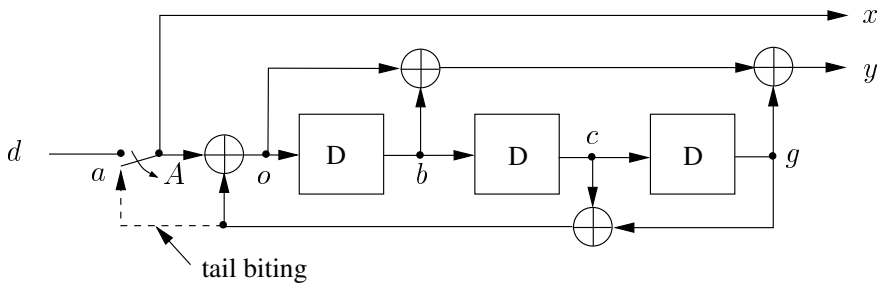


Figure 2.3 — Codeur avec la technique *tail biting* (extrait de la norme UMTS [3GPP 99]).

implique l'égalité $A = d$. Lorsque le codage de la trame est terminé, la position de A permet la liaison avec le point a ($A = a$) pour fermer le treillis. La valeur du bit en position o est alors égale à 0 :

$$o = a \oplus a = 0 \tag{2.25}$$

Ensuite, la valeur 0 se propage au sein des registres au cours des trois périodes d'horloge suivante. Ainsi, la partie systématique de *tail bits* s'écrit :

$$x = c \oplus g \tag{2.26}$$

La valeur x pendant la fermeture du treillis dépend donc du contenu des registres, à savoir l'état final du codeur. Il est donc nécessaire d'ajouter 3 *tail bits* pour fermer le treillis d'un codeur à 8 états. En effet, $v-1$ *tail bits* assurent la fermeture d'un treillis associé à un codeur à $N_e = 2^{v-1}$ états.

2.1.2.3 Etude du codage différentiel appliqué à l'algorithme de Viterbi

Le codage différentiel est appliqué dans un premier temps pour le décodeur de Viterbi associé à un codeur à 4 états. La figure 2.4 donne les performances en terme de TEB (l'axe des abscisses) par rapport au rapport SNR (l'axe des ordonnées) d'un décodeur de Viterbi avec et sans application du codage différentiel. La modulation MDP2 (Modulation De Phase à 2) est considérée dans cette étude. La longueur de trame simulée L est égale à 1000 bits. Le codeur convolutif a pour polynômes générateurs $G[1, (1+D^2)/(1+D+D^2)]$ (figure 2.1). Les données sont transmises à travers un canal symétrique. Nous ne constatons donc pas de dégradation des performances sur ces deux systèmes. Ce résultat est prévisible, car comme montré dans la section 2.1.2.1, les calculs sont identiques.

Considérons que les métriques de nœud α au cours du décodage soient représentées par une matrice M_a ayant la taille $L \times N_e$, tel que les symboles L et N_e représentent respectivement la longueur de trame et le nombre d'états :

$$M_a = \begin{bmatrix} \alpha_0^0 & \cdots & \alpha_j^0 & \cdots & \alpha_{L-1}^0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \alpha_0^i & \cdots & \alpha_j^i & \cdots & \alpha_{L-1}^i \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \alpha_0^{N_e-1} & \cdots & \alpha_j^{N_e-1} & \cdots & \alpha_{L-1}^{N_e-1} \end{bmatrix}$$

avec α_k^j représente la métrique de nœud sur l'état j à l'instant k. Nous introduisons la fonction

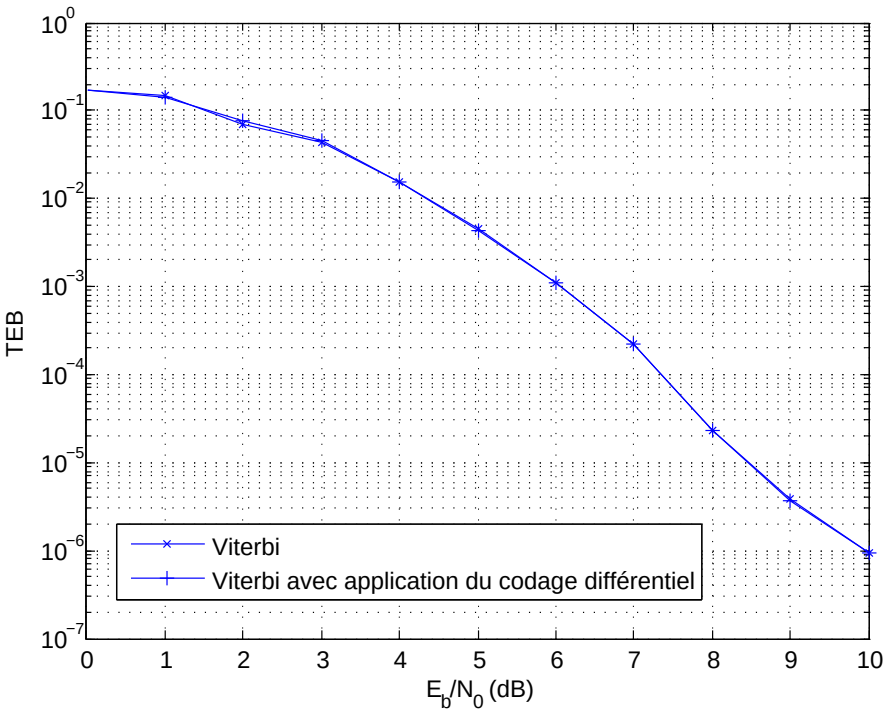


Figure 2.4 — Comparaison des performances pour le décodeur de Viterbi à 4 états avec et sans application du codage différentiel.

$\prod(k, L_c)$ pouvant prendre les valeurs 0 ou 1 comme :

$$\prod(k, L_c) = \prod_{i=0}^{\min(L_c-1,k)} F(\min(\alpha_{k-i}^0, \cdots, \alpha_{k-i}^{N_e-1}) = \alpha_{k-i}^0) = 1 \; k \in \{0, \cdots, L-1\} \tag{2.27}$$

dont le symbole L_c représente la longueur d'une fenêtre glissante tandis que la fonction booléenne $F(c)$ est égale à 1 lorsque l'expression c est vraie, 0 sinon. Le taux d'occupation T_o sur le chemin TAZ est défini par le rapport suivant :

$$T_o = \frac{1}{L} \sum_{k=0}^{L-1} \prod(k, L_c) \tag{2.28}$$

Le taux T_o caractérise donc un système de décodage différentiel en mesurant la probabilité que le chemin survivant soit ramené sur le chemin TAZ.

La figure 2.5 représente le taux d'occupation T_o en fonction du taux d'erreur binaire pour une longueur d'observation $L_c = 7$ sur le décodeur de Viterbi à 4 états avec et sans application du codage différentiel. L'axe des abscisses représente les valeurs du TEB, tandis que l'axe des ordonnées représente les taux d'occupation obtenus. Nous constatons que le taux d'occupation T_o est reparti de manière homogène et sa valeur est très proche de zéro pour le décodeur de Viterbi sans application du codage différentiel. En revanche, ce taux T_o augmente inversement à la valeur du TEB de manière significative à partir d'un TEB de l'ordre de 10^{-2}. Nous constatons que le taux T_o atteint 71% pour un TEB de l'ordre de 10^{-6}.

Nous pouvons par conséquent conclure que le décodeur de Viterbi avec application du codage différentiel a une activité réduite car le taux d'occupation du chemin TAZ augmente

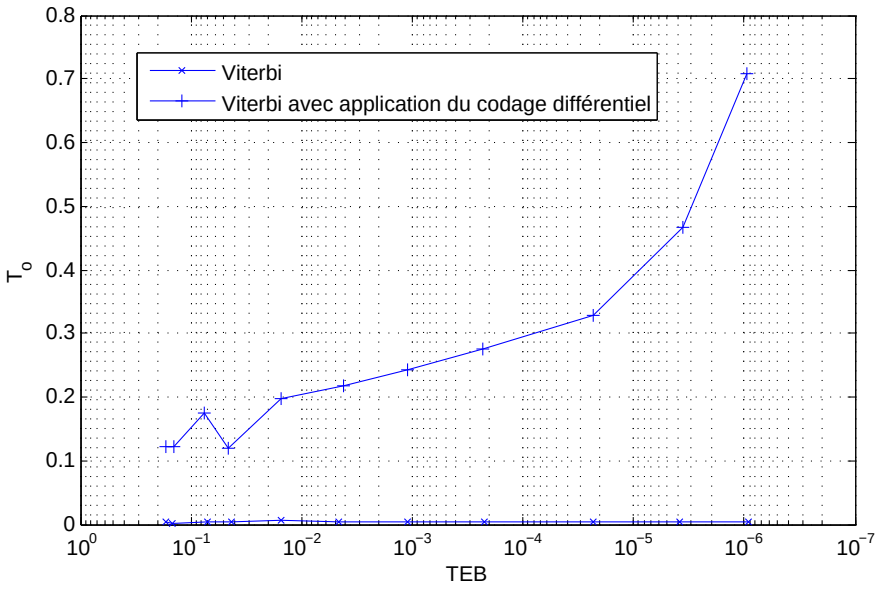


Figure 2.5 — Comparaison des taux d'occupation sur le chemin TAZ pour le décodeur de Viterbi à 4 états avec et sans application du codage différentiel.

de manière significative en fonction du SNR.

2.1.3 Application du codage différentiel à l'algorithme Max-Log-MAP

Il est possible d'étendre la technique du ré-encodage à l'algorithme Max-Log-MAP. Dans cette section, nous décrivons d'abord le principe du codage différentiel appliqué à l'algorithme Max-Log-MAP. Ensuite, le codage différentiel est utilisé dans un turbo-décodeur adapté au système UTMS. Enfin, nous présentons les performances de turbocodes avec codage différentiel et discutons l'activité du décodage, soit le nombre de transitions binaires sur les valeurs des métriques de nœud au cours du décodage.

2.1.3.1 Codage différentiel appliqué à l'algorithme Max-Log-MAP

Contrairement à l'algorithme de Viterbi étudié dans la section précédente, qui travaille sur des décisions dures du message bruité, l'algorithme Max-Log-MAP considère directement le message bruité (les informations souples) en calculant la probabilité de vraisemblance sur le bit décodé. La figure 2.6 montre l'application du codage différentiel à l'algorithme Max-Log-MAP. Dans ce cas, le ré-encodage est appliqué sur une décision dure du message bruité, plus précisément sur son bit de signe. L'opération ou-exclusif entre les mots $\mathbf{v}(x, y)$ et $\mathbf{V}'(X', Y')$, quant à elle, est effectuée tel que les bits de signe du mot $\bar{\mathbf{V}}(\bar{X}, \bar{Y})$ soient inversés tout en gardant les mêmes amplitudes lorsqu'ils rencontrent une valeur 1 (figure 2.6), sinon, leur valeur respective ne change pas. Le trait épais noir représente la valeur de l'amplitude tandis que le trait fin représente le bit de signe égal à 0 ou 1.

Les symboles X et Y représentent respectivement la partie systématique et la partie redondante du message bruité, quantifiés sur des valeurs souples. Le codage différentiel est réalisé à partir du bit de signe de X, noté MSB(X) (Most Significant Bit en anglais) dans la figure 2.6, suivi par une opération ou-exclusif $\oplus$ entre le mot codé $\mathbf{V}'(X', Y')$ et les bits

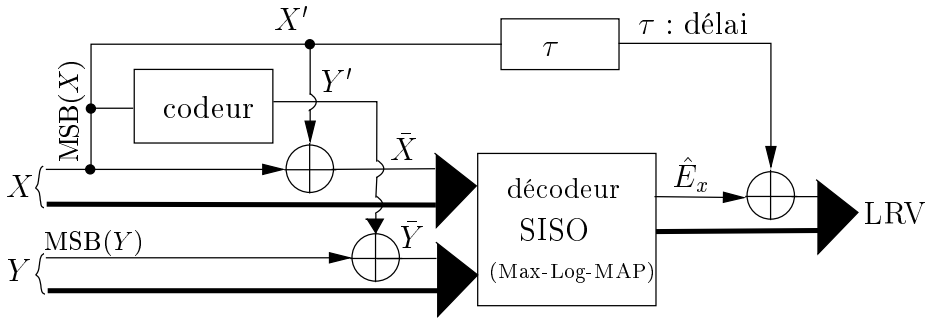


Figure 2.6 — Schéma du principe de système différentiel appliqué à l'algorithme Max-Log-MAP.

de signe du mot correspondant MSB($\mathbf{V}(X,Y)$). Les amplitudes du mot $\mathbf{V}(X,Y)$ ne sont pas modifiées au cours du codage différentiel. A partir du mot $\bar{\mathbf{V}}(\bar{X},\bar{Y})$, le décodeur SISO associé à l'algorithme Max-Log-MAP fournit une estimation de l'erreur $\hat{E}_x$ sur le bit considéré.

Comme pour le décodeur de Viterbi, une opération ou-exclusif entre le bit de signe de $\hat{E}_x$ et le bit X' retardé d'une unité de temps τ permet de reconstituer le rapport LRV (figure 2.6). Cette valeur reconstituée est identique par rapport à celle obtenue par le décodeur Max-Log-MAP sans application du codage différentiel. C'est pourquoi le système différentiel ne doit pas engendrer de dégradation en terme de performance. Si aucune erreur n'est présente sur le message $\mathbf{V}(X,Y)$, le chemin survivant lors du parcours du treillis suit le chemin TAZ avec une probabilité maximale.

2.1.3.2 Performances d'un turbo-décodeur avec codage différentiel dans le système UMTS

Dans cette section, nous présentons dans un premier temps les caractéristiques des turbocodes adoptés par la norme UMTS [3GPP 99]. Nous détaillons ensuite les performances de ce type de turbocode.

Les turbocodes dans le système UMTS

Le standard UMTS [3GPP 99] a adopté les turbocodes comme option pour le codage de canal (figure 2.7). Le codeur est constitué de deux codeurs élémentaires à 8 états en concaténation parallèle ayant pour polynômes générateurs $G[1,(1+D+D^3)/(1+D^2+D^3)]$. Le codage commence par convention à l'état initial 0. Cela implique que les bascules du codeur ont pour valeur 0 à l'initialisation. L'ordre des sorties du codeur suit :

$$\begin{array}{llll} x_0, & x_1, & \cdots & x_{L-1} \\ y_{1,0}, & y_{1,1}, & \cdots & y_{1,L-1} \\ y_{2,0}, & y_{2,1}, & \cdots & y_{2,L-1} \end{array}$$

où le symbole x_k représente le bit d'entrée du codeur 1 et de l'entrelaceur $\prod$ à l'instant k, L est la longueur de la trame, tandis que les symboles $y_{1,k}$ et $y_{2,k}$ sont respectivement les redondances produites par les codeurs 1 et 2 à l'instant k. Le codeur 2, quant à lui, reçoit les bits entrelacés notés $x'_0, x'_1, \cdots, x'_{L-1}$ avec $x'_k = x_{\prod(k)}$.

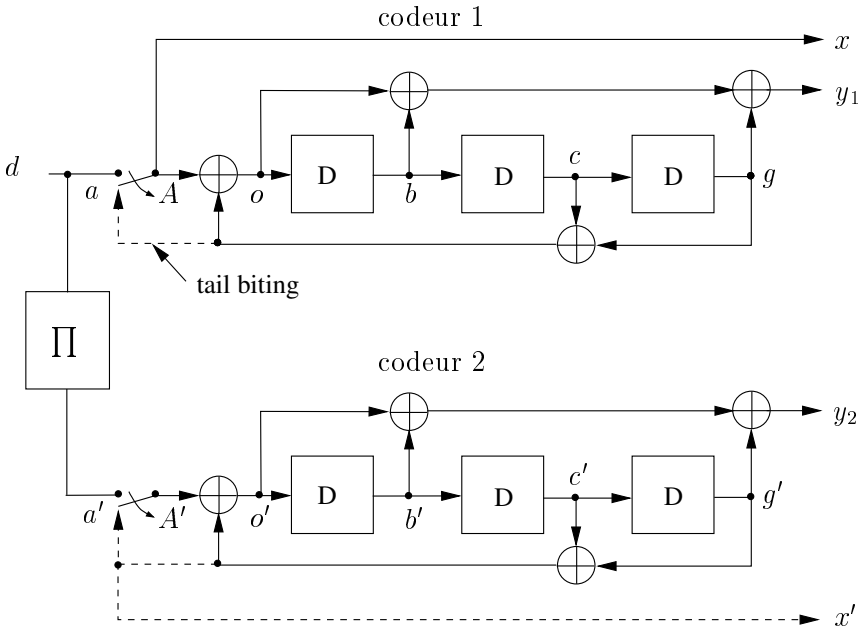


Figure 2.7 — Les turbocodes adoptés par la norme UMTS, extraits de la spécification [3GPP 99].

La fermeture de treillis est appliquée après le codage des L bits. Comme déjà indiqué dans la section 2.1.2.2, l'interrupteur A est relié à la position a dont la valeur résulte de l'opération ou-exclusif $a = c \oplus g$. Ainsi, la valeur en position o est forcée à 0. En effet, il s'agit d'une opération ou-exclusif entre a et lui-même. Par conséquent, les bits sortants de la fermeture dépendent de l'état final du codeur 1. Le codeur 2 est désactivé lorsque la fermeture de treillis prend effet pour le codeur 1. De même, ce processus s'exécute sur le codeur 2 dès que le treillis est fermé sur le codeur 1. La transmission des *tail bits* respecte l'ordre suivant :

$$\begin{array}{llllll} x_L, & x_{L+1}, & x_{L+2}, & x'_L, & x'_{L+1}, & x'_{L+2} \\ y_{1,L}, & y_{1,L+1}, & y_{1,L+2}, & \times, & \times, & \times \\ \times, & \times, & \times, & y_{2,L}, & y_{2,L+1} & y_{2,L+2} \end{array}$$

où le symbole $\times$ représente l'absence de l'information. Les traits pointillés sont utilisés uniquement pour la fermeture de treillis. Le taux de codage est donc égal à $R = 1/3$ sans tenir compte des *tail bits* supplémentaires.

Nous ne détaillons pas l'algorithmique pour l'entrelaceur et le désentrelaceur d'un système UMTS dans ce document. Pour plus d'information, le lecteur peut consulter le standard UMTS [3GPP 99].

Application du codage différentiel dans un turbo-décodeur du système UMTS

Au niveau du turbo-décodage, nous pouvons établir une relation entre le mot émis $\mathbf{v}(x, y_1, y_2)$ et le mot reçu $\mathbf{V}(X, Y_1, Y_2)$ par l'expression 2.29 :

$$\underbrace{\begin{pmatrix} x \\ y_1 \\ y_2 \end{pmatrix}}_{\mathbf{v}} + \underbrace{\begin{pmatrix} E_x \\ E_{y_1} \\ E_{y_2} \end{pmatrix}}_{\mathbf{E}} = \underbrace{\begin{pmatrix} X \\ Y_1 \\ Y_2 \end{pmatrix}}_{\mathbf{V}} \tag{2.29}$$

où les données émises x, y_1 et y_2 sont bruitées respectivement par les erreurs correspondantes E_x, E_{y_1} et E_{y_2} en produisant les éléments reçus X, Y_1 et Y_2. Le codage différentiel peut

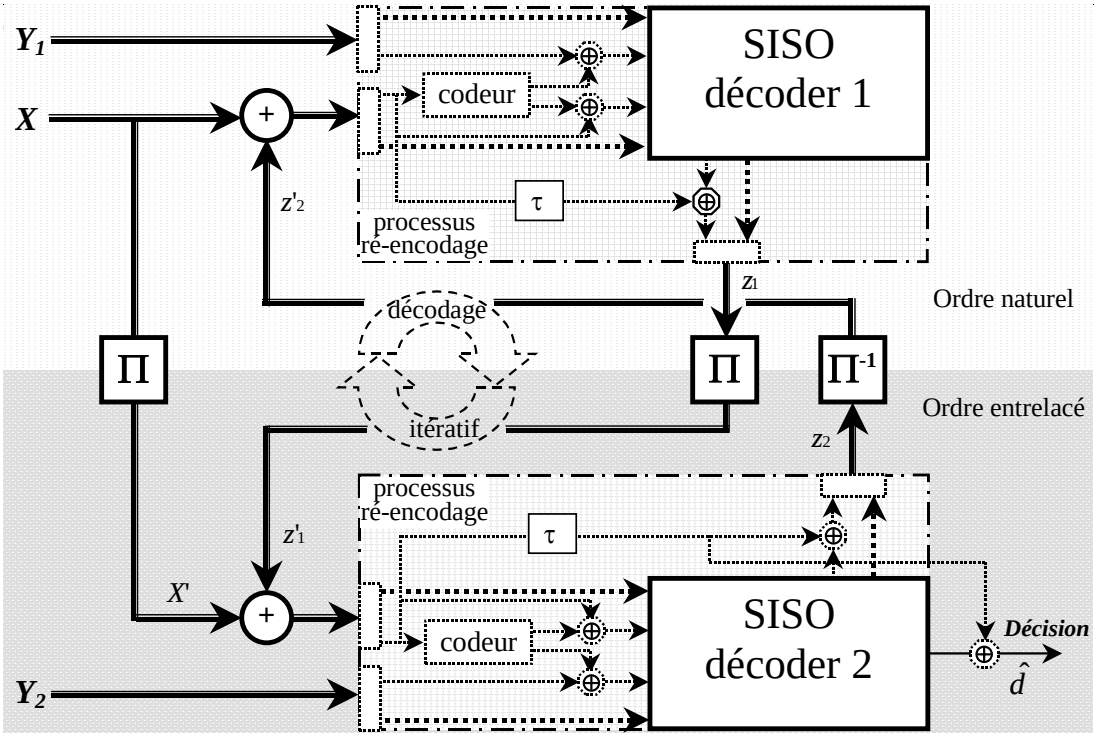


Figure 2.8 — Schéma de principe d'un système différentiel appliqué à toutes les demi-itérations de turbo-décodage.

s'appliquer à l'entrée des décodeurs élémentaires SISOs utilisant l'algorithme Max-Log-MAP (figure 2.8). Dans ce cas, l'ensemble du processus itératif est réalisé en intégrant l'aspect différentiel. Le décodage effectué dans un décodeur élémentaire SISO est dit décodage dans *une démie-itération*. Quand le codage différentiel est appliqué à toutes les demi-itérations de turbo-décodage, le ré-encodage et l'opération ou-exclusif sont effectués à partir des bits de signe des données correspondantes. La décision finale, quant à elle, est obtenue à l'aide d'une seconde opération ou-exclusif à l'issue du décodeur 2 (voir la figure 2.8). Le symbole τ représente le délai nécessaire. Outre l'entrelaceur et le désentrelaceur, le codage différentiel est effectué pour le décodeur 1 à partir de la somme $z_2' + X$. Comme expliqué précédemment, seul le bit de sign est considéré. Le même codage prend effet pour le décodeur 2 à partir de la somme $X' + z_1'$. Le ré-encodage est ainsi appliqué à toutes les demi-itérations.

Il est cependant possible d'appliquer le codage différentiel à certaines itérations et non pas à l'ensemble des itérations. Dans ce cas, le schéma de principe peut être représenté comme celui de la figure 2.9. Le codage différentiel s'applique à partir des la somme $X + z_1$ après un certain nombre d'itérations. Dans ce cas, le turbo-décodage comprend deux phases. Le codage différentiel est appliqué sur la seconde phase et non pas sur la première. De même, la décision finale $\hat{d}$ est obtenue à l'aide d'une seconde opération ou-exclusif entre la décision dure $\hat{d}_2'$ et la somme $X + z_1$ retardée d'un temps τ.

Sur la figure 2.8, le Codage Différentiel (CD) profite pleinement les informations d'échange entre les décodeurs élémentaires. Cet échange d'information est primordial car il maximise la probabilité que le chemin survivant puisse être ramené au chemin TAZ au cours du décodage. Or, cela n'est pas le cas sur la figure 2.9, puisque le codage différentiel n'est effectué qu'une

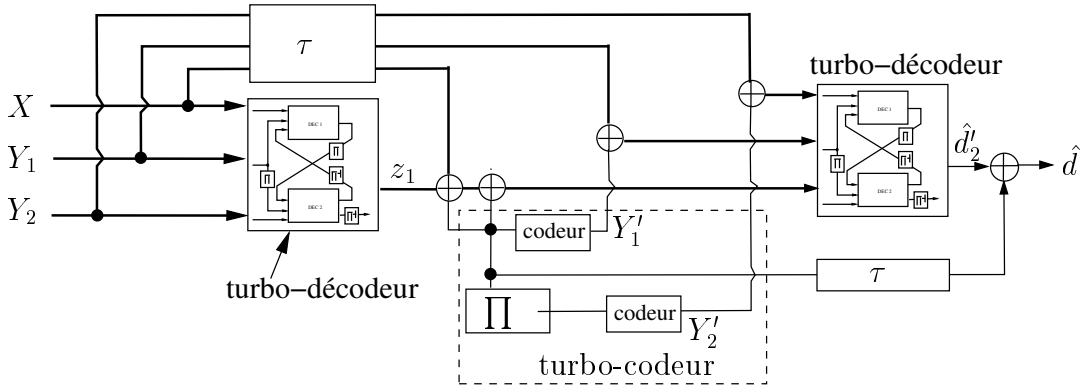


Figure 2.9 — Schéma de principe d'un système différentiel appliqué à un sous-ensemble des itérations de turbo-décodage.

seule fois et il ne profite pas tous les résultats d'une demi-itération au cours du décodage itératif. Aucun échange d'informations ne se produit à l'entrée du turbo-codeur au cours de la seconde phase du décodage. C'est pourquoi nous utilisons dans la suite du mémoire la présentation de la figure 2.8.

Remarque : lorsque les codes à l'émission sont circulaires, il n'est pas nécessaire de rechercher l'état circulaire à la réception lors du codage différentiel, comme expliqué dans la section 1.5.2. Il suffit alors d'utiliser la technique de fermeture de treillis à cette étape. Ainsi, la complexité du codage différentiel pour les codes circulaires et le temps d'exécution sont diminués. En conclusion, le codage différentiel est indépendant de la nature des codes employés : circulaires ou non circulaires.

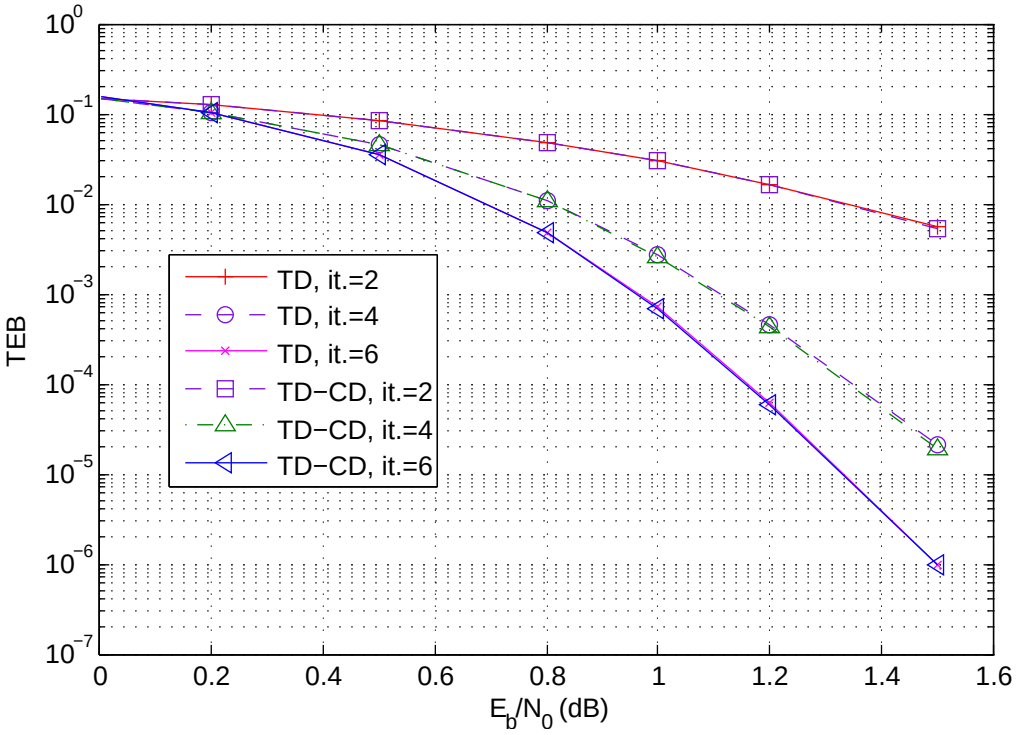


Figure 2.10 — Comparaison des performances des turbo-décodeurs (TD et TD-CD) adaptés à la norme UMTS.

Le turbo-décodeur classique et le turbo-décodeur avec application du codage différentiel sont respectivement notés TD et TD-CD. La figure 2.10 donne les performances en terme du TEB des turbo-décodeurs (TD et TD-CD) adaptés à la norme UMTS en fonction du rapport SNR pour les itérations 2, 4 et 6. Le contexte de simulation est le suivant :

- l'algorithme Max-Log-MAP est appliqué au décodeur élémentaire SISO ;
- la longueur de trame L est égale à 864 bits ;
- les modulations MDP2 ou MDP4 et un canal ABBG sont considérés ;
- les données entrantes du décodeur sont quantifiées sur 5 bits dans un intervalle [-1.875, 1.875] avec le pas de quantification $q = 0.125$;
- le coefficient de l'information extrinsèque ζ défini dans le chapitre 1 est égal à 0.5 pour les deux premières itérations, 1 pour la dernière itération, 0.75 pour le reste.
- les informations extrinsèques sont quantifiées sur 5 bits pour les deux premières itérations, 7 bits pour les autres ;
- le codage différentiel est appliqué à l'entrée des décodeurs élémentaires du système présenté par la figure 2.8 ;
- le nombre d'itération est égal à 6 sans appliquer de critère d'arrêt ;
- le décodage se déroule de façon classique sans appliquer la technique *sliding-windows* [Boutillon 07] ;

Nous ne constatons pas de perte des performances dans la figure 2.10. En conclusion, il est possible d'appliquer le codage différentiel au processus de turbo-décodage utilisant l'algorithme Max-Log-MAP. La question qu'il faut néanmoins considérer est cela diminue-t-il l'activité globale du turbo-décodeur. Nous allons traiter cette question dans la section suivante.

2.1.4 Activité d'un turbo-décodeur avec application du codage différentiel

L'activité **a** du système mesure le nombre de transitions binaires entre les valeurs binaires 1 et 0 par l'unité de temps. Pour un décodeur utilisant l'algorithme Max-Log-MAP, nous calculons en particulier le nombre de transitions pour les métriques de nœud α de la manière suivante :

$$\mathbf{a} = \frac{\sum_{j=0}^{L-1} \sum_{i=0}^{N_e-1} (\alpha_j^i \oplus \alpha_{j+1}^i)}{L} \tag{2.30}$$

Le tableau 2.1 donne une comparaison au niveau activité en moyenne pour les deux décodeurs (TD et TD-CD). Les activités respectives sont notées $\mathbf{a}_0$ et $\mathbf{a}_1$. La diminution d'activité est notée D_{a1}, qui est obtenue à partir de l'expression suivante :

$$D_{a1} = \frac{\mathbf{a}_0 - \mathbf{a}_1}{\mathbf{a}_0} \tag{2.31}$$

Nous ne constatons pas une diminution d'activité considérable. Par exemple, les activités $\mathbf{a}_0$ et $\mathbf{a}_1$ sont égales pour SNR = 0dB. Par ailleurs, une réduction de 5% est obtenue pour des valeurs respectives de SNR égales à 0.2dB, 0.5dB et 0.8dB.

Le tableau 2.1 ne donne pas un résultat favorable en terme de diminution d'activité. C'est pourquoi nous proposons une approche dans la section suivante pour tenter d'éliminer des motifs répétitifs apparus après le codage différentiel. Grâce à cette approche, nous pouvons améliorer le rapport d'occupation T_o du chemin survivant sur le chemin TAZ au cours du décodage.

SNR (dB)	0	0.2	0.5	0.8	1	1.2	1.5
$\mathbf{a}_0$ (TD)	19	20	21	22	23	23	23
$\mathbf{a}_1$ (TD-CD)	19	19	20	21	22	22	22
D_{a1}	0%	5%	5%	5%	4%	4%	4%

Tableau 2.1 — Comparaison d'activité en moyenne des décodeurs (TD et TD-CD).

2.2 Insertion d'erreurs au cours du décodage différentiel

Dans cette section, nous proposons une méthode, appelée *insertion de dummy-errors*, permettant d'éliminer les motifs périodiques dus au codage différentiel. Grâce à cette insertion, nous pouvons diminuer l'activité du système au cours du décodage.

2.2.1 Motif répétitif après codage différentiel

Reprenons l'expression 2.15 décrivant la relation entre l'état du codeur $\mathbf{s}_k$ et son état initial $\mathbf{s}_0$ ainsi qu'un vecteur $\mathbf{w}_{i-1}$ en fonction des données entrantes du codeur :

$$\mathbf{s}_k = \mathbf{A}^k \cdot \mathbf{s}_0 + \sum_{i=1}^{k} \mathbf{A}^{k-1} \cdot \mathbf{w}_{i-1} \tag{2.32}$$

Nous avons une propriété de la matrice $\mathbf{A}$ pour les codes systématiques convolutifs tel que :

$$\mathbf{A}^k = \mathbf{A}^{k+N_m} \tag{2.33}$$

avec N_m la périodicité de la matrice $\mathbf{A}$. La valeur de N_m est inférieure au nombre d'états N_e. Si l'état initial $\mathbf{s}_0 = 0$ et le terme $\mathbf{w}_{i-1} = 0$, nous obtenons alors :

$$\mathbf{s}_k = 0 \tag{2.34}$$

ce qui correspond au chemin TAZ. Par ailleurs, lorsque le terme $\mathbf{w}_{i-1}$ vaut 0 et l'état initial $\mathbf{s}_0$ n'est pas à l'état 0, soit $\mathbf{s}_0 \neq 0$, l'état du codeur évolue de manière périodique selon une périodicité N_m. Les états périodiques dans un treillis est appelées *motif d'état* et leurs redondances associées *motif de redondance*, notés respectivement M^e et M^r.

Rappelons que le principe du codage différentiel est de transmettre un vecteur d'erreur $\mathbf{E}_x$ au lieu du message émis $\mathbf{x}$. Ensuite, ce vecteur est associé à un vecteur d'erreur égal à lui-même. Grâce au codage différentiel et la combinaison des mots $\mathbf{V}$ et $\mathbf{V}'$, le mot à l'entrée du décodeur est $\bar{\mathbf{V}} = \mathbf{V} + \mathbf{V}'$. La partie redondance du mot $\bar{\mathbf{V}}$ correspond au terme $E_y + \hat{Y}_e$ dont les symboles $\hat{Y}_e$ et E_y représente respectivement la partie redondance du codage du E_x et l'erreur sur la redondance émise y (voir l'équation 2.22).

Supposons le vecteur d'erreur $\mathbf{E}_x = (1000000000)$ pour une transmission du code ayant pour polynômes générateurs $G[1, (1+D^2)/(1+D+D^2)]$, nous avons alors :

$$\mathbf{A} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \text{ et } N_m = 3.$$

La figure 2.11 montre l'évolution des états du codeur dans le treillis correspondant. L'état du codeur passe de l'état 0 à 2 dans le premier étage du treillis puisque son entrée correspondante est égale à 1. Dès lors, le terme $\mathbf{w}_{i-1}$ est égal à zéro. D'après la propriété de périodicité, le

motif répétitif d'états M^e est $M^e = (2 \rightarrow 3 \rightarrow 1)$. Nous constatons que ce motif passe par tous les états sauf l'état 0. Quant au motif de redondance associé M^r, il est représenté par $M^r = (1 \rightarrow 0 \rightarrow 1)$. Le parcours dans le treillis est représenté par le trait noir épais dans la figure 2.11.

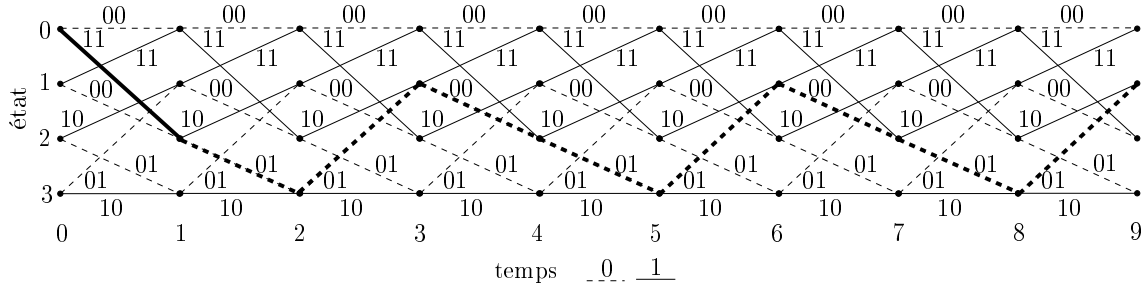


Figure 2.11 — Evolution d'états lors du parcours du treillis pour le vecteur d'erreur $\mathbf{E}_x = (1000000000)$.

Le chemin survivant peut être ce parcours dans le treillis lorsque la valeur de SNR est élevée. Nous constatons alors que le chemin survivant ne suit pas le chemin TAZ. Cette observation est contradictoire avec l'intérêt du codage différentiel car cela limite le taux d'occupation sur le chemin TAZ. Il est donc nécessaire de ramener le chemin survivant sur l'état zéro pour maximiser le taux d'occupation et ainsi rendre le système du décodage moins actif. Pour ce faire, une des méthodes consiste à profiter de la périodicité du motif en insérant des erreurs supplémentaires à l'entrée du codeur à la réception. Les erreurs insérées sont appelées *dummy-errors*.

2.2.2 Elimination du motif par insertion de dummy-errors

L'apparition du motif répétitif après codage différentiel à la réception diverge du chemin survivant du chemin TAZ. Pour limiter cet effet, nous décrivons dans cette section une méthode permettant d'enlever ce motif. En profitant de la propriété de périodicité du motif sur la redondance, nous pouvons insérer des erreurs supplémentaires $\mathbf{e}_i$, dites *dummy-errors*, au vecteur $\mathbf{X}$ de telle manière à générer le même motif au cours du ré-encodage. Par conséquent, le codage combiné du vecteur $\mathbf{X} + \mathbf{e}_i$ ne contient plus du motif. Ainsi, l'étape de ré-encodage n'est pas basée sur le message bruité reçu $\mathbf{X}$ mais sur la somme $\mathbf{X} + \mathbf{e}_i$ du message reçu et des erreurs insérées. Il est donc primordial que le résultat du décodage différentiel soit reconstitué à partir de cette somme combinée.

La recherche du motif comprend deux étapes :
- une comparaison de motif entre le terme $E_y + \hat{Y}_e$ de l'expression 2.22 et le motif théorique par une fenêtre glissante ;
- l'insertion d'une erreur supplémentaire à l'étage du treillis où le motif est détecté.

Ce processus est itéré jusqu'à la fin du codage.

En effet, cette insertion permet de modifier la valeur entrante d du terme $\mathbf{w}_{i-1}$ de l'expression 2.15 pour que l'état du codeur $\mathbf{s}_k$ puisse être ramené à l'état 0. Grâce à la périodicité du motif, il est alors possible de trouver un motif entre deux erreurs de transmission séparées dans un intervalle au moins égal à la période N_m. De plus, cette insertion n'introduit pas de dégradation en terme de performance, puisqu'elle est prise en compte durant la phase de reconstitution de la décision finale (opération ou-exclusif après le décodage Max-Log-MAP proprement dit).

2.2.3 Exemple de codage différentiel avec insertion de dummy-errors

Prenons un exemple pour illustrer le codage différentiel associé à la procédure d'insertion de dummy-errors. Nous considérons un codeur à 4 états associé aux polynômes générateurs $G[1,(1+D^2)/(1+D+D^2)]$. L'état du codeur est initialisé à l'état 0. Le message émis est représenté par le vecteur $\mathbf{x} = (101001010)$. Après le codage du $\mathbf{x}$ à l'émission, nous obtenons un mot codé $\mathbf{v} = (\mathbf{x}, \mathbf{y})$:

$$\begin{aligned} \mathbf{x} &= (101001010) \\ \mathbf{y} &= (110100000). \end{aligned}$$

La modélisation des problèmes de propagation au cours de transmission consiste à ajouter un couple d'erreurs $\mathbf{E} = (\mathbf{E}_x, \mathbf{E}_y)$ au mot $\mathbf{v}$. Nous obtenons un mot bruité $\mathbf{V} = (\mathbf{X}, \mathbf{Y})$:

$$\begin{aligned} \mathbf{X} &= (001001010) \\ \mathbf{Y} &= (110100000). \end{aligned}$$

Le ré-encodage s'applique au vecteur $\mathbf{X}$. Nous obtenons alors $\mathbf{V}' = (\mathbf{X}', \mathbf{Y}')$:

$$\begin{aligned} \mathbf{X}' &= (001001010) \\ \mathbf{Y}' &= (001111011). \end{aligned}$$

Ensuite, l'opération ou-exclusif $\bar{\mathbf{V}} = \mathbf{V}' \oplus \mathbf{V}$ donne :

$$\begin{aligned} \bar{\mathbf{X}} &= (000000000) \\ \bar{\mathbf{Y}} &= (111011011). \end{aligned}$$

Sur cet exemple, nous choisissons comme vecteur d'erreur $\mathbf{E}_x$ qui est appliqué à la partie systématique $\mathbf{x}$ du mot émis $\mathbf{v}$ est :

$$\mathbf{E}_x = (100000000).$$

Nous obtenons donc un mot codé $\hat{\mathbf{V}} = (\mathbf{E}_x, \hat{\mathbf{Y}}_\mathbf{e})$ lors du codage du vecteur $\mathbf{E}_x$:

$$\begin{aligned} \mathbf{E}_\mathbf{x} &= (100000000) \\ \hat{\mathbf{Y}}_\mathbf{e} &= (111011011). \end{aligned}$$

Nous obtenons les vecteurs :

$$\hat{\mathbf{Y}}_\mathbf{e} = \bar{\mathbf{Y}} = (111011011) \quad \text{et} \quad \mathbf{E}_y = (000000000) \tag{2.35}$$

Ainsi, le décodage à partir du mot $\bar{\mathbf{V}}$ donne un vecteur de décision $\hat{\mathbf{d}}'$:

$$\hat{\mathbf{d}}' = (100000000).$$

Le résultat final du décodage est obtenu à l'aide de l'opération ou-exclusif entre $\hat{\mathbf{d}}'$ et $\mathbf{X}$:

$$\hat{\mathbf{d}}' \oplus \mathbf{X} = \mathbf{x}$$

Pour cet exemple, le chemin survivant dans le treillis après ré-encodage a été représenté par le trait épais noir dans la figure 2.11.

L'apparition du motif périodique signifie la présence d'erreur, accompagnée d'un effet de divergence sur le chemin survivant à partir du chemin TAZ. Le motif de redondance

$M^r = (1 \to 1 \to 0)$ apparaît à partir du premier étage. Afin de supprimer ce motif pour que le chemin survivant puisse être ramené sur le chemin TAZ, il suffit d'insérer une erreur sur l'étage 1. Ainsi, le vecteur d'erreur insérée est $\mathbf{e}_i = (100000000)$. Comme le codage est linéaire, le chemin survivant après le ré-encodage à partir de la somme $\mathbf{e}_i \oplus \mathbf{E}_x$ est ramené sur chemin zéro, comme montré par la figure 2.12.

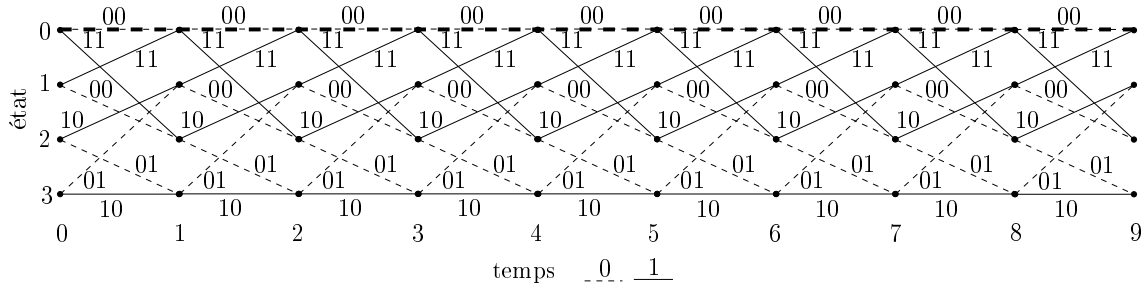


Figure 2.12 — Exemple de codage différentiel avec l'insertion de dummy-errors.

Dans ce cas, le ré-encodage est appliqué à l'expression :

$$\mathbf{X} \oplus \mathbf{e}_i = (101001010)$$

et le mot entrant dans le décodeur $\bar{\mathbf{V}} = (\bar{\mathbf{X}}, \bar{\mathbf{Y}})$ est :

$$\begin{aligned} \bar{\mathbf{x}} &= (100000000) \\ \bar{\mathbf{y}} &= (000000000), \end{aligned}$$

ce qui conduit à avoir un résultat de décodage :

$$\hat{\mathbf{d}}' = (000000000).$$

Le résultat final est obtenu après effectuer l'opération ou-exclusif entre $\mathbf{X} \oplus \mathbf{e}_i$ et $\hat{\mathbf{d}}'$:

$$\mathbf{X} \oplus \mathbf{e}_i \oplus \hat{\mathbf{d}}' = (101001010) = \mathbf{x}.$$

2.2.4 Critère d'estimation du motif répétitif appliqué à l'algorithme Max-Log-MAP

Sur l'exemple précédent, le vecteur d'erreur $\mathbf{E}_y$ est supposé comme un vecteur nul (voir l'équation 2.35). Or, cela est peu probable lors d'une transmission réelle. Comme le vecteur $\mathbf{E}_y$ a un impact direct sur l'apparition du motif après le ré-encodage, il se peut que le motif apparu ne corresponde pas exactement au motif théorique. C'est pourquoi nous mettons un critère d'estimation pour tenir compte l'impact du vecteur $\mathbf{E}_y$ lors de la détection du motif.

2.2.4.1 Mise en place d'un seuil d'estimation sur le motif répétitif

Reprenons l'expression 2.22,

$$\bar{\mathbf{V}}_k^N = \begin{pmatrix} 0 \\ E_{y,k} + \hat{Y}_{e,k} \end{pmatrix} \tag{2.36}$$

La valeur $\hat{Y}_{e,k}$ est bruitée par $E_{y,k}$. Par conséquent, le motif dans le vecteur $\hat{\mathbf{Y}}_\mathbf{e}$ peut être déformé car une erreur $E_{y,k}$ peut être ajoutée au motif M^r à l'instant donné. De même, le bruitage se produit sur le motif d'état M^e. Afin de tenir compte de cet ajout, un seuil S_m est mis en place pour estimer le motif apparu dans le terme $\bar{\mathbf{Y}} = \hat{\mathbf{Y}} + \mathbf{E}_y$. Ce seuil, mesuré par la distance de Hamming entre le motif théorique associé aux codes utilisés et les éléments du vecteur $\hat{\mathbf{Y}}$ dans une fenêtre glissante de longueur L_m, permet de repérer le motif dans le vecteur $\hat{\mathbf{Y}}$. La longueur L_m est appelée *longueur de motif*.

Fort heureusement, l'insertion de dummy-errors supplémentaires n'a pas d'impact sur les performances. La valeur appropriée du seuil S_m peut être obtenue par simulation logicielle selon l'environnement de transmission et la qualité de service souhaitée. Par convention, nous choisissons une valeur S_m permettant d'avoir un taux d'occupation maximal sur le chemin TAZ.

2.2.4.2 Taux d'occupation sur le chemin TAZ d'un turbo-décodeur appliquant le ré-encodage et l'insertion de dummy-errors

Dans cette sous-section, nous étudions le taux d'occupation T_o en fonction du seuil S_m pour un turbo-décodeur appliquant le ré-encodage et l'insertion de dummy-errors, noté TD-IE.

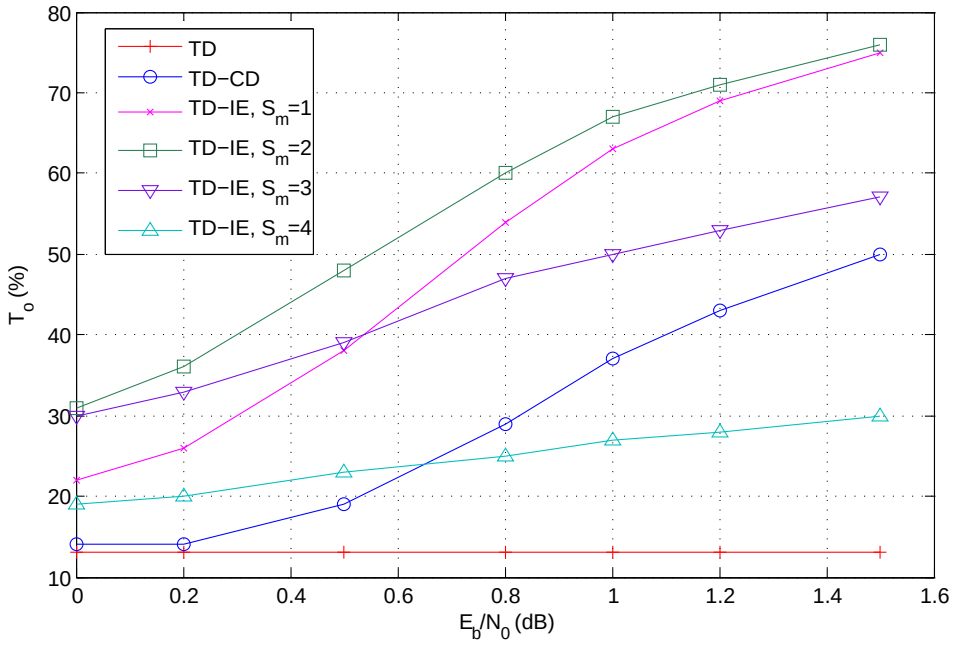


Figure 2.13 — Taux d'occupation T_o des turbo-décodeurs (TD, TD-CD et TD-IE) adaptés à la norme UMTS en fonction du seuil S_m.

Tout d'abord, nous avons mis en oeuvre un turbo-décodeur retenu dans le standard UMTS. L'environnement de simulation est le suivant. Le nombre d'itérations N_{it} est fixé à 6 sans mise en place de critère d'arrêt. La taille de trame L est égale à 864 bits. Les données émises x, y_1 et y_2 sont respectivement quantifiées sur 5 bits dont 3 bits sont pour la partie fractionnaire. Cette quantification a pour syntaxe $Q(q_t, q_f)$, où q_t est le nombre total des bits et q_f le nombre de bits pour la partie fractionnaire. L'information extrinsèque z est quantifiée sur le format $Q(5,3)$ pour les deux premières itérations et $Q(7,3)$ pour les autres itérations. La

quantification de l'information est précédée de la phase de saturation pour limiter la plage des valeurs. Le coefficient de pondération ζ de l'information z [Boutillon 07] est égal à 0.5 pour les deux premières itérations, 1 pour la dernière itération, et 0.75 pour les autres. Ce contexte est valable par défaut pour les simulations suivantes.

La figure 2.13 présente les taux d'occupation T_o des turbo-décodeurs (TD, TD-CD et TD-IE) adaptés à la norme UMTS en fonction du seuil S_m. Le nombre d'itérations est fixé à 6. La valeur du paramètre L_c de l'expression 2.28 est égale à 7 dans notre cas. Nous constatons que les taux T_o varient selon les valeurs du rapport SNR et du seuil S_m et les taux T_o augmentent lorsque la valeur du seuil S_m diminue. Par exemple, nous obtenons un taux stable de 13% pour le turbo-décodeur classique. Cela peut s'interpréter par le fait que les chemins survivants du treillis pour un turbo-décodeur classique sont repartis de manière équiprobable au sein du décodage. En revanche, ce taux atteint 50% lorsque le rapport SNR est égal à 1.5dB pour le décodeur TD-CD. De plus, les taux de 19% et de 30% sont respectivement obtenus pour des rapports SNR de 0dB et de 1.5dB lorsque le seuil est égal à 4. Nous constatons que la pente de variation pour la courbe $S_m = 4$ est très faible. Dans ce cas, l'insertion d'erreurs ne porte pas beaucoup d'intérêt que ce soit la valeur du rapport SNR. Par contre, nous obtenons un taux maximal T_o de 76% lorsque $S_m = 2$ et SNR= 1.5dB grâce à l'insertion de dummy-errors.

Pour conclure, avec l'application du codage différentiel, l'insertion de dummy-errors augmente le taux d'occupation du chemin survivant sur le chemin TAZ de manière significative. La courbe $S_m = 2$ avec une pente forte est plus avantageuse par rapport aux autres courbes en terme de taux d'occupation T_o. Nous choisissons donc un seuil égal à 2 dans la suite de la thèse.

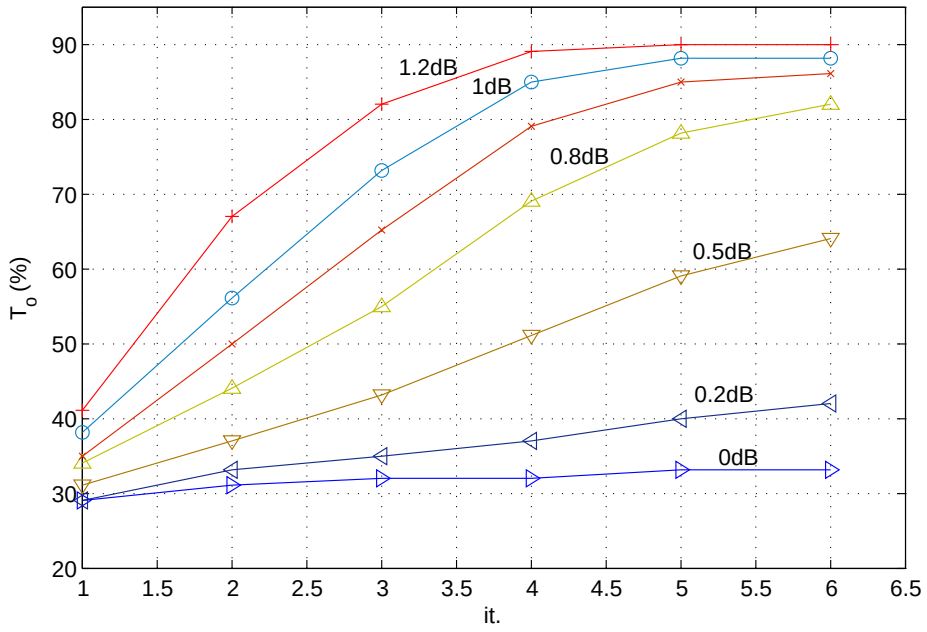


Figure 2.14 — Taux d'occupation T_o en fonction de l'itération du décodage.

La figure 2.14 montre les taux d'occupation T_o en moyenne en fonction de l'itération pour différentes valeurs de SNR. L'axe des abscisses indique le numéro l'itération tandis que l'axe des ordonnées représente le taux d'occupation T_o. Lorsque la qualité de transmission est mauvaise, *i.e.* SNR = 0dB, il est peu probable que le chemin survivant correspond au

chemin TAZ malgré l'application du codage différentiel et de l'insertion de dummy-errors. Nous obtenons un taux T_o égal à 31% pour la $2^{\text{ème}}$ itération lorsque le rapport SNR vaut 0dB. Mais ce taux peut atteindre 90% lorsque les données sont peu bruitées, comme le montre la courbe avec un SNR = 1.5dB à la $6^{\text{ème}}$ itération. Le taux d'occupation s'accroît au fur et à mesure de manière significative suivant la qualité de transmission et l'itération en cours.

2.2.5 Activité d'un turbo-décodeur avec ré-encodage et insertion de dummy-errors

SNR (dB)	0	0.2	0.5	0.8	1	1.2	1.5
$\mathbf{a}_0$ (TD)	19	20	21	22	23	23	23
$\mathbf{a}_2$ (TD-IE)	18	19	20	20	21	21	21
D_{a2}	5%	5%	5%	9%	9%	9%	9%

Tableau 2.2 — Comparaison d'activité des turbo-décodeurs (TD et TD-IE) adapté à la norme UMTS avec $S_m = 2$.

Le tableau 2.2 donne les activités en moyenne des turbo-décodeurs (TD et TD-IE) adaptés à la norme UMTS. Pour un turbo-décodeur TD-IE, son activité et la diminution d'activité correspondante sont respectivement notées $\mathbf{a}_2$ et D_{a2}. A partir des tableaux 2.1 et 2.2, l'activité du turbo-décodage différentiel est diminuée de 23 à 22 par rapport au turbo-décodage classique lorsque SNR = 1.2dB. Cette activité diminue encore lorsque l'insertion d'erreurs est appliquée. Une activité de 21 est obtenue dans ce même cas. En prenant l'activité du TD comme référence, nous constatons une diminution d'activité de 5% pour les SNRs allant de 0dB à 0.5dB. Cette diminution atteint 9% pour les SNRs allant de 0.8dB à 1.5dB. Pour conclure, l'insertion de dummy-errors ne réduit pas l'activité du turbo-décodeur TD-IE de manière significative par rapport au turbo-décodeur classique. Mais, l'insertion de dummy-errors augmente la probabilité que le chemin survivant soit ramené sur le chemin TAZ.

Une question s'est posée : est-il possible de réduire l'activité en explorant cette propriété ? Pour répondre à cette question, nous proposons une nouvelle approche permettant de réduire le nombre des accès mémoires dans la section suivante, d'où une diminution potentielle d'activité du système.

2.3 Calcul anticipé de l'information extrinsèque

Grâce à la suppression du motif répétitif, le chemin survivant peut être ramené au maximum sur le chemin TAZ. En parallèle, le codage différentiel nous donne un mot de code dont la partie systématique s'annule (voir l'expression 2.13). En considérant ces propriétés, une question s'est posée : comment pouvons-nous profiter du chemin TAZ avec un mot entrant qui contient une partie nulle. En effet, à l'instant donné, lorsque le chemin survivant est ramené sur le chemin TAZ, il est alors possible de déduire que le chemin survivant poursuit toujours le chemin TAZ si les séquences suivantes ne sont composées que des éléments nuls. Dans ce cas, nous pouvons alors calculer l'information extrinsèque z lors du premier parcours du treillis. Cette technique est appelée *calcul anticipé* sur l'information extrinsèque. Ceci permet alors de diminuer de façon significative le nombre d'accès mémoire au cours du décodage.

Dans cette section, nous détaillons d'abord le principe de cette approche. Elle est ensuite appliquée à un turbo-décodeur du système UMTS. Enfin, nous montrons les résultats de

simulation à propos des performances et des gains en terme de diminution d'accès mémoire. Les principales caractéristiques du système, comme l'information mutuelle et les corrélations des informations extrinsèques, sont ensuite étudiées par rapport à un turbo-décodeur classique.

2.3.1 Information extrinsèque explicite calculée à partir de l'algorithme Max-Log-MAP

L'information extrinsèque z_k à l'instant k est calculée d'après l'expression 2.37 :

$$z(k) = \frac{1}{2}\left(\min_{s' \underset{\rightarrow}{0} s}(\alpha_{k-1}^{s'} + \gamma_{y_k}^{s' \rightarrow s} + \beta_k^s) - \min_{s' \underset{\rightarrow}{1} s}(\alpha_{k-1}^{s'} + \gamma_{y_k}^{s' \rightarrow s} + \beta_k^s)\right) \tag{2.37}$$

où le terme $s' \underset{\rightarrow}{0} s$ représente la transition de l'état s' à l'état s associée à une entrée binaire 0. Le terme $\gamma_{y_k}^{s' \rightarrow s}$ représente la métrique de branche de la transition $(s' \rightarrow s)$ prenant en compte uniquement la contribution de la redondance instantanée y_k. Les termes α et β correspondent, quant à eux, aux métriques de nœud des sens *Aller* et *Retour*. Enfin, le facteur 1/2 permet d'obtenir une information extrinsèque sortante à la même échelle que les informations entrantes [Berrou 07]. En faisant l'hypothèse que

$$\min(\beta_k^s) = \min(\beta_k^0) \text{ et que } \beta_k^s >> \beta_k^0 \text{ pour } s \neq 0 \tag{2.38}$$

Alors, l'expression 2.37 peut être simplifié comme suit :

$$z(k) = \frac{1}{2}\left(\min_{s' \underset{\rightarrow}{0} s}(\alpha_{k-1}^{s'} + \gamma_{y_k}^{s' \rightarrow s}) - \min_{s' \underset{\rightarrow}{1} s}(\alpha_{k-1}^{s'} + \gamma_{y_k}^{s' \rightarrow s})\right) \tag{2.39}$$

Sous cette hypothèse, l'information extrinsèque z ne dépend alors que des valeurs des métriques de nœud du sens *Aller* et peut donc être calculée de façon anticipée. Nous allons déterminer, dans la section suivante, une condition permettant de garantir l'hypothèse 2.38 avec une grande fiabilité.

2.3.2 Information extrinsèque anticipée

La condition 2.38 est vérifiée si les symboles reçus $(X_{k+i}, Y_{k+i})|_{i=0 \cdots L_o-1}$ sont très proches en distance euclidienne (distance quadratique) du chemin TAZ. Par souci de simplification, nous considérons la distance de Hamming. D'après l'expression 2.40 :

$$\mathbf{s}_{k+1} = \mathbf{A} \cdot \mathbf{s}_k + \mathbf{w}_k \tag{2.40}$$

avec $\mathbf{w}_k = \begin{pmatrix} d_k \\ 0 \end{pmatrix}$, l'état du codeur s_k à l'instant k dépend de l'état précédent s_{k-1} et du vecteur d'entrée $\mathbf{w}_k$. Si $s_{k-1} = 0$ et une séquence d'entrée $d_k = 0$ pour $k = k, \cdots, k + L_o - 1$, le chemin du parcours du treillis poursuit alors le chemin TAZ. Dans ce cas, nous proposons une méthode appelée *calcul anticipé* de l'information extrinsèque z (la figure 2.15). Le turbo-décodeur appliquant cette approche est noté *TD-CA*.

Les conditions d'anticipation à l'instant k sont les suivantes (voir la figure 2.15) :
- la partie systématique $\bar{X}$ à l'entrée du décodeur est égale à la valeur ferme 0 ;
- la valeur minimale des métriques de nœud α_{k-1} est pour l'état 0. Cela signifie que la condition $\min(\alpha_{k-1}^i) = \alpha_{k-1}^0$ pour $i \in \{0, \cdots, N_e - 1\}$ est vérifiée ;

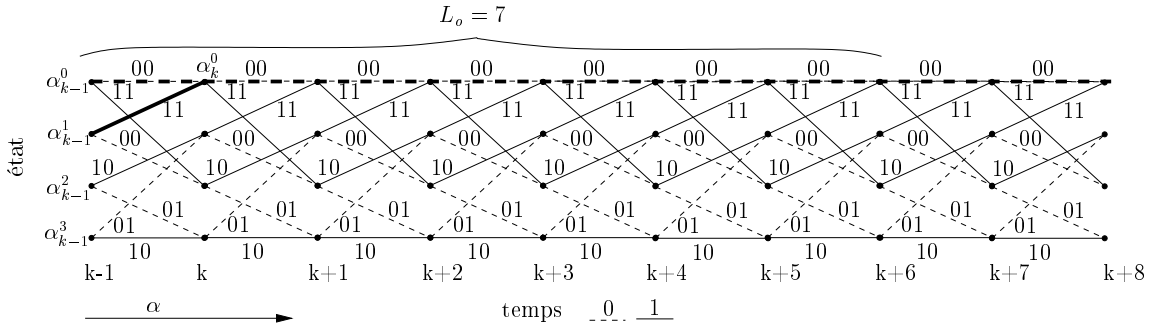


Figure 2.15 — Calcul anticipé de l'information extrinsèque. La longueur d'observation L_o est égale à 7.

- la distance de Hamming entre les L_o séquences suivantes et des séquences nulles est inférieure à un seuil S_{hd}, dont la valeur peut être obtenue de manière empirique par simulation logicielle selon les performances fixées.

Lorsque ces conditions sont réunies, l'information extrinsèque anticipée z_a est déterminée à partir du chemin TAZ et du chemin concurrent par l'expression suivante :

$$z_a(k) = \frac{1}{2}((\alpha^0_{k-1} + Y_k) - (\alpha^1_{k-1} - Y_k)) \tag{2.41}$$

Comme la valeur minimale des métriques de nœud α se trouve sur l'état 0 à l'instant donné, la métrique de nœud α^0_{k-1} peut se normaliser à $\alpha^0_{k-1} = 0$ [Boutillon 07]. L'expression 2.41 est simplifiée comme suit :

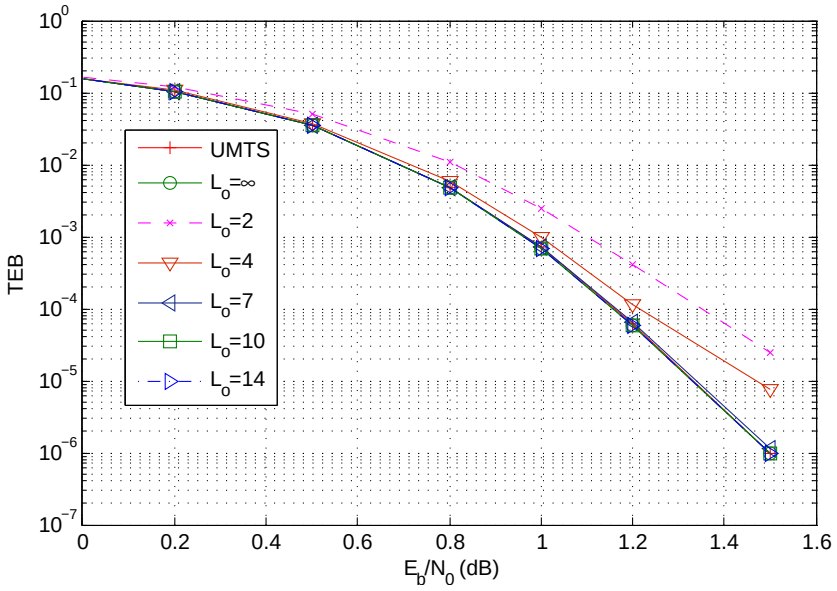
$$z_a(k) = Y_k - \frac{1}{2}\alpha^1_{k-1} \tag{2.42}$$

L'expression 2.42 nous montre que la valeur z_a ne dépend que de l'entrée Y_k et de la valeur α^1_{k-1}. Lorsque l'information extrinsèque z est calculée sans connaître les métriques de nœud β, il n'est pas nécessaire d'accéder à la mémoire dans laquelle les valeurs α sont stockées. Cette approche permet alors de diminuer le nombre d'accès mémoire et par conséquent la consommation dynamique de la mémoire de l'architecture associée.

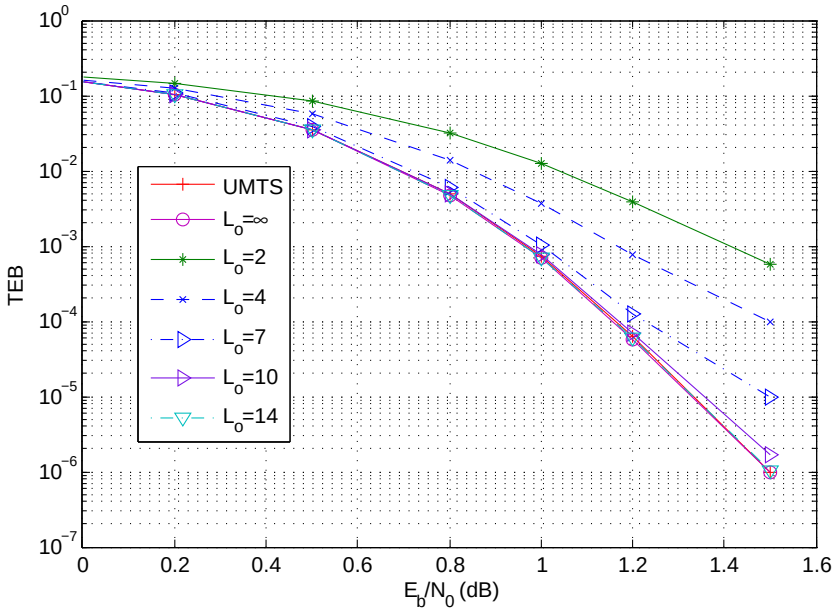
2.3.3 Impact de la méthode *calcul anticipé* sur les performances

Le calcul anticipé de l'information extrinsèque peut avoir un impact sur les performances du décodage. En effet, les expressions 2.37 et 2.42 ne produisent pas systématique les mêmes informations extrinsèques. Par conséquent, l'information anticipée z_a peut être différente de l'information extrinsèque z à l'instant donné. Or, les performances du décodeur Max-Log-MAP dépendent de la fiabilité de l'information extrinsèque venant du décodage précédent. La différence entre les informations z et z_a peut donc entraîner des dégradations en terme de performances.

Rappelons qu'un seuil S_m pour la détection du motif est fixé à 2. Dans ce cas, L'anticipation du calcul de l'information extrinsèque dépend de deux facteurs clefs : la longueur d'observation L_o et le seuil S_{hd} sur la distance de Hamming. La figure 2.16 présente la dégradation en terme de performance en fonction du couple (L_o, S_{hd}). La courbe $L_o = \infty$ représente les performances de décodage sans application du calcul anticipé. Nous constatons une dégradation de 0.1dB pour un TEB de l'ordre de 10^{-5} lorsque les paramètres sont $(L_o, S_{hd}) = (2, 1)$ et plus de 0.3dB de dégradation pour les paramètres $(L_o, S_{hd}) = (2, 2)$. En



(a) : $S_{hd} = 1$



(b) : $S_{hd} = 2$

Figure 2.16 — Performances avec application du calcul anticipé de l'information extrinsèque pour (a) : $S_{hd} = 1$ et (b) : $S_{hd} = 2$.

revanche, peu de dégradations sont constatées lorsque les paramètres sont $(L_o, S_{hd}) = (7, 1)$ ou $(L_o, S_{hd}) = (10, 2)$.

En conclusion, la dégradation des performances peut être maîtrisée lorsque la longueur d'observation L_o et le seuil S_{hd} sont bien définis. En pratique, nous choisissons les valeurs respectives pour (L_o, S_{hd}) de telle manière que nous puissions appliquer au maximum la méthode *calcul anticipé* au cours du décodage et la dégradation associée des performances est tolérable selon l'application. Il est donc nécessaire de faire un compromis entre les performances et les paramètres (L_o, S_{hd}).

2.3.4 Etude sur des diminutions en nombre d'accès mémoire et des dégradations des performances

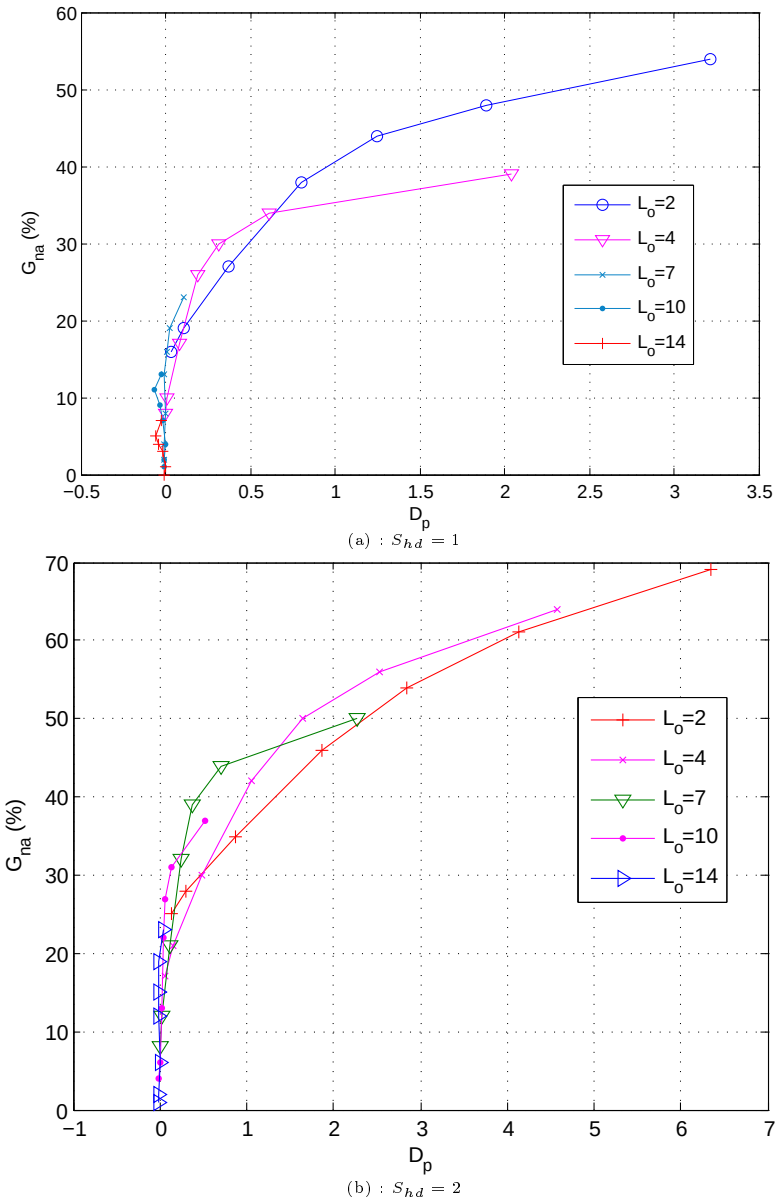


Figure 2.17 — Taux de diminution en nombre d'accès mémoire en fonction des dégradations des performances pour 6 itérations dans un turbo-décodeur TD-CA pour des valeurs $S_{hd} = 1$ (a) et $S_{hd} = 2$ (b).

Nous définissons le taux de diminution G_{na} en nombre d'accès mémoire comme suit :

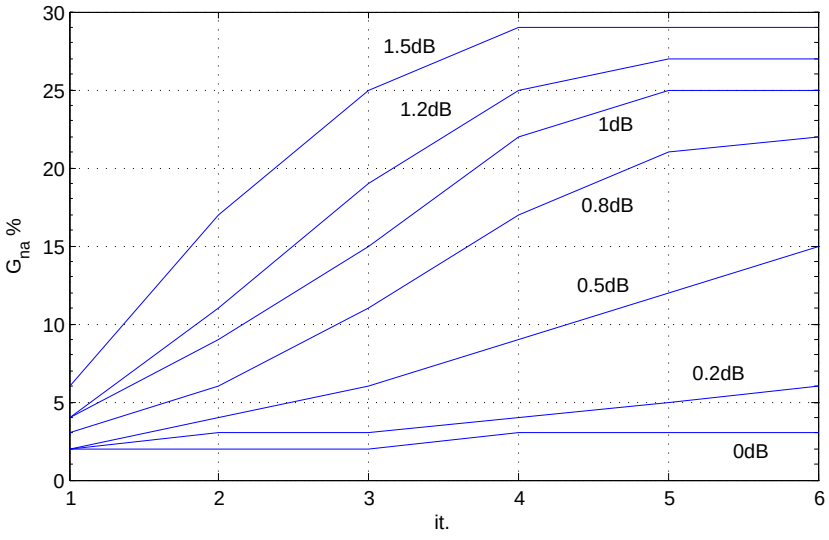
$$G_{na} = 1 - \frac{N_d}{N_s} \tag{2.43}$$

où N_s et N_d représente respectivement le nombre d'accès mémoire dans le système standard et celui dans le système différentiel avec application de la méthode *calcul anticipé*. Le degré de dégradation des performances D_p est défini par le logarithme du rapport TEB$'$ modifié sur

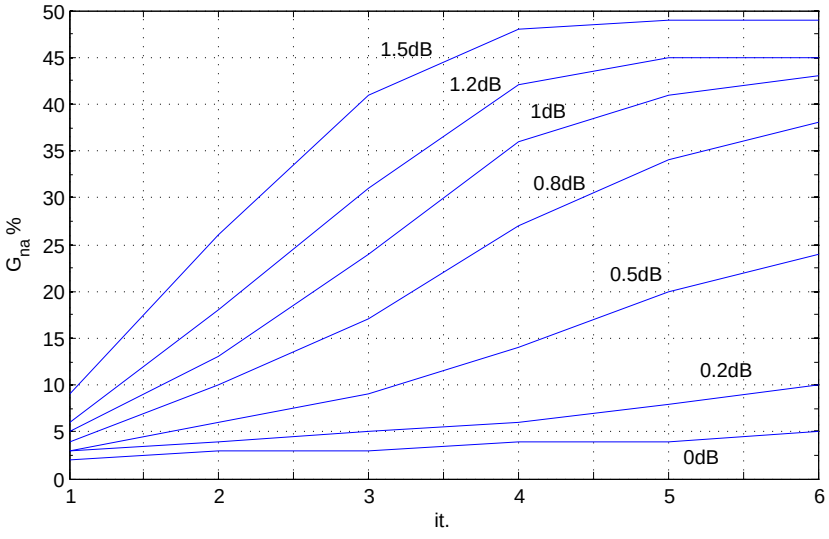
TEB standard dans un système UMTS pour un canal ABBG ayant la même variance, soit :

$$D_p = \log \frac{\text{TEB}'/\text{SNR}}{\text{TEB}/\text{SNR}} = \log \frac{\text{TEB}'}{\text{TEB}} \tag{2.44}$$

Les figures 2.17(a) et 2.17(b) illustrent respectivement les taux G_{na} obtenus au cours de la $6^{\text{ème}}$ itération pour $S_{hd} = 1$ et $S_{hd} = 2$ en fonction du degré de la dégradation des performances. Nous ne constatons pas une dégradation significative lorsque le paramètre L_o est suffisamment élevé. Lorsque les paramètres sont $(L_o, S_{hd}) = (7, 1)$, un taux de 24% est obtenu associé à un degré de dégradation des performances $D_p = 0.12$ (voir la figure 2.17(a)). De même, un taux de 38% est obtenu lorsque les paramètres sont $(L_o, S_{hd}) = (14, 2)$ pour un degré $D_p = 0.5$ (voir la figure 2.17(b)). La figure 2.17 montre que le taux G_{na} diminue lorsque la longueur d'estimation L_o augmente.



(a) : $(L_o, S_{hd}) = (7, 1)$



(b) : $(L_o, S_{hd}) = (10, 2)$

Figure 2.18 — Taux de diminution G_{na} en fonction de l'itération au cours pour deux configurations : (a) $(L_o, S_{hd}) = (4, 1)$ et (b) $(L_o, S_{hd}) = (10, 2)$.

Concernant les taux de diminution au cours du processus de décodage itératif, les figures 2.18(a) et 2.18(b) donnent respectivement les taux de diminution pour deux configurations : $(L_o, S_{hd}) = (7, 1)$ et $(L_o, S_{hd}) = (10, 2)$. Associé au *calcul anticipé* de l'information extrinsèque, le codage différentiel et l'insertion des erreurs n'ont pas un impact significatif sur le taux G_{na} lors des premières itérations avec un faible SNR. Par exemple, nous obtenons seulement des diminutions de 2% et de 3% pour $(L_o, S_{hd}) = (4, 1)$ et $(L_o, S_{hd}) = (10, 2)$ au cours de la deuxième itération lorsque le rapport SNR vaut 0dB. En revanche, ce taux augmente de manière significative en fonction de l'itération de décodage. Ainsi, nous obtenons des taux très importants lorsque le rapport SNR est élevé en particulier au cours des dernières itérations. Par exemple, lorsque le rapport SNR vaut 1.5dB, des taux de 29% et de 49% sont respectivement obtenus pour $(L_o, S_{hd}) = (7, 1)$ et $(L_o, S_{hd}) = (10, 2)$. Pour obtenir un gain significatif, nous choisissons donc le couple $(L_o, S_{hd}) = (10, 2)$ pour la suite du document. Pour conclure, la méthode *calcul anticipé* de l'information extrinsèque est très efficace pour diminuer le nombre d'accès mémoire lors des dernières itérations de turbo-décodage sur un SNR approprié.

2.3.5 Etude de l'information mutuelle en fonction de la longueur d'observation L_o

En considérant qu'une distribution de probabilité représente la connaissance sur une réalisation d'une variable aléatoire, l'entropie de cette distribution quantifie alors l'absence de cette distribution. L'information mutuelle s'exprime de la manière suivante :

$$\begin{aligned} I_m(Z, Z_a) &= H(Z) - H(Z|Z_a) \\ &= H(Z_a) - H(\bar{Z}|Z) \\ &= H(Z) + H(Z_a) - H(Z, Z_a) \end{aligned} \tag{2.45}$$

où le symbole $H(C)$ représente l'entropie de la variable C. L'information mutuelle mesure la quantité d'information apportée en moyenne par une réalisation de la variable Z sur celle de la variable Z_a. Si $I_m(Z, Z_a) = 1$, la relation linéaire entre Z_a et Z peut être exprimée par $Z_a = f(Z)$, où $f(Z)$ est une fonction linéaire de la réalisation Z.

La figure 2.19 représente l'évolution de l'information mutuelle I_m par rapport au paramètre L_o au cours de la $3^{\text{ème}}$ itération dans un système UMTS différentiel. Cette mesure est faite pour tous les points dont les informations extrinsèques sont traitées par la méthode *calcul anticipé*, c'est à dire, les points sur lesquels les conditions suivantes sont réunies :

- le minimum des métriques de nœud instantanées est sur l'état 0 ;
- les L_o séquences suivantes sont des séquences nulles.

L'information mutuelle s'accroît asymptotiquement vers 1 lorsque la valeur de L_o augmente vers l'infinie. En effet, l'information extrinsèque Z_a est égale à Z dans ce dernier cas. Plus la distance L_o est importante, *i.e.*, $L_o = 10$, plus l'information I_m est élevée. Nous constatons que l'information mutuelle atteint 1 sur la courbe $S_{hd} = 1$ lorsque la longueur L_o vaut 10. Par ailleurs, la pente de convergence de la courbe $S_{hd} = 1$ est supérieure à celle de la courbe $S_{hd} = 2$. Moins le seuil S_{hd} est élevé, plus probable le chemin survivant est sur le chemin TAZ si la condition ci-dessus est vérifiée, plus les informations Z et Z_a sont corrélées.

2.3.6 Activité d'un turbo-décodeur appliquant la méthode *calcul anticipé*

Lorsque l'information extrinsèque est anticipée au cours du parcours d'un treillis, il n'est pas nécessaire de sauvegarder les métriques de noeud α, d'où une réduction d'accès mémoire.

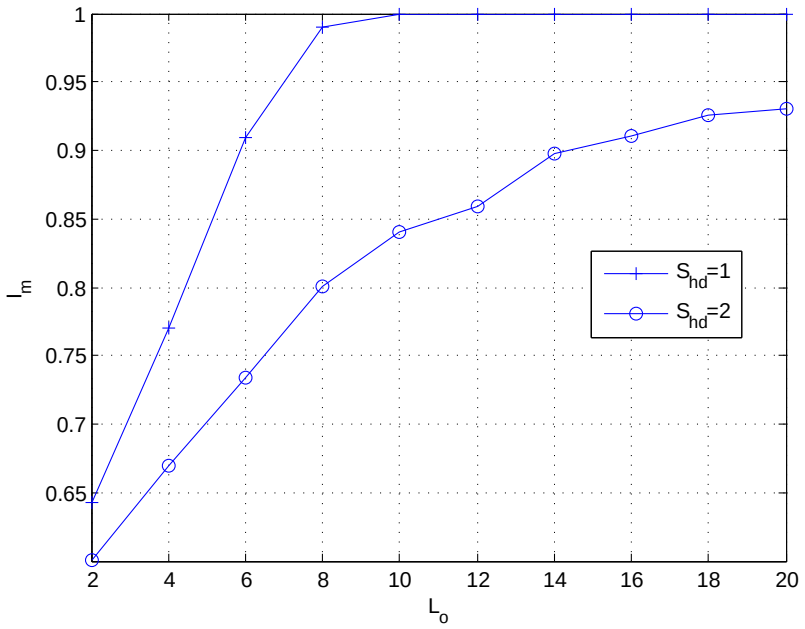


Figure 2.19 — Evolution de l'information mutuelle I_m par rapport au paramètre L_o au cours de la $3^{\text{ème}}$ itération pour un rapport SNR = 1.2dB.

Dans ce cas-la, nous considérons le nombre de transitions binaires égal à 0 (pas d'activité sur la mémoire).

Les activités du décodeur TD-CA sont présentées dans le tableau 2.3. L'activité respective et la diminution correspondante sont respectivement notées $\mathbf{a}_3$ et D_{a3}. Par rapport au turbo-décodeur classique, l'activité du TD-CA est diminuée de manière significative. Nous obtenons une réduction de 5% au niveau activité pour un faible SNR = 0dB. Cette réduction atteint 43% lorsque le SNR vaut 1.5dB.

SNR (dB)	0	0.2	0.5	0.8	1	1.2	1.5
$\mathbf{a}_0$ (TD)	19	20	21	22	23	23	23
$\mathbf{a}_1$ (TD-CA)	18	18	17	16	15	14	13
D_{a3}	5%	10%	19%	27%	35%	39%	43%

Tableau 2.3 — Comparaison des activités des turbo-décodeurs (TD et TD-CA).

Grâce à la méthode *calcul anticipé*, l'activité du système est diminuée de manière significative. Par ailleurs, cette réduction est fonction du rapport SNR. Lorsque le rapport SNR est élevé, il y a moins d'erreurs présentes dans la transmission. Dans ce cas, la probabilité que la méthode *calcul anticipé* puisse être appliquée au cours du décodage est très grande. C'est pourquoi nous pouvons obtenir une réduction très importante au niveau d'accès mémoire si les erreurs sont peu nombreuses.

2.3.7 Etude de la corrélation entre l'extrinsèque anticipée et l'extrinsèque explicite

Les méthodes de calcul des informations extrinsèques z_a et z sont différentes. C'est pourquoi certaines valeurs des informations z_a et z peuvent être différentes. La figure 2.20 présente

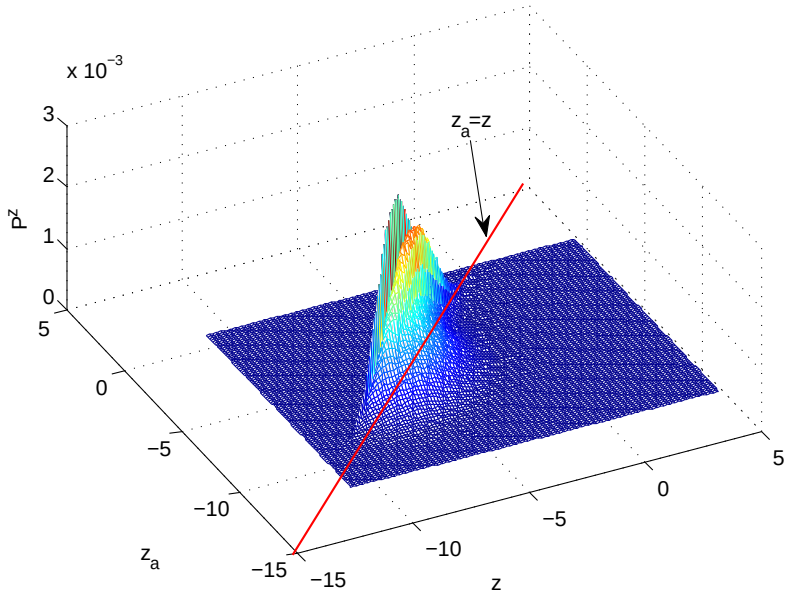


Figure 2.20 — Corrélation entre les informations extrinsèques z et z_a au cours de la $3^{\text{ème}}$ dans un turbo-décodeur différentiel. Les paramètres de simulation sont $(L_o, S_{hd}) = (10, 2)$. Le rapport SNR vaut 1.2dB.

la probabilité conditionnelle $P^z(z_a/z)$ (l'axe des ordonnées) en fonction de l'information z_a et de l'information z au cours de la $3^{\text{ème}}$ itération pour un rapport SNR = 1.2dB dans un système de décodage différentiel. Les paramètres sont $(L_{ei}, S_{hd}) = (10, 2)$. La droite $z = z_a$ signifie que les deux valeurs z_a et z sont égales.

La figure 2.20 montre que la plupart des valeurs z sont situées au-dessus de la droite $z_a = z$. Ceci veut dire que nous obtenons une probabilité plus élevée de la décision, c'est-à-dire, l'amplitude de l'information extrinsèque, lors du calcul anticipé par rapport au calcul explicite. Nous pouvons alors écrire l'expression suivante :

$$P(|z_a| \leq |z|) \gg P(|z_a| > |z|) \tag{2.46}$$

La probabilité maximale $P^z_{\max}$ vaut 2.8×10^{-3} lorsque les valeurs z_a et z sont respectivement égales à -4.6 et -5.6. Comme le calcul anticipé de l'information extrinsèque sous-estime la probabilité de décision, il est possible de multiplier la valeur z^a par un coefficient supérieur à 1 de telle manière à maximiser la probabilité $P^z(z^a = z)$. Or, la condition mise en place pour évaluer l'information extrinsèque limite le fait que la décision prise soit erronée. Dans ce cas, une probabilité sous-estimée de la décision n'a pas d'impact significatif sur les performances de décodage. D'autre part, le décodage itératif est robuste face aux imprécisions. Une estimation erronée due à la valeur z_a ne modifie guère les performances globales au cours du décodage. C'est pourquoi nous pouvons maîtriser les performances en définissant la condition d'estimation, en particulier sur le couple des paramètres (L_o, S_{hd}).

2.4 Conclusions

Pour maîtriser la consommation d'un système de turbo-décodage, une des méthodes efficaces consiste à diminuer l'activité globale du système. Pour tenter de favoriser cette diminution d'activité, une approche au niveau algorithmique reposant sur un système de décodage

différentiel est détaillée dans ce chapitre. Dans un premier temps, le principe général du codage différentiel est introduit. Puis, cette technique est appliquée pour un décodeur de Viterbi sur une transmission de canal symétrique. Les études des performances de décodage montrent que l'application de codage différentiel n'entraîne aucune perte au niveau des performances. De plus, le taux d'occupation sur le chemin TAZ atteint 71% pour un taux d'erreur binaire de l'ordre de 10^{-6}.

Dans un second temps, cette technique est étendue à un décodeur associé à l'algorithme Max-Log-MAP. En particulier, l'application de codage différentiel pour un turbo-décodeur du standard UMTS est traitée. Puisque le décodage de système différentiel consiste à considérer les vecteurs d'erreur, le codage différentiel dépend fortement de la quantité d'erreurs à l'entrée du décodeur. .

Notre étude démontre l'existence de motifs périodiques au cours du processus de décodage. L'inconvénient majeur de ces motifs est de maintenir une activité permanente dans le décodeur. C'est pourquoi, nous avons proposé une méthode d'insertion de dummy-errors supplémentaires afin d'éliminer ces motifs tout en ramenant le chemin survivant au chemin TAZ. Cette insertion peut se faire de manière dynamique au cours du ré-encodage sans aucun impact sur les performances de décodage. Grâce à cette insertion, le taux d'occupation sur le chemin TAZ peut atteindre 76% pour un TEB de l'ordre de 10^{-6}. De plus, il est possible de calculer l'information extrinsèque lors du parcours du treillis dans le sens *Aller* sans connaître les valeurs des métriques de nœud β dans le sens *Retour*. Cette caractéristique favorise donc la diminution des accès mémoire dans l'architecture de turbo-décodage.

Afin de valider cette approche, un certain nombre d'études ont été présentées dans la dernière partie de ce chapitre. Ainsi, l'information mutuelle et la corrélation entre l'information extrinsèque calculée et l'information extrinsèque anticipée sont discutées. En fixant des valeurs appropriées pour les paramètres de seuils, la dégradation des performances obtenue est peu significative de l'ordre de 0.1dB. Mais nous obtenons un taux de diminution d'accès mémoire de 49% et une réduction d'activité de 43% sans avoir de dégradation majeure des performances pour un TER de l'ordre de 10^{-5}. L'architecture du codage différentiel dédié au système UMTS sera présentée dans le chapitre 4.

CHAPITRE

3 Etude sur la quantification des données au sein du turbo-décodeur

DANS ce chapitre, nous proposons des solutions architecturales visant à diminuer la consommation d'un turbo-décodeur, et en particulier la consommation des mémoires. Notre recherche est focalisée sur la quantification des données, et plus spécifiquement sur celle des métriques de nœud au cours du décodage. Le décodeur élémentaire SISO (Soft Input Soft Output) utilise l'algorithme Max-Log-MAP dans notre étude. Pour ce faire, nous proposons une nouvelle méthode permettant de diminuer la taille des mémoires, qui entraîne une diminution de consommation. La méthode a été présentée au cours de la conférence SIPS2007 [Liu 07]. Elle consiste à saturer les métriques de nœud tout en maîtrisant la dégradation des performances.

Une architecture dédiée à l'algorithme Max-Log-MAP est présentée dans la section 3.1.1. Cette architecture autorise l'utilisation de la technique *sliding windows* [Raouafi 99], qui permet de commencer le décodage *Retour* sans attendre la fin du décodage *Aller*. Grâce à la technique *sliding windows*, le processus de décodage est accéléré et la taille des mémoires est diminuée. Le problème de la normalisation des métriques de nœud est ensuite abordé dans la section 3.1.2. Puis, nous proposons une nouvelle méthode basée sur la saturation des métriques de nœud dans la section 3.2. Cette méthode permet de diminuer le nombre de bits nécessaires à la quantification des métriques de nœud sans avoir une perte importante des performances en terme de TEB (Taux d'Erreur Binaire). Au niveau de la mise en œuvre, deux possibilités sont envisagées pour appliquer notre approche : Saturation A l'Extérieur (SAE) et Saturation A l'Intérieur (SAI) du calcul récursif. Les architectures associées sont respectivement détaillées dans les sections 3.2.1 et 3.2.2. La section 3.2.3.2 donne une comparaison des performances en terme de TEB pour un turbo-décodeur dédié à la norme UMTS.

La consommation due à la mémoire est modélisée dans la section 3.3. Tout d'abord, l'outil d'estimation CACTI est présenté dans la section 3.3.1. Ensuite, une architecture générique du système microélectronique est donnée dans la section 3.3.2. Cette architecture comprend une mémoire principale, une mémoire cache et un microprocesseur. A propos de la mémoire cache, son principe de fonctionnement est rappelé dans la section 3.3.3. Dans la section 3.3.4, les mémoires nécessaires utilisées dans un turbo-décodeur sont détaillées. Enfin, les surfaces des mémoires et l'énergie consommée par un accès mémoire sont respectivement estimées à l'aide de l'outil CACTI. Les résultats d'estimation sont alors présentés dans la section 3.3.5.

3.1 Architecture d'un décodeur SISO associé à l'algorithme Max-Log-MAP

Dans cette section, nous allons dans un premier temps présenter l'architecture d'un décodeur SISO utilisant l'algorithme Max-Log-MAP. La technique *sliding windows* est utilisée pour réduire la latence de décodage et la taille de mémoire. Dans un second temps, nous abordons les différentes techniques existantes pour la normalisation des métriques de nœud.

3.1.1 Architecture dédiée à l'algorithme Max-Log-MAP

L'algorithme Max-Log-MAP a été détaillé au niveau algorithmique dans le chapitre 1. Nous en faisons ici un bref rappel et présentons une architecture associée. L'algorithme de décodage Max-Log-MAP est organisé de la manière suivante. Les valeurs des mots transmis $\mathbf{v}(x, y)$ sont prises dans un alphabet $A = \{0, 1\}$. Le décodeur reçoit une séquence de L mots bruités $\mathbf{V}(X, Y)$ à travers un canal ABBG (Additif Bruit Blanc Gaussien). Le traitement revient à considérer un treillis de L étages. Dans ce cas, la métrique de branche γ_k à l'instant k peut être calculée comme suit [Pietrobon 96] :

$$\gamma_k(s', s) = -V_k \cdot c_k(s', s), \tag{3.1}$$

où $c_k(s', s)$ est le mot de code représentant la branche de l'état s' à l'état s dans un treillis.

L'algorithme se décompose en trois étapes :

1. calcul récursif de α. La métrique de nœud α du sens *Aller* est calculée de façon récursive à partir de l'expression suivante :

$$\alpha_k(s) = \min_{(s',s)}(\alpha_{k-1}(s') + \gamma_k(s', s)),\ k = 0, \cdots, L-1. \tag{3.2}$$

   Les valeurs α sont ensuite sauvegardées dans une mémoire dédiée.

2. calcul récursif de β. La métrique de nœud β du sens *Retour* est obtenue à partir de l'expression suivante :

$$\beta_k(s') = \min_{(s,s')}(\beta_{k+1}(s) + \gamma_k(s', s)),\ k = L-1, \cdots, 1. \tag{3.3}$$

3. calcul du LRV (Logarithme du Rapport de Vraisemblance) $L(x_k)$. Il est obtenu à partir de l'expression suivante :

$$\begin{aligned} L(x_k) = & \min_{\substack{(s',s) \\ x_k=+1}} \alpha_{k-1}(s') + \gamma_k(s', s) + \beta_k(s) \\ - & \min_{\substack{(s',s) \\ x_k=-1}} \alpha_{k-1}(s') + \gamma_k(s', s) + \beta_k(s). \end{aligned} \tag{3.4}$$

Le cœur de l'algorithme Max-Log-MAP est essentiellement basé sur un composant ACS (Add-Compare-Select). La figure 3.1 décrit l'architecture d'un décodeur SISO utilisant l'algorithme Max-Log-MAP. A l'instant k, le module $Aller(\alpha)$ calcule récursivement la métrique de nœud α_k à partir de la métrique de branche γ_k et la métrique de nœud précédente α_{k-1}. Ces métriques α sont ensuite stockées dans une mémoire dédiée, notée αLIFO (Last In First

Out). Sa taille est alors égale à L mots de taille $P = Q_{sm} \times N_e$ bits, où Q_{sm} représente le nombre de bits nécessaires pour la quantification des métriques de nœud, tandis que N_e est le nombre d'états du treillis. De même, le calcul récursif des métriques de nœud β du sens *Retour* est réalisé dans le module *Retour*(β). Les valeurs correspondantes de α sont ensuite lues dans la mémoire αLIFO. Les opérations de l'expression 3.4 sont alors réalisées dans le module LRV. Il fournit donc le résultat $L(x_k)$ à partir des valeurs α_{k-1}, γ_k et β_k. Il est également possible d'effectuer le traitement du sens *Retour* précédant le processus du sens *Aller* au niveau architectural. Dans ce cas, c'est les métriques de nœud β qui sont sauvegardées. Le choix de l'ordonnancement des traitements se fait sur des considérations pratiques.

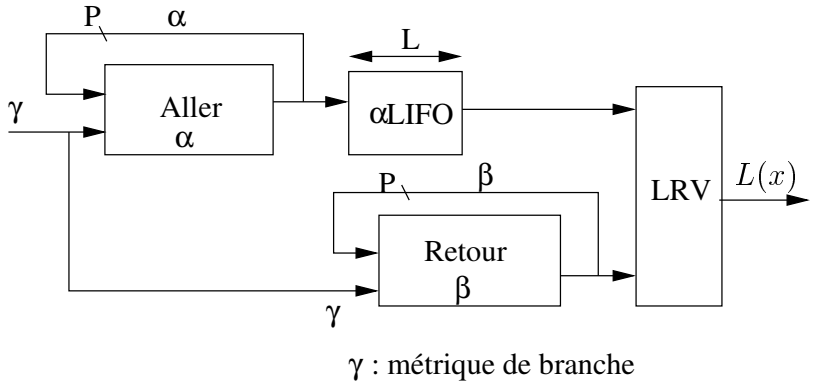


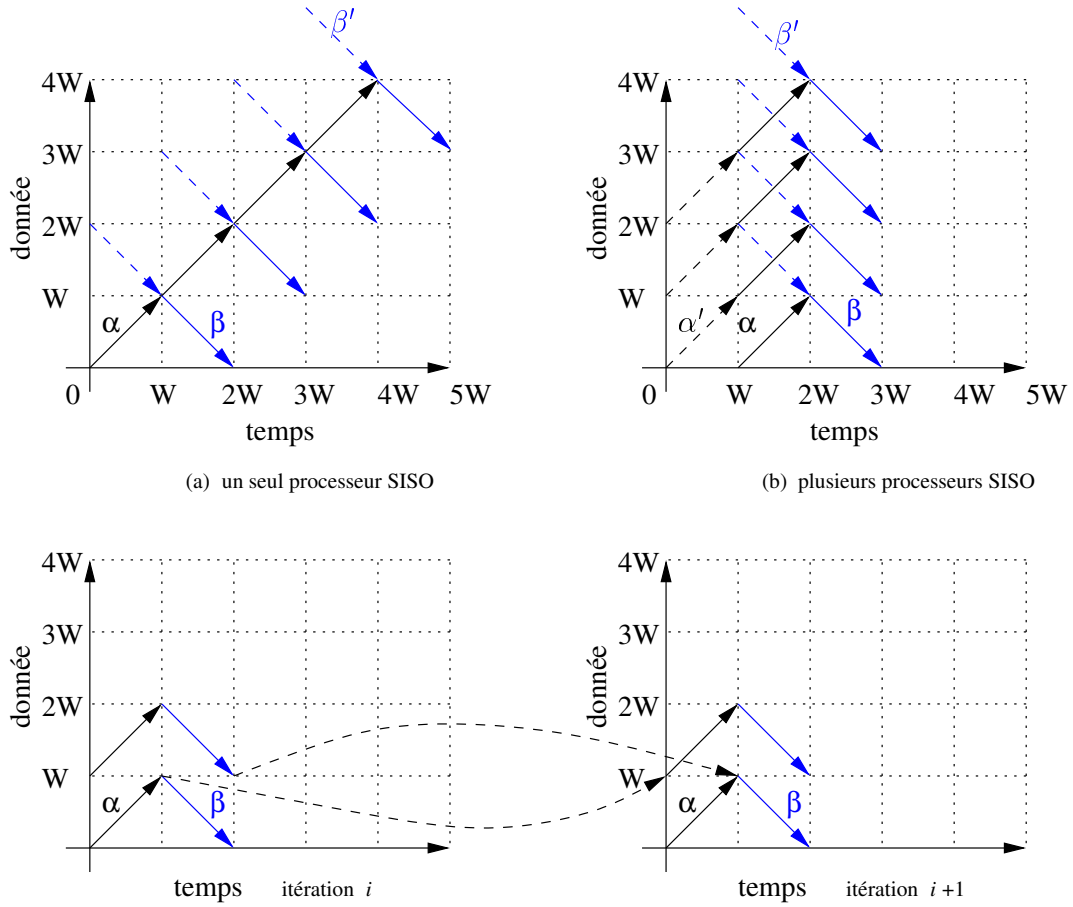
Figure 3.1 — Architecture d'un décodeur SISO associé à l'algorithme Max-Log-MAP.

Classiquement, le décodage du sens *Retour* ne commence que lorsque celui du sens *Aller* est terminé. Cela impose donc un délai important lorsque la longueur L est élevée, par exemple, $L = 1024$ bits. Dans ce dernier cas, la taille de mémoire αLIFO est alors égale à 7k octets lorsque les métriques de nœud α sont quantifiées sur 7 bits pour un décodeur à 8 états. Or, la consommation de mémoire peut atteindre 50% de la consommation globale d'un turbo-décodeur [Schurgers 01]. C'est pourquoi il est primordial de réduire la taille de la mémoire afin de diminuer la consommation statique du système.

La technique *sliding windows* [Raouafi 99] permet de réduire à la fois la taille de mémoire et la latence du décodage. Dans ce cas, la séquence de L mots est découpée en $N_w = L/W$ sous-séquences de taille W mots. Puis chaque sous-séquence est traitée de manière indépendante par un ou plusieurs processus SISO. Puisque la récursivité de calcul pour α (β respectivement) est découpée sous forme de fenêtres de longueur W, une initialisation pertinente aux deux extrémités d'une fenêtre est alors requise pour obtenir de bonnes performances du décodage [Boutillon 07]. Grâce à cette technique, la taille de la mémoire αLIFO est ainsi égale à W mots de taille P bits pour une fenêtre. Nous obtenons alors un facteur de diminution F_T exprimé de la manière suivante :

$$F_T = 1 - \frac{N_w \times W}{L} \tag{3.5}$$

Selon l'ordre d'exécution du décodage et le nombre de processus SISO, différentes solutions sont possibles pour initialiser les métriques de nœud (α et β) avant d'effectuer les calculs récursifs. Dans la figure 3.2, les données sont découpées en plusieurs fenêtres ayant la même longueur W. Lorsqu'un seul processus SISO est considéré, le décodage s'effectue alors de façon séquentielle sur l'ensemble de fenêtres dans l'ordre naturel. Dans ce cas, les métriques de nœud α sont d'abord stockées dans un tampon lors du décodage du sens *Aller*. Ensuite, elles sont utilisées avec les métriques de nœud β obtenues lors du décodage du sens *Retour* pour calculer le LRV.



(a) un seul processeur SISO

(b) plusieurs processeurs SISO

(c) initialisation d'après l'itération précédente

Figure 3.2 — Technique *sliding windows* appliquée à l'algorithme Max-Log-MAP.

Au moment du traitement de la première fenêtre, les valeurs des métriques de nœud α sont initialisées d'après la règle du codage. Dans notre cas, la métrique de nœud α à l'état 0 est initialisée à la valeur 0, une valeur constante élevée pour les autres états. Pour le reste du traitement, les valeurs initiales des métriques de nœud α succèdent à leurs valeurs finales respectives de la fenêtre précédente. Or, cela n'est pas le cas pour le décodage du sens *Retour*. Dans ce dernier cas, les métriques de nœud β doivent être proprement initialisées pour chaque fenêtre. Pour ce faire, nous pouvons effectuer un pré-décodage β' sur la fenêtre adjacente pour obtenir les valeurs initiales des métriques de nœud β (trait pointillé dans la figure 3.2(a)).

Lorsque nous procédons à plusieurs processus SISO en parallèle, il est alors nécessaire d'initialiser les métriques de nœud α et β. Comme dans la figure 3.2(a), les processus correspondants d'initialisation α' et β' sont respectivement effectués sur la fenêtre adjacente (figure 3.2(b). D'autre part, une solution élégante et efficace pouvant éliminer l'étape du pré-décodage est indiquée dans la figure 3.2(c). Dans ce cas, nous exploitons le fait que les processus SISO sont effectués de façon itérative. Pour l'itération i, les valeurs finales des métriques de nœud (α et β) d'une fenêtre sont alors propagées à la fenêtre adjacente (précédente ou suivante) dans la prochaine itération $i+1$. Cette approche conduit à avoir une perte de performances négligeable lors que la longueur de fenêtre est suffisamment élevée (voir [Boutillon 07]).

3.1.2 Normalisation des métriques de nœud

La quantification de données est un élément crucial lors de l'implémentation sur un circuit VLSI ou FPGA. En effet, la précision des données a un impact direct sur les performances du système mais aussi sur la complexité matérielle. Dans l'algorithme Max-Log-MAP, les métriques de nœud s'accumulent au fur et à mesure au cours du décodage. Il est donc indispensable de mettre en œuvre une technique permettant d'éviter le débordement. La technique est appelée la normalisation des métriques de nœud.

Le problème de la quantification a fait l'objet de nombreux travaux de recherche dans le cas de l'algorithme de Viterbi. Les méthodes proposées sont essentiellement basées sur le fait que la plage dynamique des métriques de nœud, *i.e.*, la différence entre la métrique maximale et la métrique minimale à l'instant donné, peut être bornée par une valeur constante Δ_{max}. Cette constante est fonction du code utilisé et de la valeur maximale de la métrique de branche Γ. La métrique de branche, quant à elle, est souvent normalisée dans un intervalle $[0, \Gamma]$.

Soit m la longueur de mémoire du codeur, la borne s'exprime alors comme suit [Cain 84] :

$$\max(\alpha_k) - \min(\alpha_k) \leq m \cdot \Gamma = \Delta_{max} \text{ pour } k \in \{0, \cdots, k-1\}. \tag{3.6}$$

A partir de cette borne, il est possible de proposer des méthodes minimisant le nombre de bits nécessaires Q_{sm} pour la quantification des métriques de nœud n'ayant pas d'impact sur les performances. Dans la littérature, les méthodes proposées peuvent être classées en 3 catégories : normalisation dynamique, normalisation constante et normalisation par modulo. Ces techniques sont détaillées dans les sous-sections suivantes.

3.1.2.1 Normalisation dynamique

L'expression 3.4, nous montre que nous ne nous intéressons qu'à la différence entre les métriques correspondantes. Lorsqu'une constante est soustraite à toutes les métriques de nœud, le résultat final n'est pas modifié par la soustraction. La normalisation dynamique consiste donc à soustraire le minimum parmi les métriques de nœud pour chaque étage du treillis. Dans ce cas, toutes les métriques de nœud sont supérieures ou égales à zéro. Le nombre de bits nécessaires pour quantifier les métriques de nœud est alors égal à :

$$Q_{sm} = \lceil \log_2(m\Gamma) \rceil, \tag{3.7}$$

où $\lceil x \rceil$ est le plus petit entier égal ou supérieur à x. Il est alors nécessaire de connaître le minimum des métriques à chaque étage du treillis pour appliquer cette technique. Cette contrainte a un impact non négligeable au niveau de la mise en œuvre matérielle de cette méthode de normalisation.

3.1.2.2 Normalisation par soustraction périodique d'une constante

Au lieu de soustraire la métrique minimale des métriques de nœud à chaque étage du treillis, une autre technique consiste à soustraire périodiquement une constante au cours du parcours du treillis. A partir de l'expression 3.6, nous pouvons établir la relation suivante :

$$\max(\alpha_{k+1}) - \min(\alpha_k) \leq (m+1) \cdot \Gamma. \tag{3.8}$$

La relation 3.8 montre que la valeur du terme $\max(\alpha_{k+1})$ ne dépasse pas $2(m+1)\Gamma$ si $\min(\alpha_k) < (m+1)\Gamma$. En revanche, si $\min(\alpha_k) \geq (m+1)\Gamma$, toutes les valeurs des métriques de nœud sont prises dans l'intervalle $[(m+1)\Gamma, 2(m+1)\Gamma]$. Dans ce cas, il est possible de soustraire la constante $(m+1)\Gamma$ pour toutes les métriques de nœud. Le résultat final n'est pas modifié puisque la différence entre les métriques est équivalente. Cette soustraction permet de maintenir les valeurs quantifiées dans une plage définie par l'expression suivante :

$$2^{Q_{sm-1}} \geq (m+1)\Gamma, \tag{3.9}$$

Ainsi, le nombre de bits nécessaires pour la quantification peut s'exprimer de la manière suivante [Boutillon 07] :

$$Q_{sm} = \lceil \log_2((m+1)\Gamma) \rceil + 1. \tag{3.10}$$

Au niveau de la mise en œuvre, il suffit alors d'observer les bits MSBs ayant le poids maximal. Lorsque tous les bits observés sont égaux à 1 après le calcul d'un étage du treillis, ils sont alors remis à 0 à l'aide d'une porte logique *et*. Ceci revient à soustraire une constante $2^{Q_{sm}-1}$ à toutes les métriques de nœud de cet étage. Dans le cas contraire, les valeurs ne sont pas modifiées.

3.1.2.3 Normalisation par modulo

La normalisation par modulo a initialement été proposée pour l'algorithme de Viterbi dans [Hekstra 89]. Au lieu de limiter le débordement de données, il est possible d'adopter le débordement à chaque étage de calcul.

A partir de l'expression 3.8, la différence des métriques de nœud à l'étage $k+1$ ne dépasse pas à $(m+1)\Gamma$. Si nous utilisons une représentation de donnée en complément à 2, il est alors possible de représenter toutes les donnée dans un cercle dont le périmètre représente la plage de quantification $[-(2^{Q_{sm}-1}), +(2^{Q_{sm}-1}-1)]$ sur des valeurs signées.

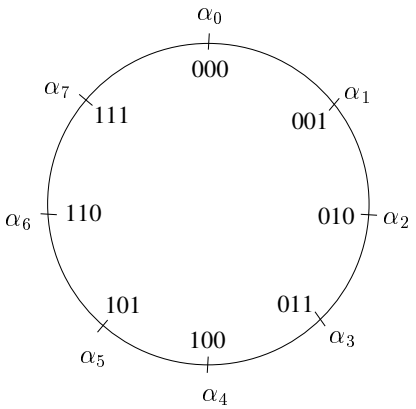


Figure 3.3 — Exemple d'une normalisation par modulo pour $Q_{sm} = 3$.

La figure 3.3 montre un exemple sur la normalisation par modulo pour $Q_{sm} = 3$. Toutes les valeurs possibles de 3 bits sont représentées dans un cercle dont la longueur de périmètre est égale à 8. Si la longueur d'un arc reliant deux points est inférieure à $8/2$, soit 4, il est alors possible de déterminer quel point représente la valeur supérieure. En effet, il suffit d'effectuer une soustraction entre les deux valeurs signées (représentation à complément à 2). Prenons

un exemple suivant.

$$\alpha_5 - \alpha_2 = 101 - 010 = 101 + 110 = 011 \rightarrow 3 \Rightarrow \alpha_5 > \alpha_2. \tag{3.11}$$

Dans ce cas, l'opération ACS est considérée comme l'opération arithmétique de type complément à 2 à chaque étage. Ainsi, le nombre de bits nécessaires correspond à la relation suivante :

$$2^{Q_{sm}-1} - 1 \geq (m+1)\Gamma. \tag{3.12}$$

Nous obtenons alors :

$$Q_{sm} = \lceil \log_2((m+1)\Gamma + 1) \rceil + 1. \tag{3.13}$$

3.1.2.4 Bilan sur les techniques de normalisation

La normalisation dynamique permet d'obtenir un meilleur nombre de bits pour la quantification des métriques de nœud. Mais elle nécessite de maintenir la valeur minimale des métriques de nœud égale à 0 à chaque étage du treillis. Au niveau de la mise en œuvre, il est donc nécessaire d'ajouter un module supplémentaire pour effectuer le tri des métriques de nœud, suivi d'une soustraction entre les métriques de nœud et leur valeur minimale. Cet ajout représente d'une part une augmentation du coût matériel, d'autre part une augmentation du chemin critique du système. Il est donc important de prendre soin des contraintes temporelles lors de l'implémentation de la normalisation dynamique.

La normalisation par soustraction de constante, quant à elle, n'est pas compliquée à réaliser. Pour ce faire, il suffit alors d'ajouter quelques portes logiques. D'un point de vue matériel, sa réalisation n'est donc pas coûteuse. En examinant les expressions 3.7 et 3.10, la normalisation par soustraction peut générer deux bits de pénalité par rapport à la normalisation dynamique. De même, il est aisé d'implémenter la normalisation par modulo. Dans ce cas, il suffit d'implémenter l'opérateur ACS par des additionneurs et des comparateurs ayant pour une représentation en complément à 2. En comparant les expressions 3.10 et 3.13, nous constatons que la normalisation par modulo peut avoir un bit de pénalité dans le pire cas par rapport à la normalisation par soustraction.

3.2 Saturation des métriques de nœud

Dans cette section, nous proposons une méthode basée sur la saturation des métriques de nœud pour tenter de diminuer la taille de la mémoire αLIFO. Au niveau de la réalisation, deux approches sont considérées : Saturation A l'Intérieur (SAI) et Saturation A l'Extérieur (SAE) du calcul récursif des métriques de nœud. Leurs architectures respectives sont également présentées. Enfin, en appliquant notre approche, nous présentons les performances d'un turbo-décodeur en terme de TEB.

3.2.1 Saturation A l'Extérieur (SAE) du calcul récursif

Pour diminuer la taille de la mémoire αLIFO égale à $W \times P$ avec $P = Q_{sm} \times N_e$, nous pouvons limiter le nombre de bits P pour quantifier les métriques de nœud α. Dans l'équation 3.4, la sortie souple du décodeur $L(x)$ est égale à la différence de deux minimums. Considérons l'ensemble $\{\alpha(s') + \gamma(s', s) + \beta(s)|_{x=1}\}$, si la valeur $\alpha(s')$ est suffisamment élevée, il est peu probable que l'élément correspondant à cet ensemble $\{\alpha(s') + \gamma(s', s) + \beta(s)|_{x=1}\}$ ait une

valeur minimale. Le même raisonnement peut être appliqué à l'ensemble concurrent pour $x = 0$. Ainsi, il est possible de saturer les valeurs élevées des métriques de nœud α sans dégrader les performances.

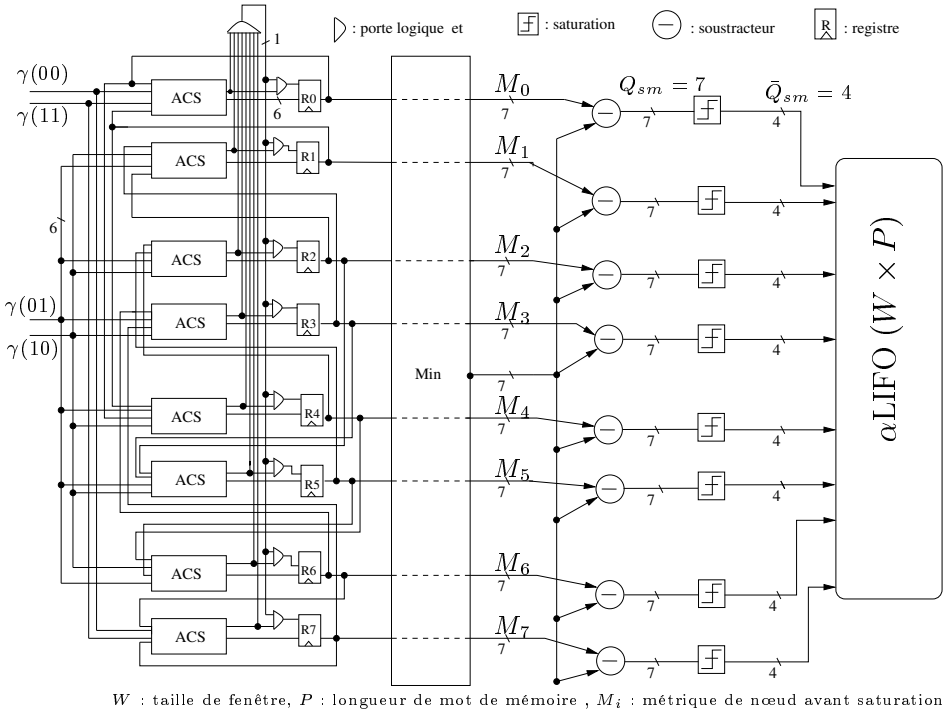


Figure 3.4 — Mise en œuvre de la méthode SAE pour un décodeur SISO à 8 états.

Lorsque la saturation se produit à l'extérieur du calcul récursif des métriques de nœud (voir la figure 3.4), c'est-à-dire, la saturation précède la mémorisation des métriques α, elle n'a pas d'impact sur les calculs des métriques suivantes. Grâce à la saturation, le nombre de bits pour la quantification des métriques de nœud peut être fixé à $\bar{Q}_{sm}$ bits. La taille de la mémoire αLIFO est alors égale à $W \times (\bar{Q}_{sm} \times N_e)$. Il est évident que la complexité du calcul du LRV diminue par rapport à celle pour la méthode classique, car le nombre de bits de quantification après saturation est réduit. La normalisation des métriques de nœud est réalisée à l'aide d'une des méthodes présentées dans la section précédente.

La figure 3.4 détaille l'architecture dédiée à la SAE (Saturation A l'Extérieur) du calcul récursif pour un décodeur SISO à 8 états. Le rendement de code est égal à 1/2. Le symbole $\gamma(xy)$ décrit la métrique de branche associée au mot de code $\mathbf{v}(x, y)$ d'un treillis. Pour un codeur binaire, il y a donc 4 métriques de branche différentes : $\gamma(00), \gamma(01), \gamma(10)$ et $\gamma(11)$. Au niveau de la quantification, la partie systématique X et la partie redondante Y du mot reçu $\mathbf{V}(X, Y)$ sont quantifiées sur 4 bits. La quantification des métriques de branche est détaillée dans la section 3.2.3.2. Les métriques de branche sont quantifiées sur 6 bits dans notre cas. Il y a 8 ACS travaillant en parallèle, ce qui correspond à 8 états.

Après l'étage des blocs ACS sur le calcul récursif des métriques de nœud, les résultats sont temporisés dans une file de registres (notés $R_0, \cdots, R_7$) sur $7 \times N_e$ bits. En parallèle, les valeurs (notées $M_0, \cdots, M_7$) quantifiées sur $Q_{sm} = 7$ bits alimentent le module « Min ». La normalisation par soustraction constante est appliquée à la file des registres à l'aide d'une porte logique *et* de 8-entrées, qui observe en parallèle les MSBs des entrées des registres. Lorsque tous les MSBs sont égaux à 1, ils sont remis à 0 grâce à une autre porte logique *et* de 2-entrées (technique de soustraction, voir la section 3.1.2.2).

Le module « Min » recherche la valeur minimale des métriques de nœud α. Cette valeur est ensuite soustraite à toutes les métriques de nœud. Ainsi, la soustraction d'un minimum permet de maintenir une valeur minimale égale à 0. Il est alors possible de quantifier les résultats de soustraction sur les valeurs non-signées à 7 bits. Enfin, une saturation de 7 bits à 4 bits précède la mémorisation des métriques de nœud. La taille de la mémoire αLIFO a donc pour valeur $4 \times 8 \times W$ bits. Nous constatons que la saturation est réalisée à l'extérieur du calcul ACS. Dans ce cas, il est nécessaire d'appliquer la normalisation par soustraction de constante pour maintenir les métriques de nœud dans une plage cohérente.

En résumé, la saturation SAE n'a pas d'impact sur la fréquence de fonctionnement. Elle permet par contre une diminution à la fois de la taille de la mémoire et de la complexité du module LRV et ce, au prix d'une augmentation du coût matériel liée aux opérations du tri et de la soustraction. Par ailleurs, le calcul récursif des métriques n'est pas concerné par la méthode proposée. C'est pourquoi la normalisation par soustraction de constante est appliquée à l'issu du composant ACS, comme le montre la figure 3.4.

3.2.2 Saturation A l'Intérieur (SAI) du calcul récursif

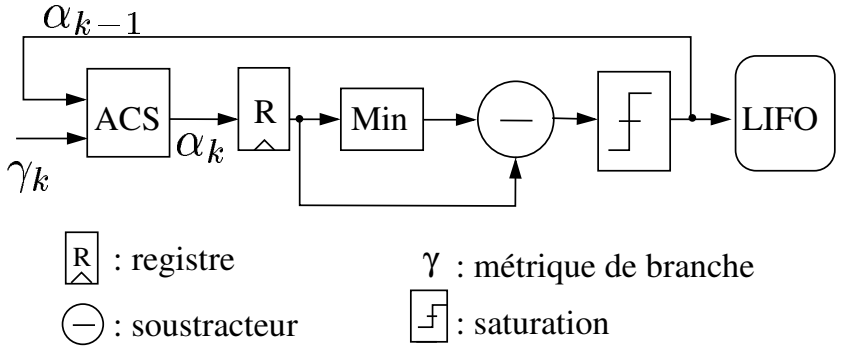


Figure 3.5 — Saturation à l'intérieur du calcul récursif sur le composant ACS.

La saturation de métrique peut être appliquée à l'intérieur du calcul récursif sur le composant ACS. Dans ce cas, les métriques de nœud saturées α_{k-1} sont utilisées pour le prochain calcul des métriques α_k (figure 3.5). Pour ce faire, toutes les métriques sont soustraites par une valeur minimale dans chaque étage, ce qui est équivalent à une normalisation dynamique. Ainsi, l'ensemble des métriques de nœud contient au moins une valeur minimale 0 et les autres sont égales ou supérieures à 0. Comme pour la méthode SAE, le nombre de bits après saturation est égal à $\bar{Q}_{sm} = 4$ bits. Mais contrairement à la méthode SAE, les valeurs saturées interviennent directement dans le calcul récursif. Les métriques de nœud sont donc modifiées au cours du parcours du treillis.

3.2.3 Etude sur les complexités matérielles et les performances des méthodes SAE et SAI

Dans cette section, nous comparons d'abord les complexités matérielles des méthodes SAE et SAI. Ensuite, les performances en terme de taux d'erreurs binaires de ces méthodes sont présentées.

3.2.3.1 Les complexités matérielles des méthodes SAE et SAI

La saturation des métriques de nœud est obtenue à l'aide des opérations suivantes. La première opération consiste à trouver la valeur minimale parmi les métriques de nœud. Ensuite, une soustraction entre la valeur minimale et les métriques de nœud est effectuée avant de procéder à la saturation. Grâce à cette soustraction, la métrique de nœud minimale est maintenue à la valeur 0.

Pour mettre en œuvre notre approche, Leurs architectures respectives sont détaillées dans le chapitre 4. Le module « Min » pouvant trouver la valeur minimale en fonction des entrées est réalisé à l'aide de 7 comparateurs et 7 multiplexeurs, ce qui correspond à un décodeur de 8 états. De plus, nous utilisons 8 soustracteurs pour effectuer les soustractions totales. Quant à la réalisation d'une saturation, elle nécessite 1 *ou* de 3-entrées et 4 *ou* de 2-entrées. Il y a donc 8 *ou*s de 3-entrées et 32 *ou*s de 2-entrées pour 8 saturations. Par rapport au décodeur classique SISO, l'application des méthodes SAE et SAI augmentent légèrement la complexité matérielle du décodeur. Or, la réduction de la quantification Q_{sm} réduit aussi la complexité matérielle du module LRV, car le nombre de bits à son entrée passe de 7×16 bits à 4×16 bits.

La normalisation dynamique est appliquée pour la méthode SAI par une soustraction du minimum. Les valeurs à l'instant $k+1$ sont obtenues à partir des valeurs saturées à l'instant k. Ainsi, le module supplémentaire « Min » et les soustractions introduisent un délai sur le chemin critique, d'où une augmentation du chemin critique qui affecte le débit du système. Cependant, il est possible d'utiliser la technique pipeline en entrelaçant plusieurs processus SISO pour diminuer ce délai (voir [Boutillon 07]). L'impact sur le délai du décodage est ainsi minimisé. En revanche, si la méthode SAE est considérée, la normalisation constante est appliquée. Le chemin critique n'est donc pas modifié dans ce cas.

3.2.3.2 Les performances des méthodes SAE et SAI

Nous obtenons les mêmes performances en terme de TEB pour les approches SAE et SAI. C'est la raison pour laquelle nous ne présentons que les performances obtenues pour la méthode SAE dans cette section.

La figure 3.6 donne les performances en terme de TEB d'un turbo-décodeur après application de la méthode SAE. Le décodeur adopté par le système UMTS a pour polynômes générateurs $G(13,15)_\circ$. Son rendement de codage est égal à 1/3. La longueur de trame est égale à 864 bits. Le mapping MDP2 est considérée. Le nombre d'itérations est fixé à 6. Pour éviter la corrélation entre les informations extrinsèques, le facteur de pondération ζ [Boutillon 07] est égal à 0.5 pour la première itération, 1 pour la dernière itération et 0.75 pour les autres itérations.

La courbe, dite *flottante*, est définie de la manière suivante. Les symboles $\mathbf{V}(X, Y_1, Y_2)$, les informations extrinsèques z et les métriques de nœud (α et β) sont représentés par leurs valeurs réelles au cours du décodage. Pour ce qui est des « versions quantifiées », le pas de quantification q est égal à 0.38. Les entrées X, Y_1 et Y_2 sont saturées puis quantifiées sur $Q_V = 4$ bits. De même, les informations extrinsèques z sont quantifiées sur $Q_z = 6$ bits. Une constante positive de 7 bits est ajoutée à la métrique de branche pour que cette dernière soit positive [Boutillon 07]. Dans ce cas, les valeurs γ sont non-signées et quantifiées sur $\mathbb{Q}_{bm} = 6$ bits. Selon [Montorsi 01], le nombre de bits Q_{sm} pour la quantification des métriques de nœud devrait être de $Q_V + 3$ bits, c'est-à-dire, 7 bits dans notre cas. En réalité, nous utilisons

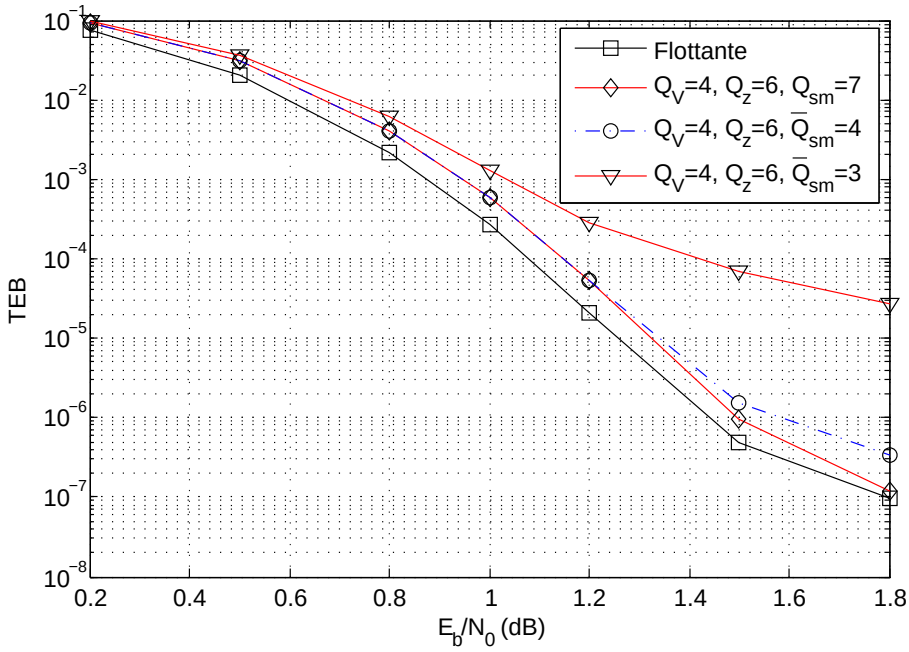


Figure 3.6 — Performances d'un turbo-décodeur adopté par le système UMTS avec l'application de la méthode SAE.

$\bar{Q}_{sm} = 3$ bits ou 4 bits sans changement du pas de quantification, mais avec une saturation au $8^{\text{ème}}$ (ou respectivement $16^{\text{ème}}$) niveau de discrétisation.

En comparant à la courbe de référence (*flottante*) de la figure 3.6, nous obtenons une perte de 0.05dB pour un TEB de 10^{-6} lorsque $Q_V = 4$ bits, $Q_z = 6$ bits et $Q_{sm} = 7$ bits. Une perte supplémentaire de 0.1dB est obtenue lorsque les valeurs des métriques de nœud sont saturées (passage de $Q_{sm} = 7$ bits à $\bar{Q}_{sm} = 4$ bits). En revanche, la saturation sur 3 bits génère une perte plus de 3.6dB sous la même condition.

En conclusion, les simulations ne montrent pas de différence entre les deux méthodes en terme de performances. Ceci peut s'interpréter par le fait que, si la valeur $\alpha(s')$ est élevée, elle ne contribue pas à la décision finale car le terme $\{\alpha(s')+\gamma(s',s)\}$ a aussi une valeur élevée. Par conséquent, il est improbable que ce terme soit sélectionné pour la prochaine itération (voir l'expression 3.2). Grâce à l'approche proposée, les métriques de nœud peuvent être saturées et puis quantifiées sur 4 bits tout en maîtrisant les performances du décodage. Ainsi, nous pouvons diminuer de 43% la taille de la mémoire, qui stocke les métriques de nœud α.

3.3 Estimation de la consommation à l'aide de l'outil CACTI

Dans cette section, la consommation de la mémoire dans un turbo-décodeur est estimée lorsque la saturation des métriques de nœud est appliquée. Tout d'abord, nous présentons l'outil d'estimation de consommation CACTI [Tarjan 06]. L'architecture d'une mémoire cache est brièvement revue. Ensuite, les résultats d'estimation au niveau de la surface des mémoires et de la consommation associée sont donnés en fonction du nombre de fenêtres (application du principe *sliding windows*). Dans nos expérimentations, la mémoire utilisée est considérée de type SRAM, en particulier de type mémoire cache LIFO (Last In First Out) utilisant la technologie 100 nm.

3.3.1 Présentation de l'outil CACTI

Les travaux de recherche sur l'architecture d'un système électronique consistent souvent à trouver un meilleur compromis parmi plusieurs solutions. Ceci ne peut pas se faire sans avoir une connaissance suffisante sur les coûts éventuels de chaque solution. Il est par exemple impossible de comparer les architectures de deux types de cache sans tenir compte de la différence sur les temps d'accès mémoire. De même, la surface du circuit et la consommation associée doivent aussi être prise en compte pour évaluer une architecture. En général, nous ne pouvons prendre une décision architecturale que lorsque suffisamment d'informations sont disponibles pour discriminer les différentes solutions.

Or, il est difficile de déterminer les coûts exacts sans passer par l'étape de l'implémentation. Pour répondre à cette demande, une solution consiste à employer des modèles mathématiques permettant de calculer approximativement ces coûts à partir des caractéristiques du système. C'est dans ce contexte que les laboratoires de recherche de l'entreprise HP ont développé un outil, appelé CACTI [Tarjan 06]. Il modélise le temps d'accès mémoire et détermine une meilleure configuration en terme d'architecture. En parallèle, il estime la consommation de la mémoire à partir de la technologie utilisée et de sa taille. La nouvelle version de l'outil prend en compte non seulement la consommation dynamique, mais aussi la consommation statique. Comme nous l'avons mentionné dans le chapitre 1, la consommation statique est proportionnelle à la surface du circuit et elle devient de plus en plus un critère dominant au niveau de la consommation [Sakurai 03]. L'expression 3.14 (voir Chapitre 1) décrivant la consommation dynamique P_{dyn} et la consommation statique P_{lkg}, est considérée comme la fonction de base selon le document [Tarjan 06] :

$$P_{dyn} = V_{DD}^2 \cdot f \cdot \alpha \cdot C_L, \quad P_{lkg} = V_{DD} \cdot I_{lkg}, \tag{3.14}$$

où les symboles V_{DD}, f, α, C_L, I_{lkg} représentent respectivement la tension d'alimentation, la fréquence de fonctionnement, l'activité du système, la capacité parasite et le courant statique.

3.3.2 Architecture d'un système microélectronique utilisant la mémoire cache

Dans un système microélectronique, la mémoire cache établit une passerelle entre le microprocesseur et la mémoire principale. Les informations les plus utilisées d'une mémoire principale sont souvent stockées dans une mémoire cache. En général, la mémoire cache est plus petite par rapport à la mémoire principale. Ainsi, son accès rapide aux données permet d'accélérer les traitements. Les données peuvent être un programme, un bloc d'image à traiter etc.

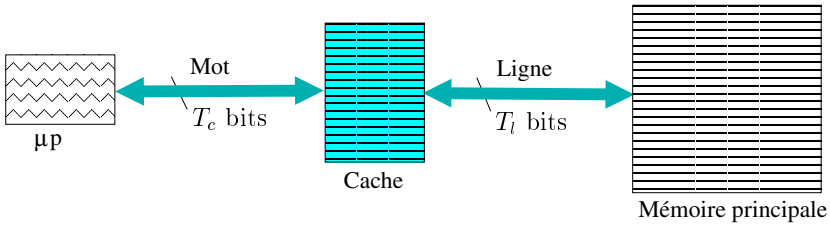


Figure 3.7 — Architecture d'un système microélectronique utilisant la mémoire cache.

La figure 3.7 représente de manière intuitive l'architecture d'un système microélectronique utilisant la mémoire cache. Le système fonctionne de la manière suivante. Le microprocesseur

(μp) envoie d'abord une demande de données à la mémoire cache. Cette dernière vérifie si les données requises sont présentes. Lorsque la mémoire cache est en possession de ces données, dit *succès de cache*, elle transmet à chaque envoi un *mot de cache* de taille T_c bits au microprocesseur à travers un bus. Lorsque les données demandées ne sont pas présentes dans la mémoire cache, dit *défaut de cache*, la mémoire cache demande alors à la mémoire principale d'envoyer les données correspondantes. Dans ce cas, la mémoire principale transmet une *ligne* de données de taille T_l bits à la mémoire cache à travers un bus. Ces données sont ensuite écrites dans la mémoire cache. Le procédé est appelé *la mise à jour* de la mémoire cache. Comme le montre la figure 3.7, la mémoire cache est notamment utilisée entre le microprocesseur et la mémoire vive RAM (Random Access Memory en anglais). Mais il est possible de l'utiliser entre tout fournisseur de données (réseau informatique, disque dur, mémoire principale) et le consommateur de données. La gestion d'écriture et de lecture d'une mémoire cache ne fait pas l'objet de la thèse. Pour plus d'information, les articles [Belady 66], [H. Al-Zoubi 04] détaillent les autres politiques sur la gestion des accès mémoires (écriture et lecture).

Comme mentionné dans le paragraphe précédent, par définition, la mémoire cache est relativement petite. Son principal intérêt est de pouvoir accélérer les traitements grâce à l'accès rapide aux données demandées. Ainsi, la mémoire cache ne peut contenir toutes les données de la mémoire principale. Il est alors nécessaire d'indiquer une adresse à la mémoire cache au moment où une ligne de données est arrivée de la mémoire principale. La méthode d'adressage est dite le *mapping*. Il existe trois types de mapping répandus dans les caches aujourd'hui :

1. les mémoires caches complètement associatives (fully associative cache en anglais) ;
2. les mémoires caches N-associatives (N-way set associative cache en anglais) ;
3. les mémoires caches directes (direct mapped cache en anglais).

Nous considérons le mapping direct pour notre expérimentation suivante.

3.3.3 Architecture d'une mémoire cache

Nous décrivons d'abord l'architecture d'une mémoire cache [Wilton 94]. La figure 3.8 montre les différents blocs de l'architecture dédiée à une mémoire cache. Cette architecture est essentiellement composée d'une matrice d'adresses, dit tag (en anglais), une matrice de données et un décodeur d'adresse ainsi que la partie contrôle. Dans un premier temps, l'adresse entrante est traitée dans le bloc *décodeur d'adresse*, qui aliment en parallèle les lignes de mot dans les matrices d'adresses et de données. Chaque matrice est une combinaison des lignes de mot (wordline en anglais) et des lignes de bit (bitline en anglais). Elle contient autant de lignes que le permet sa taille.

Une cellule mémoire attaché à la ligne de mot est associée à deux lignes de bit, qui sont initialement préchargées à 1. Lorsqu'une ligne de mot est activée, une des deux lignes de bit est déchargée à 0. La cellule associée est alors sélectionnée. Il est à noter que c'est la valeur stockée dans la cellule qui détermine quelle ligne est déchargée à 0. L'amplificateur, quant à lui, observe l'événement des lignes de bit. Lorsqu'une ligne de bit déchargée est détectée, l'amplificateur détermine la valeur stockée dans la cellule. Il est possible de partager un amplificateur entre plusieurs couples de ligne de bit à l'aide des multiplexeurs. Les informations lues dans la matrice d'adresse sont ensuite comparées à l'adresse entrante. Grâce aux résultats des comparaisons, les données correspondantes sont sélectionnées à la sortie et le signal de validation est activé.

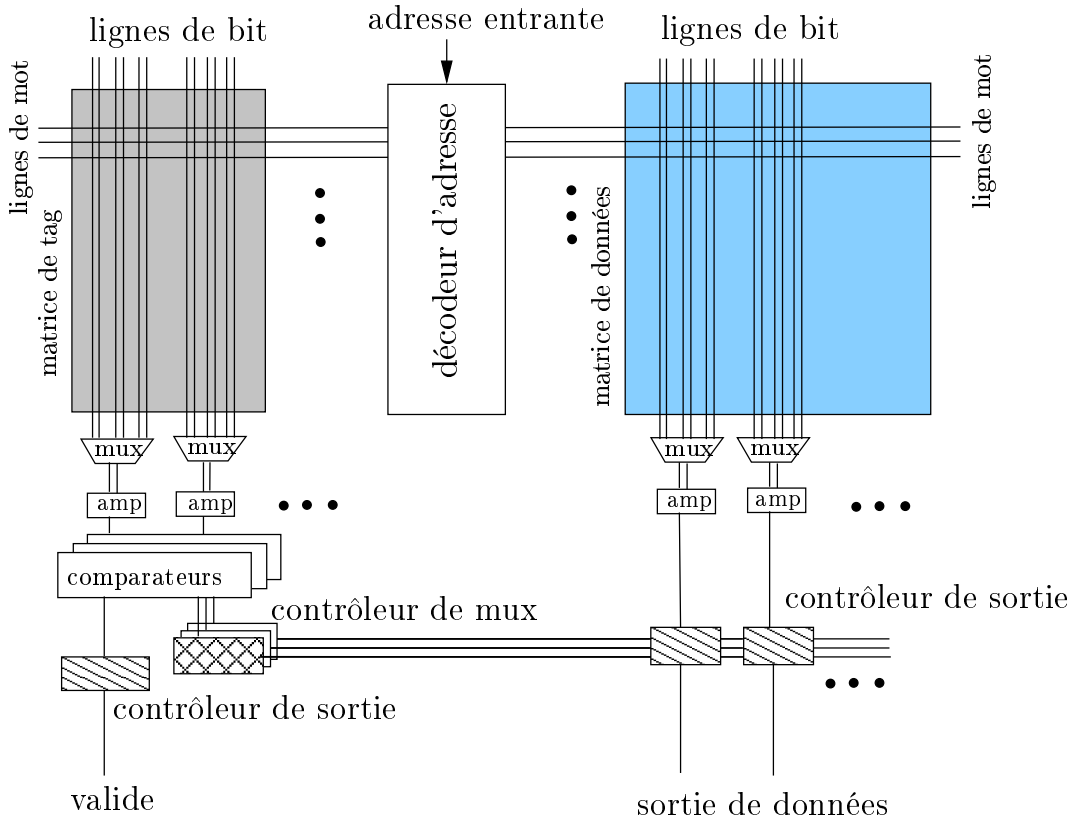


Figure 3.8 — Architecture d'une mémoire cache.

3.3.4 Les mémoires nécessaires dans un turbo-décodeur

L'objectif a consisté à évaluer l'impact énergique en appliquant la saturation des métriques de nœud à un turbo-décodeur. Nous considérons le turbo-décodeur spécifié dans la norme UMTS. Pour obtenir une bonne estimation de la consommation, il est nécessaire de l'implémenter au niveau RTL (Register Tranfer Level) dans une chaîne de communications numériques. Cependant, une estimation de la consommation due aux mémoires peut être réalisée avec l'outil CACTI. C'est pourquoi nous considérons uniquement la partie mémoire dan un turbo-décodeur dans la suite de cette section.

Tout d'abord, nous décrivons les mémoires principales dans un turbo-décodeur. Trois mémoires, notées $\mathbb{M}_X$, $\mathbb{M}_{Y_1}$ et $\mathbb{M}_{Y_2}$, sont respectivement consacrées à la mémorisation des données X, Y_1 et Y_2 (voir la figure 3.9). Leur taille totale est égale à $3 \times Q_V \times L$ bits. La mémoire $\mathbb{M}_z$ de taille $Q_z \times L$ bits est dédiée à la mémorisation de l'information extrinsèque z. Enfin, les métriques de nœud α sont stockées dans la mémoire αLIFO. Sa taille a pour valeur $N_w \times (W \times (Q_{sm} \times N_e))$ bits en fonction du nombre de fenêtres N_w traitées en parallèle. Nous avons alors :

$$L \leq W \times N_w. \tag{3.15}$$

Pour l'expérimentation, nous prenons la configuration suivante : $Q_V = 4$; $Q_z = 6$; $W = 64$; $L = 864$; $N_w = 1, 2, 4$; $Q_{sm} = 7, 4$; $N_e = 8$. La taille totale des mémoires de données, y compris les mémoires $\mathbb{M}_X$, $\mathbb{M}_{Y_1}$ et $\mathbb{M}_{Y_2}$, ont alors pour valeur 3×3456 bits. La mémoire $\mathbb{M}_z$ a pour taille 5184 bits. De plus, la taille de la mémoire αLIFO pour le traitement d'une fenêtre est égale à $512 \times Q_{sm}$ bits. La mémoire dispose des caractéristiques suivantes dans notre cas.

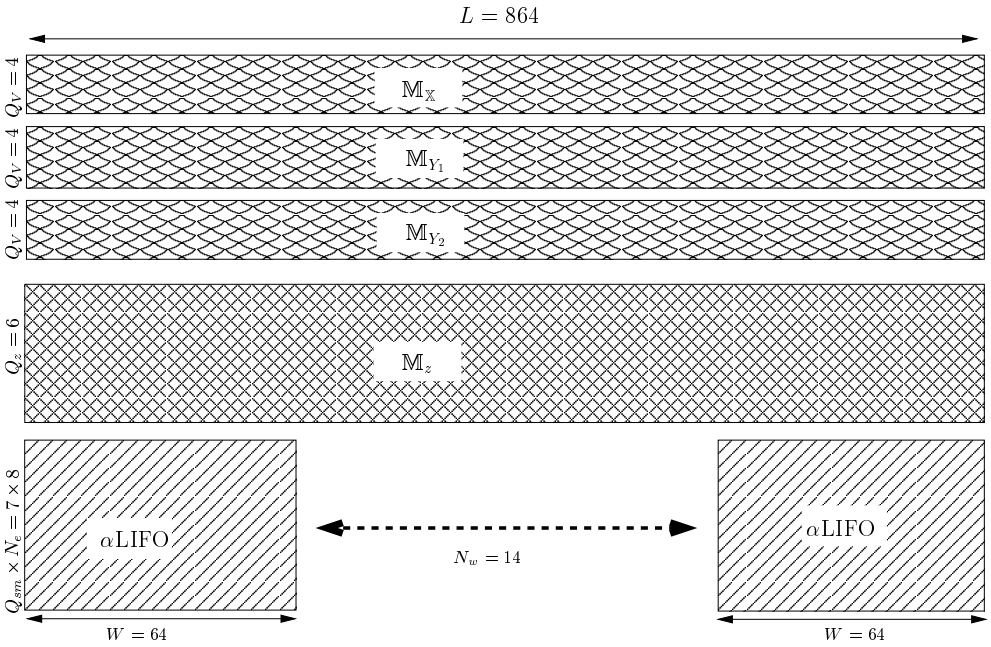


Figure 3.9 — Les mémoires principales dans un turbo-décodeur.

Elle est réalisée à partir de la technologie 100 nm avec une simple porte (lecture et écriture). Au niveau de l'adressage, le mapping direct est considéré. La tension d'alimentation est égale à 1.1 v et la mémoire fonctionne à 400MHz. Nous ne considérons que l'énergie consommée par la matrice de données dans la mémoire cache.

Pour les paramètres de l'outil, il est nécessaire de préciser les tailles d'une ligne entrante et d'un mot sortant. La taille de ligne T_l pour l'écriture de la mémoire αLIFO (voir la figure 3.7) est égale à $N_w \times Q$ bits, où le symbole Q est le nombre de bits pour la quantification des données considérées, *i.e.*, $Q = 6$ bits pour l'information extrinsèque. Si $N_w \times Q < 64$ bits, nous prenons $T_l = 64$ bits (contrainte imposée par l'outil). En parallèle, le mot sortant de la mémoire αLIFO a pour taille $T_c = N_w \times Q$ bits, soit $N_w \times 6$ bits pour l'information extrinsèque, $N_w \times 4$ bits pour les données X, Y_1 et Y_2 et $N_w \times 8 \times Q_{sm}$ bits pour les métriques de nœud.

Rappelons que l'objectif de nos approches est de diminuer la consommation énergétique du turbo-décodeur. La saturation des métriques de nœud est appliquée dans un turbo-décodeur adapté à la norme UMTS. La sous-section suivante détaille respectivement les surfaces des mémoires et l'énergie consommée par un accès mémoire pour $Q_{sm} = 7$ bits et $\bar{Q}_{sm} = 4$ bits en fonction du nombre de fenêtres N_w.

3.3.5 Estimation de l'énergie consommée par les mémoires

L'énergie consommée par les mémoires peut être évaluée à partir de la relation suivante :

$$E = E_{lkg} + (E_R + E_W),\ E_{lkg} = \frac{1}{f} \times P_{lkg} \tag{3.16}$$

où les symboles f, P_{lkg}, E_{lkg}, E_R et E_W représentent respectivement la fréquence de travail de la mémoire, la puissance statique, l'énergie statique et l'énergie dynamique pour une lecture et une écriture.

Tableau 3.1 — Estimation de la surface de mémoire pour $Q_{sm} = 7bits$ et $\bar{Q}_{sm} = 4$ bits.

N_w	A_V ($\times 10^{-3}$mm^2)	A_z ($\times 10^{-3}$mm^2)	$^{\triangle}A_{sm}$ ($\times 10^{-3}$mm^2)	$^{\triangle}\tilde{A}_{sm}$ ($\times 10^{-3}$mm^2)	F_A
1	17×3	17	26	13	14%
2	17×3	17	35	30	5%
4	17×3	25	81	41	25%

$^{\triangle}$ Les symboles A_{sm} et $\tilde{A}_{sm}$ correspondent respectivement à la surface de mémoire pour $Q_{sm} = 7$ bits et $\bar{Q}_{sm} = 4$ bits.

Le facteur de réduction de la surface F_A est obtenu à partir de l'expression suivante :

$$F_A = \left(1 - \frac{A_V + A_z + \tilde{A}_{sm}}{A_V + A_z + A_{sm}}\right). \tag{3.17}$$

où les symboles A_V, A_z, A_{sm} et $\tilde{A}_{sm}$ représentent respectivement les surfaces de la mémoire de données $\mathbb{M}_X$, $\mathbb{M}_{Y_1}$ et $\mathbb{M}_{Y_2}$, de la mémoire de l'information extrinsèque $\mathbb{M}_z$ et de la mémoire αLIFO pour $Q_{sm} = 7$ bits et $\bar{Q}_{sm} = 4$ bits.

Le tableau 3.1 présente les facteurs de réduction F_A en fonction du nombre de fenêtres N_w dans un turbo-décodeur. Nous obtenons une réduction de 12% au niveau de la surface lorsque le nombre de bits Q_{sm} pour la quantification des métriques de nœud passe de 7 bits à 4 bits pour $N_w = 1$. De plus, cette réduction atteint 19% lorsque le nombre de fenêtres en parallèle est égal à 4.

Le taux d'occupation T_{om} pour la mémoire des métriques de nœud αLIFO par rapport à la surface globale peut s'exprimer comme suit :

$$T_{om} = \frac{A_{sm}}{A_{sm} + A_z + A_V}. \tag{3.18}$$

Considérons la surface occupée par la mémoire de métriques de nœud. Pour 4 fenêtres en parallèle, nous obtenons respectivement un taux d'occupation de 52% et de 35% par rapport à la surface globale pour $Q_{sm} = 7$ bits et $\bar{Q}_{sm} = 4$ bits. En conclusion, la surface de la mémoire αLIFO est significative lorsque le nombre de fenêtres est important.

Comme pour les surfaces des mémoires, le facteur de réduction énergétique F_E est obtenue à partir de l'expression suivante :

$$F_E = \left(1 - \frac{E_V + E_z + \tilde{E}_{sm}}{E_V + E_z + E_{sm}}\right) \tag{3.19}$$

où les symboles E_V, E_z, E_{sm}, et $\tilde{E}_{sm}$ sont respectivement l'énergie consommée dans la mémoire de donnée, la mémoire de l'information extrinsèque et la mémoire des métriques de nœud avant et après saturation. Au niveau de la consommation énergétique, nous distinguons trois types d'énergie : énergie statique E_{lkg}, énergie dynamique consommée respectivement par une lecture E_R et une écriture E_W.

D'après le tableau 3.2, nous obtenons une réduction respective de 18% et de 31% pour des nombres de fenêtre égaux à 1 et 4. En parallèle, le tableau 3.2 démontre que la consommation statique est aussi une source principale de la consommation globale. Par exemple, nous obtenons une consommation statique de 20×10^{-4}nj ($\tilde{E}_R^{sm} + \tilde{E}_W^{sm}$) et une consommation dynamique de 80×10^{-4}nj lorsque les métriques de nœud sont saturées à $\bar{Q}_{sm} = 4$ bits pour 4 fenêtres. Pour conclure, l'application de notre approche diminue la consommation énergétique

Tableau 3.2 — Estimation de la consommation énergétique pour un décodeur avec saturation

N_w	$E_V(\times 10^{-4}\text{nj})$			$E_z(\times 10^{-4}\text{nj})$			$^{\triangle}E_{sm}(\times 10^{-4}\text{nj})$			$^{\triangle}\tilde{E}_{sm}(\times 10^{-4}\text{nj})$			F_E
	E_R^V	E_R^W	E_W^{lkg}	E_R^z	E_W^z	E_{lkg}^z	E_R^{sm}	E_W^{sm}	E_{lkg}^{sm}	$\tilde{E}_R^{sm}$	$\tilde{E}_W^{sm}$	$\tilde{E}_{lkg}^{sm}$	
1	21	6	24	9	3	12	29	5	10	13	3	6	18%
2	27	9	24	11	4	12	56	14	18	33	6	11	22%
4	36	9	24	18	4	13	124	29	37	64	16	20	31%

$^{\triangle}$ Les symboles E_{sm} et $\tilde{E}_{sm}$ correspondent respectivement à l'énergie consommée pour $Q_{sm} = 7$ bits et $\bar{Q}_{sm} = 4$ bits.

d'un turbo-décodeur en fonction du nombre de fenêtres.

3.4 Conclusion

Dans l'algorithme de décodage Max-Log-MAP, les métriques de nœud s'accumulent au fur et à mesure lors du parcours du treillis. Il est donc nécessaire de les normaliser afin d'éviter leur débordement. A ce sujet, différentes techniques de normalisation sont résumées dans ce chapitre. Au niveau architectural, nous avons présenté la technique *sliding windows* permettant à la fois de réduire le délai et la complexité du décodage ainsi que la taille de mémoire. La mémoire, quant à elle, est une source dominante au niveau de la consommation en particulier dans un turbo-décodeur. C'est pourquoi nous avons mené une étude sur la mémorisation des métriques de nœud. D'après la description de l'algorithme Max-Log-MAP, seule la différence des métriques de nœud au cours du décodage permet de fournir une décision finale. Dans ce cas, une métrique de nœud ayant une valeur élevée ne contribue pas à la décision finale du décodage. Il est alors possible de saturer ces métriques sans dégrader les performances en terme de TEB. Le grand intérêt de la saturation réside dans le fait que le nombre de bits nécessaires à la quantification des métriques de nœud peut être diminué, d'où une réduction de la taille de mémoire et de la consommation associée.

Au niveau de la réalisation, nous avons proposé deux possibilités : Saturation A l'Extérieur (SAE) et Saturation A l'Intérieur (SAI) du calcul récursif. L'application d'une SAE ne modifie pas les résultats issus des opérateurs ACS. Dans ce cas, la normalisation des métriques de nœud est appliquée au cours du parcours de treillis. Cette méthode ne modifie pas le chemin critique. Sur la méthode SAI, les valeurs saturées des résultats de l'opérateur ACS sont prises pour le calcul suivant du prochain étage. Dans ce cas, il est alors possible d'utiliser la technique pipeline pour réduire le délai du décodage. Les deux méthodes présentent les mêmes performances en terme de TEB. Il est à noter que la diminution de la taille de mémoire est obtenue au prix d'une légère augmentation du coût matérielle pour effectuer la saturation. Nous obtenons 0.1dB de dégradation pour un turbo-décodeur du système UMTS lorsque le nombre de bits des métriques de nœud est diminué de 7 bits à 4 bits pour un TEB de l'ordre de 10^{-6}.

Pour mesurer son impact sur la consommation énergétique, nous avons tout d'abord présenté l'outil d'estimation CACTI. Ensuite, la méthode proposée a été appliquée à un turbo-décodeur spécifié dans la norme UMTS. Les résultats des estimations sur la surface de mémoire et l'énergie consommée ont été présentés. Ces résultats sont fonction du nombre de fenêtres considérées. Grâce à notre méthode, nous obtenons respectivement une réduction de 25% sur la surface des mémoires totales et une réduction de 31% sur l'énergie consommée par un accès mémoire (écriture et lecture).

La saturation des métriques de nœud est aussi applicable au turbo-décodeur différentiel

(voir le chapitre 2). Dans ce cas, nous pouvons tirer profit non seulement de l'approche sur la saturation des métriques de nœud pour diminuer la taille de mémoire, mais aussi du décodage différentiel pour baisser l'activité du système. Le chapitre suivant détaille les architectures d'un turbo-décodeur différentiel dédié au système UMTS.

CHAPITRE

4 Architecture du Turbo Décodage Différentiel et Insertion d'Erreurs

DANS ce chapitre, nous décrivons les architectures permettant de réaliser le décodage différentiel, qui consiste à effectuer un codage à partir des symboles reçus avant de procéder au décodage proprement dit. Dans un premier temps, nous faisons dans la section 4.1 un bref rappel sur le rôle des turbocodes dans une chaîne de communications numériques. La gestion des mémoires et l'organisation des données sont respectivement présentées dans les sections 4.2 et 4.2.1. A ce propos, nous expliquons dans la section 4.2.1 l'organisation des données à la réception, Nous décrivons l'architecture globale d'un turbo-décodeur dans la section 4.2.3.

Dans un deuxième temps, nous proposons l'architecture d'un décodeur élémentaire SISO (Soft Input Soft Output en anglais) (*c.f.* la section 4.3). La section 4.4 est consacrée à la solution architecturale d'un turbo-décodeur différentiel, qui est composé d'un codeur convolutif et d'un décodeur élémentaire SISO. Nous détaillons les techniques d'insertion d'erreurs et d'estimation du chemin survivant au niveau architectural. Puis, nous donnons la solution architecturale pour réaliser le calcul anticipé de l'information extrinsèque.

4.1 Turbocodes dans une chaîne de communications numériques

Le rôle des turbocodes est de protéger les messages émis de manière à augmenter les performances du système en terme de TEB. De manière générale, la source numérique est d'abord codée dans un codeur convolutif (figure 4.1). Ensuite, ce message codé est modulé et transmis à travers un canal de transmission (ABBG dans notre cas). Nous considérons une MDP2 (Modulation De Phase à 2). Du côté récepteur, une démodulation cohérente MDP2 est d'abord appliquée au message reçu. Ensuite le turbo-décodeur fournit aux destinataires une estimation dure sur le message transmis à partir des données démodulées.

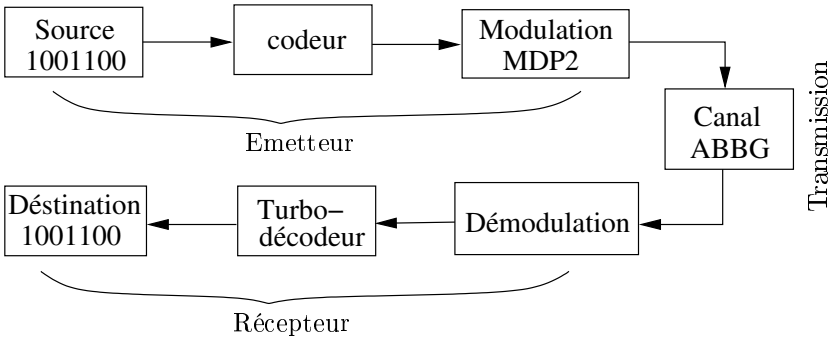


Figure 4.1 — Structure d'une chaîne de communications numériques.

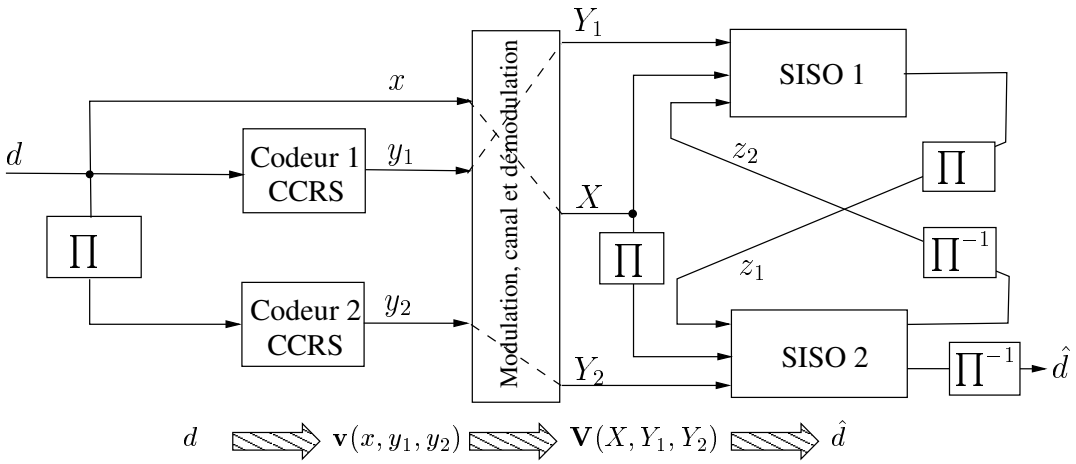


Figure 4.2 — Système d'un turbo-(dé)codeur adapté à la norme UMTS.

Nous nous intéressons aux parties codage et turbo-décodage dans cette thèse. C'est pourquoi la chaîne de communications numériques peut être modélisée par trois composants élémentaires : codeur, canal de transmission et turbo-décodeur (figure 4.2). Ciblant le standard UMTS, nous utilisons une concaténation en parallèle d'un code de type CCRS (Code Convolutif Récursif Systématique). Le codeur 1 reçoit le message dans l'ordre naturel, tandis que le codeur 2 reçoit le message dans l'ordre entrelacé (voir l'entrelaceur de la figure 4.2). A l'issu d'un tel codage, un mot codé $\mathbf{v}(x, y_1, y_2)$, comprenant une partie systématique x et deux parties redondantes y_1 et y_2, est construit à partir de la donnée entrante d. Les redondances y_1 et y_2 sont respectivement produites par les codeurs 1 et 2. Quant à la fermeture de treillis, la technique *tail biting* est appliquée à la fin du codage de chaque trame (voir Chapitre 1).

Dans notre cas, chaque codeur produit 3 *tail bits*, ce qui correspond à 2^3 états. L'envoi d'une trame codée suit l'ordre suivant :

$$x_0,\ y_{1,0},\ y_{2,0},\ \cdots,\ x_{L-1},\ y_{1,L-1},\ y_{2,L-1},\ x_L,\ y_{1,L},\ x_{L+1},\ y_{1,L+1},\ x_{L+2},\ y_{1,L+2},\ x_{L+3},\ y_{2,L},\ x_{L+4},\ y_{2,L+1},\ x_{L+5},\ y_{2,L+2}$$

Les symboles ($x_L\ \cdots\ y_{2,L+2}$) indiquent les envois des *tail bits* produits dans les deux codeurs respectifs.

Le message codé est donc successivement modulé, bruité et démodulé avant d'entrer dans le turbo-décodeur. Ce turbo-décodeur, quant à lui, est composé de deux décodeurs élémentaires SISOs, auxquels le processus itératif s'applique. Chaque décodeur SISO produit une information extrinsèque z à partir du mot bruité correspondant et de l'information extrinsèque z' produite par le décodeur concurrent. C'est alors que chaque décodeur SISO tire profit de l'information extrinsèque du décodage précédent. L'échange d'informations extrinsèques joue un rôle primordial pour le turbo-décodage. Cet échange permet d'aboutir à de bonnes performances du décodage après un certain nombre d'itérations. A l'issu d'un tel décodage itératif, le turbo-décodeur fournit une estimation dure $\hat{d}$ sur le message émis à partir du mot reçu $\mathbf{V}(X, Y_1, Y_2)$. Le traitement d'un décodeur élémentaire SISO est appelé *demi-itération*. Cela signifie que l'information extrinsèque pour chaque bit considéré n'est mise à jour qu'après le décodage d'une demi-itération. Par convention, les informations extrinsèques entrantes sont initialisées à 0 lors de la première demi-itération.

4.2 Gestion des mémoires de données

Au cours de la réception, les messages reçus sont dans un premier temps stockés dans les mémoires de données, notées $\mathbb{M}_X$, $\mathbb{M}_{Y_1}$ et $\mathbb{M}_{Y_2}$, respectivement allouées pour les éléments reçus X, Y_1 et Y_2. De plus, une mémoire $\mathbb{M}_z$ est mise en place pour stocker les informations extrinsèques z. Quant à l'entrelaceur, les adresses entrelacées sont stockées dans une ROM (Read Only Memory en anglais). Ainsi, la première tâche à prendre en considération est la gestion des mémoires au niveau des accès en écriture et en lecture pour éviter les conflits potentiels. Cette gestion assure le bon fonctionnement de l'architecture proposée dans la suite de ce chapitre. Dans cette section, nous discutons tout d'abord de la gestion des données dans les mémoires. Ensuite, la gestion des accès mémoires (lecture et écriture) est décrite pour les différentes mémoires présentées ci-dessus.

4.2.1 Organisation des données sur les accès mémoires

Pour la mémorisation des données, l'organisation des accès mémoires est primordiale. De manière générale, les données sont écrites dans la mémoire grâce à un bus de données dont la taille correspondant à celle d'un mot à mémoriser. De plus, un bus d'adresse indique l'emplacement des données à stocker. Dans notre cas, la taille de la mémoire dépend directement de la longueur de la trame à mémoriser. Dans le cas d'une organisation structurée de la mémoire, chaque écriture (lecture respectivement) incrémente (décrémente respectivement) l'adresse courante d'une unité. Afin de garantir un traitement en temps réel, une donnée doit être écrite dans la mémoire correspondante dès sa réception. Par conséquent, l'ancienne donnée stockée à l'adresse indiquée est écrasée. Il est ainsi indispensable de prendre en compte le temps de traitement nécessaire pour décoder une trame afin d'éviter la perte de données. La figure 4.3 illustre l'organisation de l'écriture d'une trame dans la mémoire. Nous supposons

qu'une écriture prend un cycle d'horloge. De l'instant 0 à l'instant $L+5$, les données reçues sont alors successivement écrites dans la mémoire de l'adresse 0 à l'adresse $L+5$, en tenant compte des 6 *tail bits*.

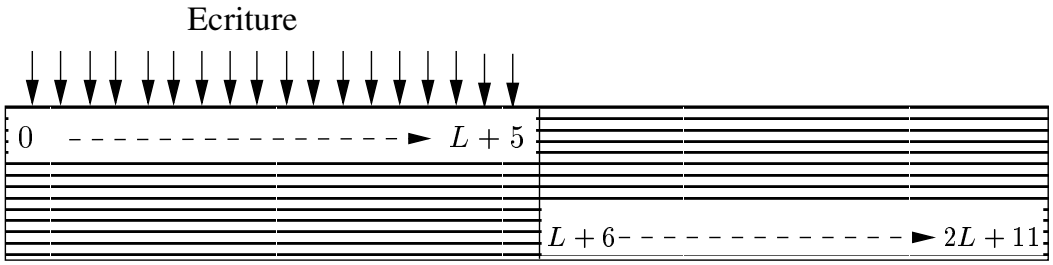


Figure 4.3 — Ecriture des données de l'instant 0 à l'instant $L+5$.

Supposons que les trames de données soient transmises sans interruption. Une fois que la première trame est stockée, l'écriture de la deuxième trame de l'adresse $L+6$ à l'adresse $2L+11$ succède immédiatement à celle de la première trame suivant la réception de données. Ainsi, la deuxième trame est stockée dans la deuxième partie de la mémoire. En parallèle, le décodeur traite la première trame stockée de l'adresse 0 à $L+5$ (figure 4.4). Pour ce faire, nous pouvons utiliser deux mémoires ou une mémoire à double accès. L'entrée et la sortie fonctionnent à deux fréquences différentes. L'écriture suit la fréquence d'émission tandis que la lecture suit la fréquence du décodage. Pour résumer, le temps nécessaire pour décoder une trame ne doit pas dépasser celui pour écrire une trame. Dans le cas contraire, une mémoire plus grande s'avère nécessaire.

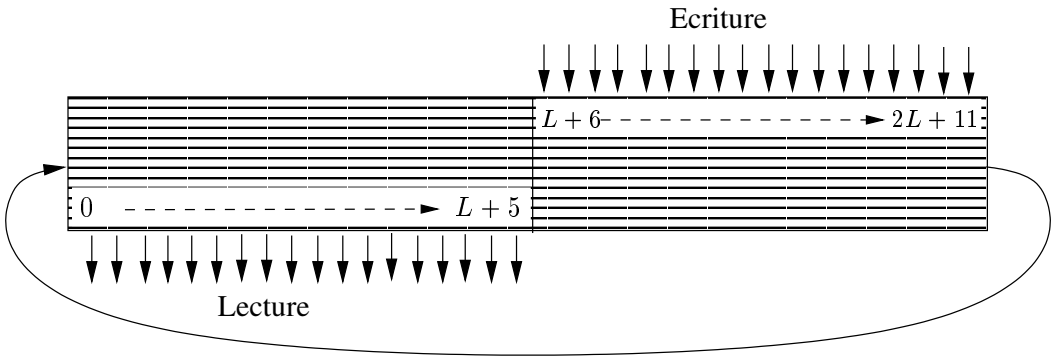


Figure 4.4 — Lecture de la première trame et écriture de la deuxième trame.

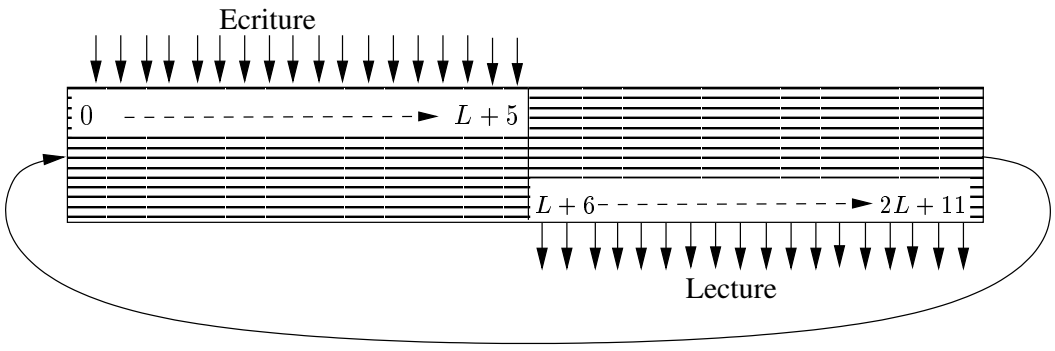


Figure 4.5 — Lecture de la deuxième trame et écriture de la troisième trame.

Pendant que la deuxième trame est écrite, le décodage de la première trame est réalisé. Le contenu des adresses 0 à $L+5$ est ensuite écrasé par une troisième trame (figure 4.5). De même, le décodage de la deuxième trame est effectué en parallèle. Ce processus est alors effectué sans cesse pendant la transmission.

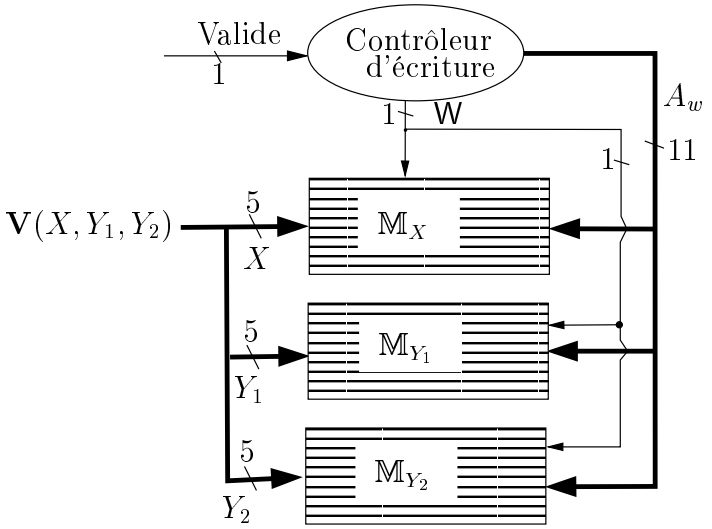


Figure 4.6 — Gestion de l'écriture des données.

Au niveau de la réalisation, nous fixons $L = 864$ bits, une des longueurs de trame de la norme UMTS. Nous utilisons 3 mémoires $\mathbb{M}_X$, $\mathbb{M}_{Y_1}$ et $\mathbb{M}_{Y_2}$ respectivement pour les données X et Y_1 et Y_2. Leur taille respective est égale à $2 \times ((L+6) \times 5)$ bits, soit 8700 bits (figure 4.6). Le nombre de bits nécessaires pour l'adresse est égal à 11 bits dans ce cas. Ces mémoires partagent le même contrôleur d'écriture. Lorsqu'un mot $\mathbf{V}(X, Y_1, Y_2)$ est disponible, le signal de contrôle (le signal *Valide* dans la figure 4.6) est alors activé ($Valide = 1$). Puis, le contrôleur active un signal d'écriture W (ceci correspond au mot Write en anglais) propagé sur les mémoires ainsi qu'une adresse associée A_w.

4.2.2 Mémoire ROM (Read Only Memory) pour le (dés)entrelaceur

L'architecture d'un (dés)entrelaceur ne fait pas l'objet de cette thèse. Il est néanmoins important de noter qu'une architecture adéquate favorise la maîtrise de la consommation. La règle d'entrelacement est souvent variable selon les applications. L'adresse entrelacée est fonction de la longueur de la trame dans la norme UMTS. Pour simplifier notre implémentation, la longueur de trame a été fixée. Il est donc possible d'utiliser une mémoire ROM, notée $\mathbb{M}_\prod$, pour stocker les adresses entrelacées d'un tel (dés)entrelaceur. A partir de la longueur L, ces adresses entrelacées sont obtenues à l'aide de simulation fonctionnelle d'après la norme UMTS. Pour représenter 870 adresses, un mot de la mémoire $\mathbb{M}_\prod$, qui correspond à une adresse entrelacée, est codé sur 10 bits. La taille du bus d'adresse est alors égale à 10 bits. Ainsi, la taille de la mémoire ROM est égale à 870×10 bits. Il est à noter qu'une lecture de la ROM prend un cycle d'horloge dans notre cas.

4.2.3 Lecture des données pour l'application de turbo-décodage

Au cours du décodage, les mémoires $\mathbb{M}_X$, $\mathbb{M}_{Y_1}$, $\mathbb{M}_{Y_2}$ et $\mathbb{M}_z$ fournissent les données nécessaires au décodeur élémentaire SISO. Le décodeur SISO, quant à lui, effectue le décodage à

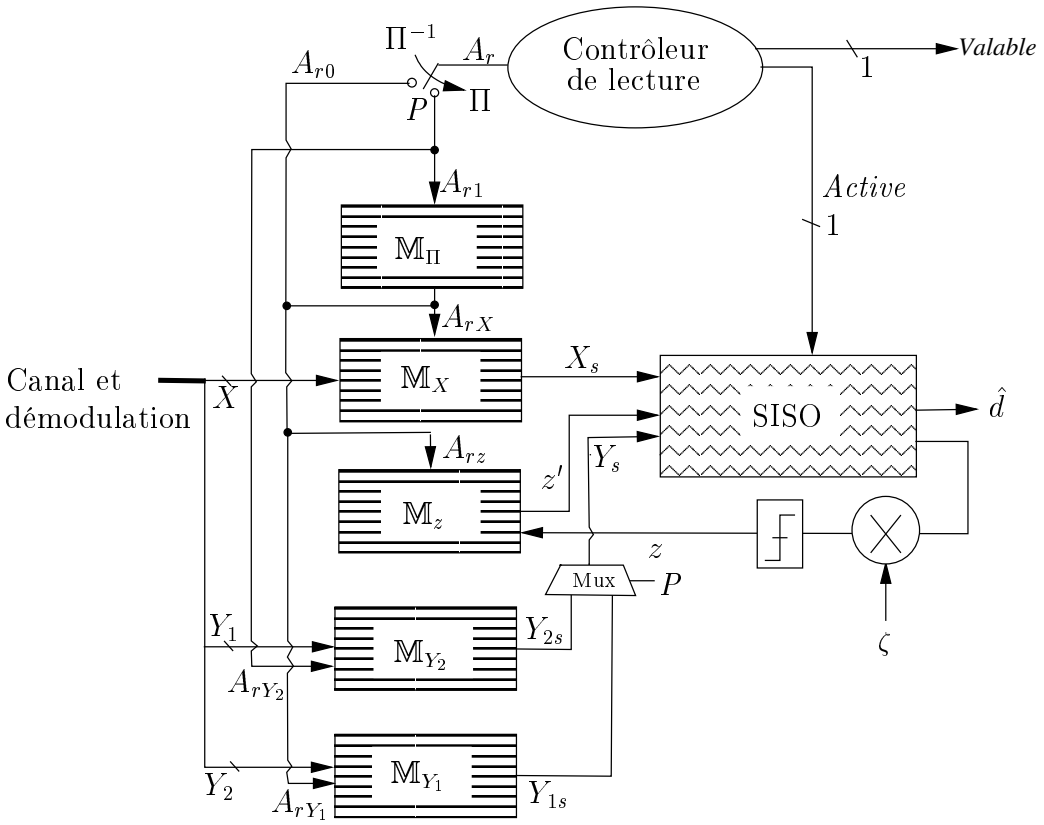


Figure 4.7 — Lecture des mémoires au cours d'une demi-itération de décodage.

partir des données X_s, Y_s et z' (figure 4.7). D'ailleurs, ce décodeur est contrôlé par le signal *Active*, qui lui-même est délivré par le contrôleur de lecture. Lorsque le signal *Active* est égal à 1, le décodeur SISO est à l'état *marche*, c'est-à-dire, le décodage est en cours. Dans le cas contraire ($Active = 0$), le décodeur ne traite aucune donnée. En pratique, nous implémentons un seul décodeur SISO qui travaille à la fois sur les données dans l'ordre naturel et celles dans l'ordre entrelacé.

La lecture des données dans les mémoires est décrite de la manière suivante. Dans l'ordre naturel, le signal P est connecté à la configuration $\prod^{-1}$. L'adresse de la lecture A_r est alors envoyée aux mémoires $\mathbb{M}_X$, $\mathbb{M}_z$ $\mathbb{M}_{Y_1}$. Grâce à la mise en place d'un multiplexeur, qui lui est aussi contrôlé par le signal P, la donnée Y_s à l'entrée du décodeur SISO reçoit la sortie Y_{1s} de la mémoire $\mathbb{M}_{Y_1}$. Dans l'ordre entrelacé, le signal P est connecté à la configuration $\prod$. Dans ce cas, les mémoires $\mathbb{M}_X$, $\mathbb{M}_z$ reçoivent une adresse entrelacée issue de la mémoire ROM $\mathbb{M}_\Pi$ à partir de l'adresse A_r. Enfin, la sortie Y_{2s} de la mémoire $\mathbb{M}_{Y_2}$ est sélectionnée pour l'entrée Y_s du décodeur SISO.

Le décodeur SISO, quant à lui, fournit à la fois l'information extrinsèque et la décision dure $\hat{d}$ sur le bit considéré. Au niveau du turbo-décodage, l'information extrinsèque issue du décodeur est d'abord multipliée par un facteur ζ et puis saturée à 7 bits à chaque itération (voir Chapitre 1). Elle est ensuite stockée dans la mémoire $\mathbb{M}_z$. La valeur du facteur ζ est variable selon l'itération : $\zeta = 0.75$ pour les deux premières itérations, 1 pour la dernière itéraiton, 0.5 pour le reste. Lorsque toutes les itérations du turbo-décodage sont terminées, le contrôleur active le signal *Valable* indiquant au composant suivant que l'information extrinsèque est disponible. En parallèle, la décision dure $\hat{d}$ est également fournie par le décodeur élémentaire SISO.

4.3 Architecture d'un décodeur SISO

4.3.1 Introduction

Le décodeur SISO est un composant élémentaire du turbo-décodeur. Dans cette section, nous présentons l'architecture d'un décodeur SISO appliquant l'algorithme Max-Log-MAP. Cette architecture est décomposée en deux parties : une partie contrôle et une partie traitement. La partie contrôle permet de synchroniser les composants tandis que la partie traitement s'applique aux opérations arithmétiques. Comme nous l'avons mentionné dans le chapitre 1, le décodeur SISO calcule l'information extrinsèque z et fournit une décision dure $\hat{d}$ sur le bit considéré à partir des données entrantes X, Y et z'.

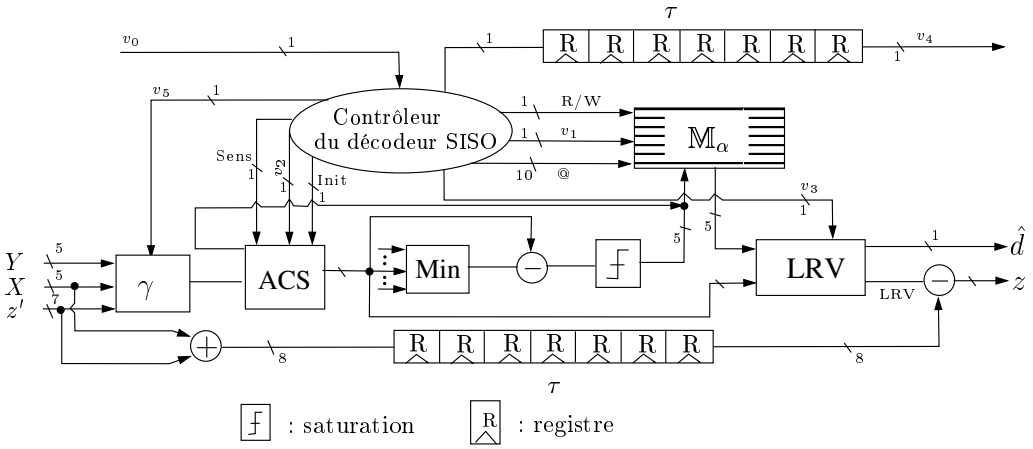


Figure 4.8 — Architecture d'un décodeur SISO appliquant l'algorithme Max-Log-MAP.

La figure 4.8 présente l'architecture globale d'un décodeur SISO dédié aux codes convolutifs. Cette architecture comprend les modules suivants. Un module se charge de calculer les métriques de branche γ à partir des entrées X, Y et z'. Ensuite, un module réalise l'opération ACS (Add Compare Select) à partir des métriques de branche et des métriques de nœud précédentes. La métrique de nœud minimale est recherchée dans un module « Min ». Puis, un module est dédié à la saturation des métriques de nœud. Enfin, le dernier module concerne le calcul du LRV (Logarithme du Rapport de Vraisemblance) et la décision dure $\hat{d}$. Un contrôleur du décodeur SISO est alors nécessaire pour réaliser l'ensemble de ces opérations.

Concernant le pas de quantification, les études du chapitre 3 ont adopté un pas de quantification $q = 0.38$. Pour faciliter l'implémentation, nous utilisons une représentation directe des données à l'issu du canal. Ainsi, nous avons adopté un pas de quantification $q = 2^{-3}$ dans la suite de la thèse pour éviter une ré-quantification après le bruitage du canal. Dans ce cas, pour obtenir de bonnes performances du décodage, les entrées du décodeur X, Y_1, Y_2 sont alors quantifiées sur 5 bits dont 3 bits sont dédiés à la partie fractionnaire. Ce format de quantification est noté (5,3). La quantification de l'information z' correspond alors au format $(7,3)$. Dans ce cas, les valeurs des métriques de nœud peuvent être saturées sur 5 bits selon la quantification des entrées sans dégrader les performances.

Nous supposons que le décodage se déroule d'abord dans le sens *Aller*, puis dans le sens *Retour*. Les métriques de nœud α sont alors stockées dans la mémoire $\mathbb{M}_\alpha$. Cette mémoire est contrôlée par un signal de validation v_1 et un signal de lecture et d'écriture R/W (correspondant aux mots Read/Write en anglais). Son adresse de lecture ou d'écriture est déterminée

par le contrôleur. Enfin, le module « LRV » est contrôlé par le signal v_3. Le délai τ est obtenu à partir des bascules de type Flip-Flop en concaténation série. Les différents modules de l'architecture sont successivement détaillés dans les sous-sections suivantes.

4.3.2 Contrôleur du décodeur SISO

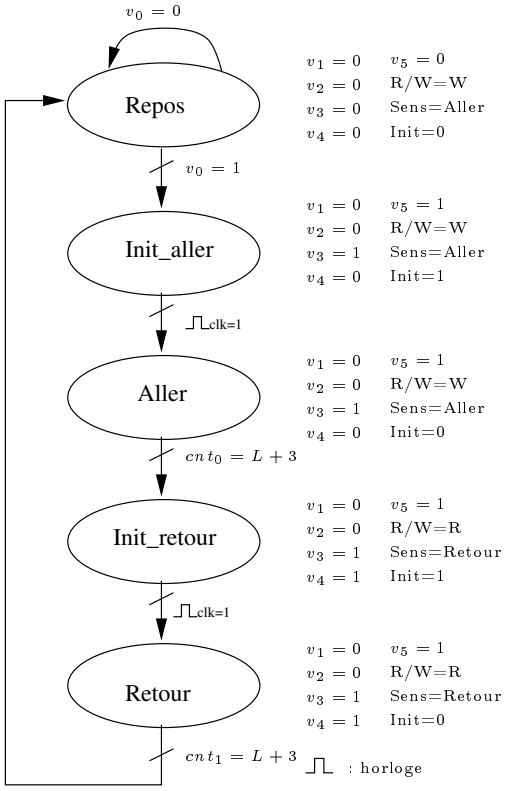


Figure 4.9 — Machine d'états modélisant le contrôleur du décodeur SISO.

Le contrôleur du décodeur SISO est modélisé par une MDE (Machine D'Etats). Nous distinguons 5 états dans ce cas : *Repos*, *Init_aller*, *Aller*, *Init_retour* et *Retour*. Les signaux de contrôle sont associés à chaque état. Le diagramme d'états est présenté dans la figure 4.9. Le contrôleur est initialisé à l'état *Repos*. Le décodeur SISO attend alors l'arrivée des données entrantes. Lorsque le signal v_0 est activé ($v_0 = 1$), soit les données attendues sont disponibles, le contrôleur passe à l'état suivant *Init_ aller*. Dans cet état, les métriques de nœud α sont initialisées à la valeur 0 pour l'état 0 du treillis et à une valeur maximale pour les autres états du treillis. Par ailleurs, les métriques de nœud α obtenues sont alors enregistrées dans la mémoire $\mathbb{M}_\alpha$, c'est-à-dire, $R/W = W$. Le cycle suivant, la machine d'états passe à l'état *Aller*. Dans cet état, le signal *Init* est désactivé puisqu'il s'agit du calcul récursif des métriques de nœud. Un compteur cnt_0 est mise en place pour compter le nombre d'étages parcourus sur le treillis. A la fin du parcours en tenant compte du traitement des *tail bits*, soit $cnt_0 = L + 3$, la machine d'états passe à l'état *Init_retour* pour recommencer le décodage dans le sens *Retour*.

A l'état *Init_retour*, toutes les métriques de nœud β sont alors initialisées à la valeur 0 pour tous les états du treillis ($Init = 1$). Comme pour le décodage dans le sens *Aller*, le signal *Init* est désactivé le cycle suivant. Un compteur cnt_1 compte le nombre d'étages parcourus dans le sens *Retour*. Les métriques de nœud α stockées dans la mémoire $\mathbb{M}_\alpha$ sont lues pour

calculer le LRV. Dans ce cas, le signal R/W est égal à R. Le module LRV est ensuite mis en service. En parallèle, le signal $v_4 = 1$ est activé pour indiquer que l'information extrinsèque est disponible. En effet, cette information extrinsèque est obtenue à l'aide d'une soustraction entre la somme $X + z$ retardée et la sortie LRV (figure 4.8). Une décision dure $\hat{d}$ pour le bit considéré est fourni par ce même module. Lorsqu'une demi-itération est terminée, soit $cnt_1 = L + 3$, la machine d'états revient à l'état initial *Repos* en attendant l'arrivée d'une nouvelle trame ou commençant l'itération suivante.

4.3.3 Architecture du module « calcul des métriques de branche »

Pour chaque étage du treillis, la première opération consiste à calculer les métriques de branche à partir des valeurs X, z' et Y. De plus, ce module donne une valeur positive ou nulle pour les métriques de branche afin de faciliter le processus de décodage. Nous avons alors 4 métriques de branche différentes pour un décodeur binaire, notées respectivement γ_{00}, γ_{01}, γ_{10}, γ_{11} dans le cas d'un code convolutif ayant un rendement $R = 1/2$. Leur valeur respective est obtenue à partir de l'expression suivante :

$$\gamma_{00} = X + z' + Y;\ \gamma_{01} = X + z' - Y;\ \gamma_{10} = -(X + z' - Y);\ \gamma_{11} = -(X + z' + Y) \quad (4.1)$$

Nous constatons que :

$$\gamma_{10} = -\gamma_{01};\ \gamma_{11} = -\gamma_{00} \quad (4.2)$$

Il est néanmoins possible de multiplier les termes de l'expression 4.1 par un facteur 1/2 sans dégrader les performances ([Pietrobon 98]). Nous obtenons alors l'expression suivante :

$$\gamma_{00} = \frac{X + z' + Y}{2};\ \gamma_{01} = \frac{X + z' - Y}{2};\ \gamma_{10} = -\frac{X + z' - Y}{2};\ \gamma_{11} = -\frac{X + z' + Y}{2} \quad (4.3)$$

La relation 4.2 reste toujours valable. En effet, cette multiplication permet de gagner un bit de quantification au niveau de la complexité. Pour la mise en œuvre, il suffit de décaler un bit vers la gauche au sein de l'opération pour réaliser une division par 2. D'après l'algorithme Max-Log-MAP, seule la différence des métriques de nœud détermine la décision finale. C'est pourquoi nous pouvons ajouter une constante V_{const} à chaque terme de l'expression 4.3 sans modifier le résultat final. Cet ajout permet d'obtenir des valeurs supérieures ou égales à 0 pour les métriques de branche.

Au sein du décodeur, la représentation décimale des métriques de branche dépend de celle des entrées X, Y et z'. Nous pouvons exprimer le nombre de bits nécessaires N_{bm} pour les métriques de branche de la manière suivante :

$$N_{bm} = \left\lceil \log\left(\frac{X + z' + Y}{2}\right) \right\rceil \quad (4.4)$$

où $\lceil x \rceil$ représente le plus petit entier supérieur ou égal à x. Les entrées X, Y et z sont considérées comme des valeurs signées. A propos de leur quantification, les entrées X et Y sont quantifiées sur 5 bits et les informations extrinsèques z' sont quantifiées sur 7 bits (*c.f.* la section 4.3.1). La représentation décimale des métriques de branche peut alors s'exprimer dans la plage [-48,+48]. Il suffit alors de quantifier les métriques de branche sur 7 bits à partir de l'expression 4.4.

La figure 4.10 présente une architecture pour calculer les métriques de branche, à savoir qu'elles sont calculées lors du décodage dans les deux sens (*Aller* et *Retour*). La somme

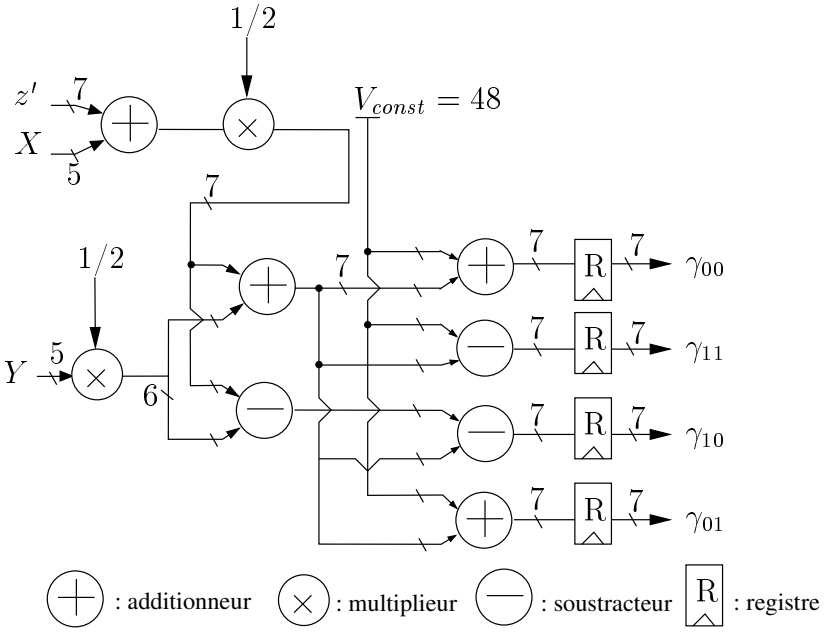


Figure 4.10 — Architecture pour le calcul des métriques de branche.

$X + z'$ et la donnée Y sont multipliées par un facteur 1/2. Un additionneur et un soustracteur réalisent respectivement les opérations $\frac{(X+z')+Y}{2}$ et $\frac{(X+z')-Y}{2}$. Enfin, une constante 48 est respectivement ajoutée à la somme $\frac{(X+z')+Y}{2}$ et $\frac{(X+z')-Y}{2}$ pour obtenir les métriques de branche γ_{00} et γ_{01}. De même, des soustractions respectives entre la constante et les sommes $\frac{(X+z')+Y}{2}$ et $\frac{(X+z')-Y}{2}$ permettent d'obtenir les métriques de branche γ_{11} et γ_{10}. Dès lors, les valeurs des métriques de branche sont positives ou nulles. C'est pourquoi, nous considérons une représentation non-signée pour ces valeurs à l'issu des telles opérations. De plus, elles sont quantifiées sur 7 bits dans un intervalle non-signé [0,96] en représentation décimale.

4.3.4 Architecture du module ACS (Add Compare Select)

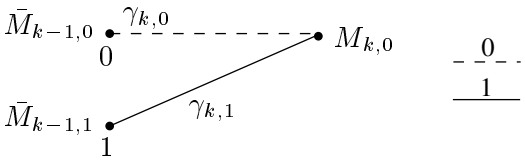


Figure 4.11 — Opération élémentaire ACS.

L'opération élémentaire ACS représentée par la figure 4.11 peut être exprimée de la manière suivante :

$$\begin{aligned} M^e_{0,k} &= \bar{M}_{k-1,0} + \gamma_{k,0}, \\ M^e_{1,k} &= \bar{M}_{k-1,1} + \gamma_{k,1}, \\ M_{0,k} &= \min\left(M^e_{k,0}, M^e_{k,1}\right). \end{aligned} \tag{4.5}$$

où les symboles $M^e_{k,0}$ et $M^e_{k,1}$ représentent respectivement les métriques de noeud associées aux entrées 0 et 1 à l'instant k sur l'état e, tandis que les symboles $\gamma_{k,0}$ et $\gamma_{k,1}$ représentent respectivement les métriques de branche associées aux entrées 0 et 1. La métrique de nœud saturée est notée $\bar{M}$. La métrique de nœud $M_{k,0}$ finale est obtenue en sélectionnant la valeur

minimale parmi les deux métriques de noeud $M^e_{k,0}$ et $M^e_{k,1}$.

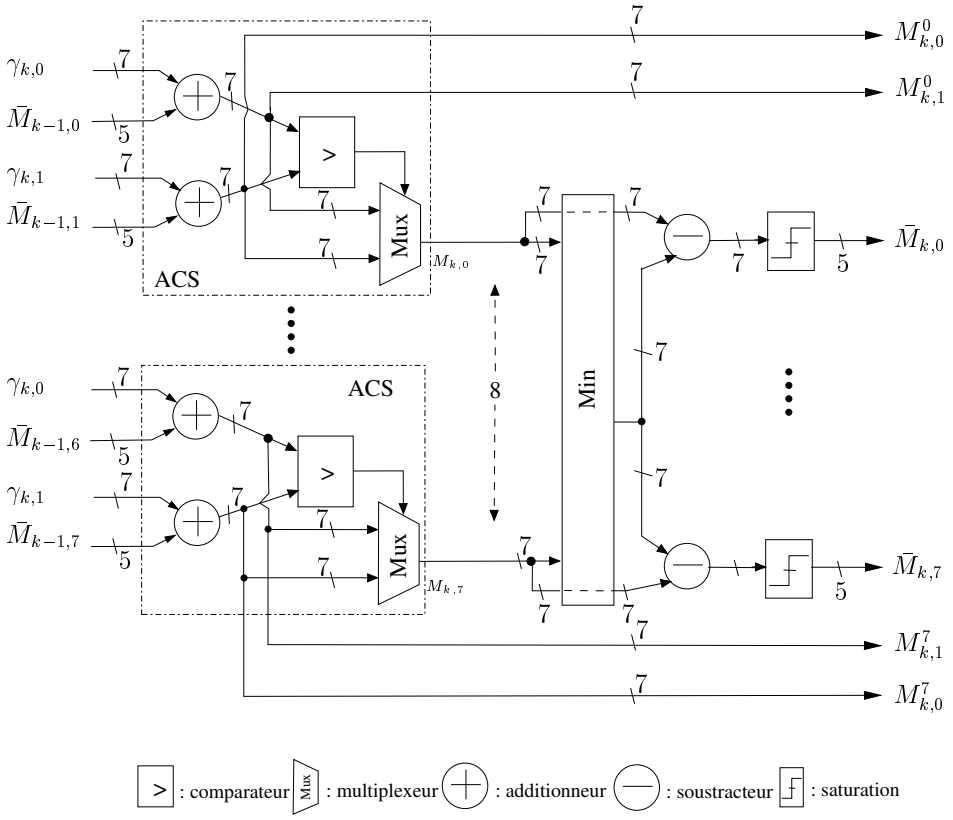


Figure 4.12 — Architecture pour l'opération élémentaire ACS.

Rappelons que les valeurs décimales γ au sein du décodeur sont représentées dans la plage [0, 96] dans notre cas (voir la section 4.3.3). Les entrées X et Y sont quantifiée sur 5 bits en utilisant une représentation directe de données. Il est alors possible de saturer les métriques de nœud au cours du décodage en appliquant la méthode présentée dans le chapitre 3. Lorsque les métriques de nœud M_{k-1} sont saturées sur 5 bits, notée $\bar{M}_{k-1}$, (*c.f.* la section 4.3.1), il suffit de quantifier les métriques de nœud $M^0_{0,k}$ et $M^0_{1,k}$ sur 7 bits, car la somme $\gamma_k + \bar{M}_{k-1}$ ne dépasse pas la valeur 128 pour une représentation non-signée.

La figure 4.12 représente l'architecture correspondant à l'opération ACS. La valeur $M^e_{0,k}$ ($M^e_{1,k}$ respectivement) pour $e \in \{0 \cdots 7\}$ est obtenue à l'aide d'un additionneur, sur lequel est réalisée l'opération $M_{0,k-1} + \gamma_{k,0}$ ($M_{1,k-1} + \gamma_{k,1}$ respectivement). La métrique de nœud $M_{0,k}$ est ensuite sélectionnée à l'aide d'un comparateur. Ainsi, l'expression ACS est mise en œuvre à l'aide des deux additionneurs, un comparateur et un multiplexeur (voir le bloc pointillé dans la figure 4.12).

Les métriques de nœud $M^e_{0,k}$ et $M^e_{1,k}$ servent à calculer le LRV lors du second parcours du treillis tandis que les métriques de nœud $M_{0,k}$ sont normalisées pour le prochain étage de décodage. Nous appliquons la normalisation dynamique (voir Chapitre 3) dans notre implémentation. C'est pourquoi un module « Min » (voir la section suivante) et un soustracteur sont également mis en place pour effectuer cette normalisation. La normalisation dynamique permet d'avoir un nombre minimal, 7 bits dans notre cas, pour la quantification des métriques de nœud sans avoir d'impact sur les performances. La métrique de nœud $M_{k,0}$ peut elle encore être saturée sur 5 bits, notée $\bar{M}_{k,0}$ (*c.f.* la section 4.3.1). Ainsi, la taille de la mémoire $\mathbb{M}_\alpha$ (figure 4.8) est réduite de $L \times 7$ bits à $L \times 5$ bits. Pour chaque étage de décodage, il est possible

d'implémenter 8 ACS en parallèle dans le cas d'un treillis de 8 états.

4.3.5 Architecture du module « Min »

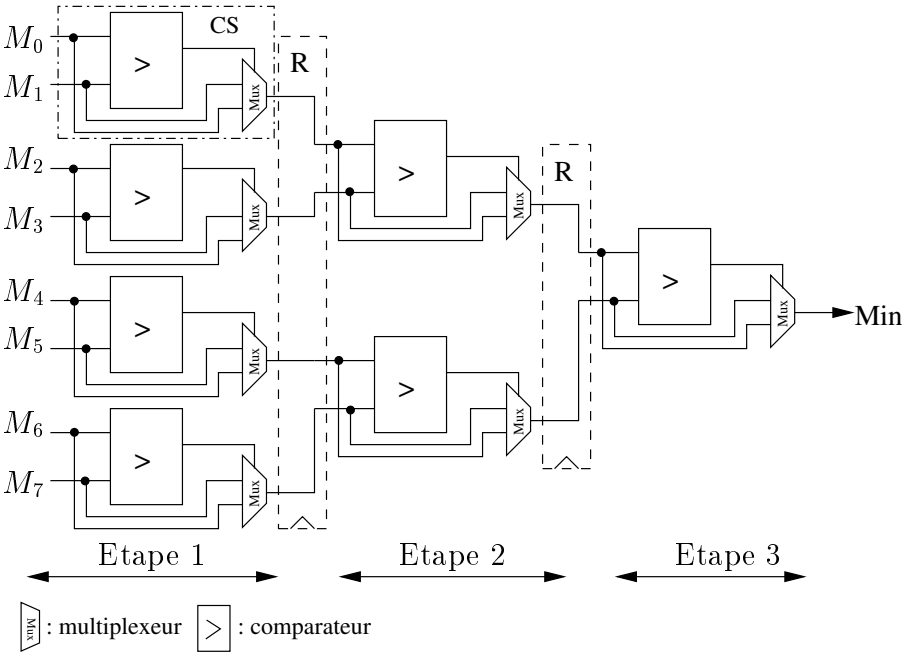


Figure 4.13 — Architecture du module « Min » pour un code à 8 états.

Le module « Min » permet de trouver la valeur minimale parmi les métriques de nœud à chaque étage du treillis. En pratique, cela correspond à un tri des valeurs d'un arbre. La figure 4.13 représente l'architecture constituée des composants CS (Comparaison et Sélection). Le composant CS est lui-même composé d'un comparateur et d'un multiplexeur. Il permet à chaque opération de sélectionner le minimum parmi les deux entrées. En général, $N_{cs} - 1$ composants CS sont nécessaires pour trier N_{cs} valeurs.

De manière générale, l'architecture d'un tel tri peut être divisée en N étapes avec $2^N = N_e$, nombre d'états du codeur. Dans notre cas, $N_e = 8$ et $N = 3$. Nous pouvons alors sélectionner 4 minimums durant la première étape, 2 minimums durant la deuxième étape et le résultat final est obtenu durant la troisième étape. Il y a au total 7 composants CS. Afin de diminuer le chemin critique, il est possible d'insérer une file de registres entre deux étapes (composants pointillés dans la figure 4.13).

4.3.6 Architecture de la saturation

La saturation permet d'éliminer les débordements de valeurs dans une plage définie. Ainsi, il est possible de diminuer le nombre de bits nécessaires à la quantification des métriques de nœud. La figure 4.14 présente l'architecture de la saturation des métriques de nœud. Rappelons que les valeurs considérées sont non-signées. La métrique de nœud α (β respectivement) à l'entrée du bloc est quantifiée sur 7 bits. Les deux MSBs sont observés par une porte logique ou. Si tous les MSBs sont 0, les 5 LSBs (Lowest Significant Bit) sont alors propagés pour la sortie $\bar{\alpha}$ ($\bar{\beta}$ respectivement). Dans le cas contraire, tous les bits de la sortie sont mis à 1 à l'aide des portes logiques ou.

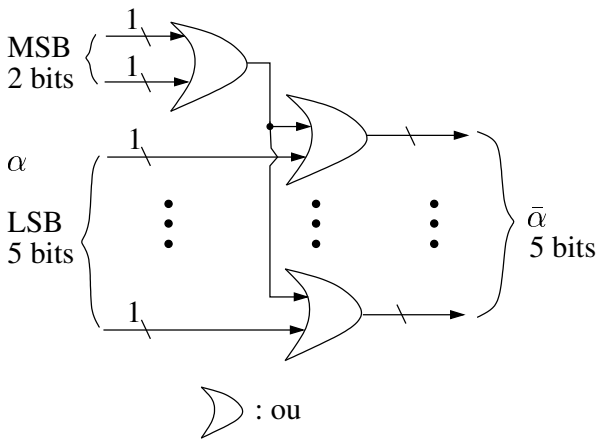


Figure 4.14 — Architecture pour l'opération saturation.

4.3.7 Architecture pour le calcul du LRV (Logarithme du Rapport de Vraisemblance)

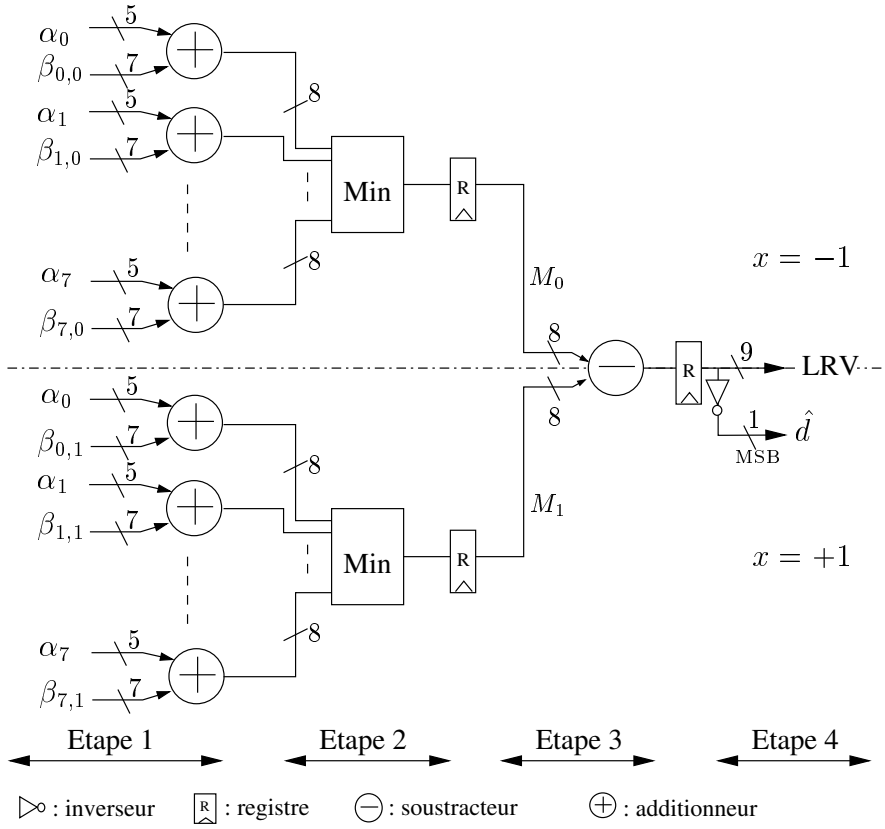


Figure 4.15 — Architecture pour le calcul du LRV.

La dernière étape de l'algorithme Max-Log-MAP consiste à calculer le LRV et à fournir une décision dure en parallèle. Comme montré dans le chapitre 1, le LRV $L(x_k)$ est obtenu à

partir de l'expression suivante :

$$\begin{aligned} L(x_k) = & \min_{\substack{(s',s) \\ x_k=+1}} \alpha_{k-1}(s') + \gamma_k(s',s) + \beta_k(s) \\ - & \min_{\substack{(s',s) \\ x_k=-1}} \alpha_{k-1}(s') + \gamma_k(s',s) + \beta_k(s). \end{aligned} \tag{4.6}$$

En fait, le LRV est égal à la différence de deux valeurs minimales produites par les sommes des métriques. Il est alors nécessaire de trouver respectivement les métriques de nœud minimales pour $x = +1$ (équivalent à une valeur dure $d = 1$) et $x = -1$ (équivalent à une valeur dure $d = 0$). Les calculs sont alors similaires pour $x = -1$ et $x = +1$. Lors du second parcours du treillis, la somme $\gamma_k(s', s) + \beta_k(s)$ est notée $\beta_{e,0}$ (respectivement $\beta_{e,1}$) pour $x = -1$ ($x = +1$), où le symbole e représente l'état du treillis. Ceci est produit par le module ACS. Dans la suite, nous détaillons la recherche d'une valeur minimale pour le terme $x = -1$ de l'expression 4.6.

La première étape consiste à calculer l'ensemble des sommes $\{\alpha_e + \beta_{e,0}\}$. Elles sont implémentées à l'aide de 8 additionneurs dans notre cas. Puis, nous cherchons la valeur minimale M_0 parmi les 8 sommes à l'aide du 8 modules « Min ». En parallèle, la valeur minimale concurrente M_1 est recherchée et stockée. La troisième étape consiste à effectuer une soustraction entre les valeurs M_0 et M_1 pour obtenir le LRV ainsi que la décision dure correspondante $\hat{d}$. En fait, cette décision dure est égale à l'inverse du signal MSB du LRV.

4.4 Architecture d'un turbo-décodeur différentiel

Dans cette section, nous détaillons les architectures d'un turbo-décodeur différentiel. Cette architecture est basée sur un codeur différentiel et un décodeur élémentaire SISO. Dans un premier temps, nous présentons l'architecture d'un codeur convolutif bidirectionnel, qui permet de parcourir le même chemin du treillis dans les deux sens. Dans un second temps, la détection du motif répétitif est présentée au niveau architectural. Enfin, l'architecture d'un décodeur différentiel SISO, appelé *architecture SISO-D* pour la différencier de l'architecture classique d'un décodeur SISO, est présentée. Cette architecture prend non seulement en compte l'estimation du chemin TAZ, mais aussi le calcul anticipé de l'information extrinsèque.

4.4.1 Rappel sur le principe du décodage différentiel

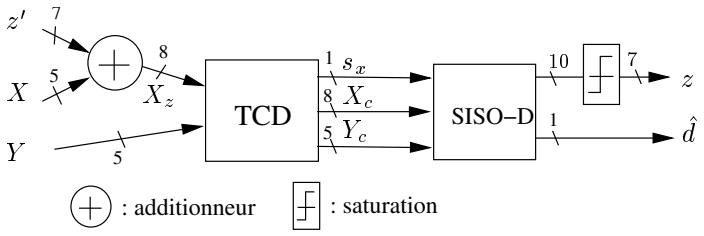


Figure 4.16 — Structure d'un bloc de décodage différentiel.

Le turbo-décodage différentiel consiste d'abord à appliquer un codage convolutif à partir de la séquence reçue. Pour ce faire, un codeur convolutif est inséré avant de procéder au décodage SISO. La figure 4.16 présente la structure en bloc de décodage différentiel. Elle

comprend deux blocs principaux : un bloc TCD (Turbo Codeur Différentiel) et un bloc SISO-D, qui permet de prendre en compte le calcul anticipé de l'information extrinsèque z. Tout d'abord, le bloc TCD reçoit la partie redondance Y et la somme entre la partie systématique X et l'information extrinsèque z' venant de l'itération du décodage précédent. Cette somme $X + z$, notée X_z, est quantifiée sur 8 bits. Le bloc TCD produit les résultats s_x, X_c, Y_c. Le bloc suivant SISO-D génère l'information extrinsèque z et calcule la décision dure $\hat{d}$. Le signal s_x est propagé pour effectuer une opération ou-exclusif servant à reconstituer l'information extrinsèque et la décision dure au sein du décodeur SISO-D (voir la section suivante). Il est à noter que l'information extrinsèque issue du bloc SISO-D est quantifiée sur 10 bits puis saturée sur 7 bits. Les différents blocs de la figure 4.16 sont détaillés dans les sous-sections suivantes.

4.4.2 Codeur convolutif bidirectionnel

Le codeur convolutif ayant pour polynôme générateur $G[1, (1 + D + D^3)/(1 + D^2 + D^3)]$ est considéré dans la norme UMTS. Ce codeur peut avoir deux présentations graphiques différentes (figure 4.17). Dans la figure 4.17(a), nous constatons que :

$$y = a \oplus b \oplus e. \tag{4.7}$$

Or,

$$a = f \oplus g = f \oplus (c \oplus e). \tag{4.8}$$

Nous en déduisons que :

$$y = f \oplus b \oplus c. \tag{4.9}$$

Pour la partie récursive, nous obtenons :

$$a = f \oplus g = f \oplus (e \oplus c). \tag{4.10}$$

La 4.17(b), qui est le modèle implémenté dans la thèse, représente la structure du codeur correspondant aux expressions 4.9 et 4.10. Le trait pointillé n'est utilisé que pour la fermeture du treillis. Comme précédemment indiqué, la technique *tail biting* est alors appliquée dans ce cas.

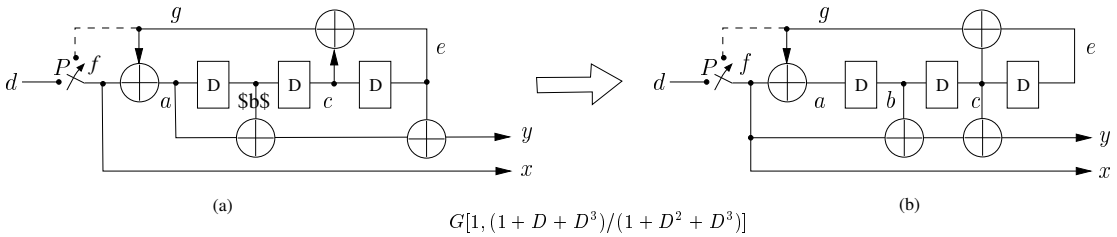


Figure 4.17 — Codeur convolutif ayant pour polynôme générateur $G[1, (1 + D + D^3)/(1 + D^2 + D^3)]$ pour le codage dans le sens *Aller*.

Le décodage Max-Log-MAP se décompose en deux étapes : décodage dans le sens *Aller* et décodage dans le sens *Retour*. La métrique de branche est re-calculée au cours du deuxième décodage. Pour effectuer le codage différentiel dans le sens *Retour*, il est alors nécessaire de remonter le même chemin du treillis dans l'ordre inverse par rapport au codage dans le sens *Aller*. Cela implique que le polynôme générateur du codeur dans le sens *Retour* devient $G[1, (1 + D^2 + D^3)/(1 + D + D^3)]$. Ce polynôme générateur du codeur conduit à avoir une

représentation graphique de la figure 4.18(a).

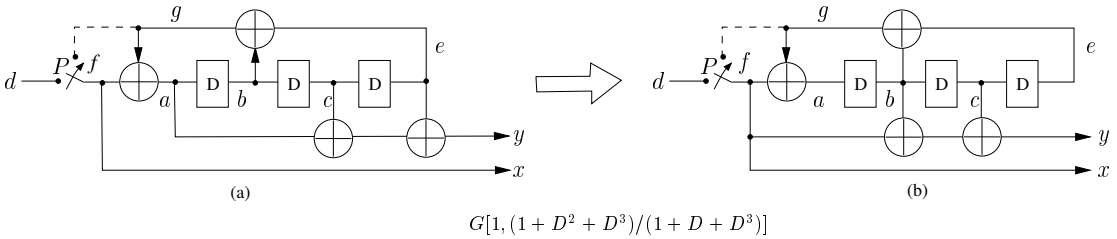


Figure 4.18 — Codeur convolutif ayant pour polynôme générateur $G[1,(1+D^2+D^3)/(1+D+D^3)]$ pour le codage dans le sens *Retour*.

De même, le codeur de la figure 4.18(a) peut être transformé à celui de la figure 4.18(b). En comparant les figures 4.17(b) et 4.18(b), nous constatons que la valeur du signal g est attribuée en fonction du sens indiqué. Il suffit alors de modifier le signal g pour que le codage dans le sens *Retour* puisse parcourir le même chemin du treillis dans l'ordre inverse.

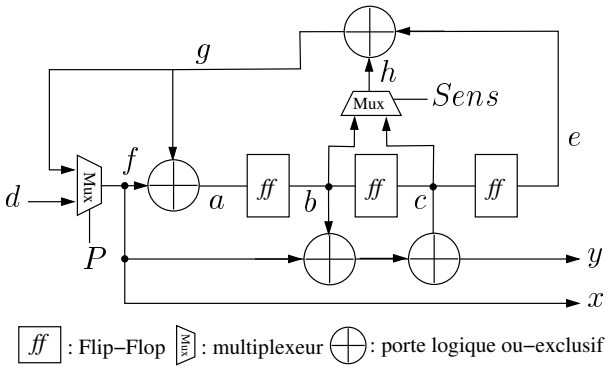


Figure 4.19 — Architecture dédiée au codeur convolutif bidirectionnel.

La figure 4.19 représente l'architecture du codeur, qui combine à la fois le codeur ($G[1,(1+D^2+D^3)/(1+D+D^3)]$) et le codeur ($G[1,(1+D+D^3)/(1+D^2+D^3)]$). Ce codeur est appelé *codeur convolutif bidirectionnel*. En effet, un multiplexeur permet de sélectionner la valeur du signal g en fonction du sens sélectionné. Pour le codage dans le sens *Aller*, le signal c est sélectionné pour la sortie h. Sinon, le signal b est sélectionné pour le codage dans le sens *Retour*.

Pour produire les *tail bits* à la fin du codage, un multiplexeur permet de sélectionner soit le signal g, soit la donnée entrante d pour le signal f. Un compteur est mis en place pour compter le nombre de bits codés. Le signal de contrôle P est activé ($P=1$) lorsque le codage d'une trame est terminé. La valeur g est alors sélectionnée lorsque $P=1$ pour produire les *tail bits*. Dans le cas contraire, la donnée entrante d est choisie pour générer la partie systématique x. Le décalage des bits est réalisé à l'aide de 3 bascules de type Flip-Flop en série (voir les signaux a, b et c de la figure 4.19).

4.4.3 Architecture pour le codage différentiel et l'insertion d'erreurs

Le codage différentiel consiste à effectuer une opération ou-exclusif entre le mot reçu $V(X,Y)$ et le mot re-codé $V'(X',Y')$ à partir du mot reçu. Nous avons alors :

$$\bar{V}(\bar{X},\bar{Y}) = V(X,Y) \oplus V'(X',Y'), \tag{4.11}$$

où le symbole $\bar{V}(\bar{X}, \bar{Y})$ représente le mot obtenu après le codage différentiel. Il est alors possible de faire apparaître le motif répétitif dans le terme $\bar{Y}$. En effet, ce motif est généré par les erreurs de transmission. Le principal problème réside dans le fait que le chemin survivant diverge du chemin TAZ à cause du motif au cours du décodage. Pour lutter contre cette divergence, une méthode efficace consiste à insérer des erreurs supplémentaires en détectant le motif apparu dans le terme $\bar{Y}$. Les erreurs insérées sont alors appelées *dummy-errors*. Cette méthode permet de supprimer les motifs répétitifs de la redondance lors du codage. C'est ainsi que le chemin survivant est ramené sur le chemin TAZ.

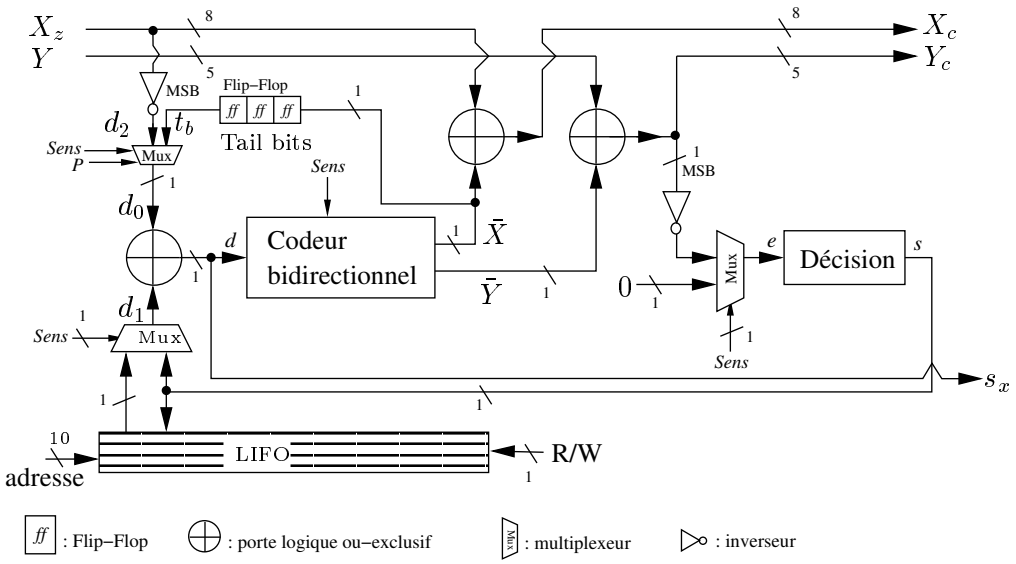


Figure 4.20 — Architecture pour le codage différentiel.

La figure 4.20 représente l'architecture du codage différentiel tenant compte de l'insertion d'erreurs. Le codage est réalisé à l'aide d'un codeur bidirectionnel. Il est alors alimenté par le signal d en formant un mot $\bar{V}(\bar{X}, \bar{Y})$ à la sortie du codeur. Deux portes sont nécessaires pour effectuer l'opération ou-exclusif entre $V(X_z, Y)$ et $\bar{V}(\bar{X}_z, \bar{Y})$. Naturellement, le MSB qui correspond à la décision binaire est retenu pour cette opération.

Nous choisissons de faire le décodage d'abord dans le sens *Aller*. L'entrée du codeur d est le résultat de l'opération ou-exclusif entre d_0 et d_1. Pour le sens *Aller*, la valeur inversée du MSB(X_z), notée d_2, est choisie pour le signal d_0. De plus, la valeur s est choisie pour le signal d_1. En effet, le signal s sortant du bloc *Décision* détecte la présence du motif répétitif. Si $s = 1$, une erreur est alors insérée. En parallèle, cette insertion est mémorisée dans une mémoire LIFO, qui sera lue lors du décodage dans le sens *Retour* pour repérer la position où une erreur a été insérée dans le sens *Aller*. La mémoire LIFO permet de sortir la valeur stockée la plus récente de la mémoire à l'aide d'une adresse entrante et d'un signal de contrôle R/W (R pour la lecture et W pour l'écriture). Par ailleurs, les *tail bits* sont stockés dans 3 bascules.

Pour le décodage dans le sens *Retour*, il est alors nécessaire de re-produire la même séquence d dans l'ordre inverse par rapport au décodage précédent. Pour ce faire, les *tail bits* stockés lors du décodage précédent sont tout d'abord utilisés dans l'ordre inverse. Cela permet de remonter le même chemin lors du parcours du treillis dans le sens *Retour*. C'est pourquoi l'insertion d'erreurs dans le sens *Retour* est remplacée par une simple lecture de la mémoire LIFO. Ceci implique que la détection du motif n'est pas modifiée lors du codage dans le sens *Retour*. L'entrée du bloc *Décision* est alors mise à zéro et la sortie s n'est cependant pas

utilisée.

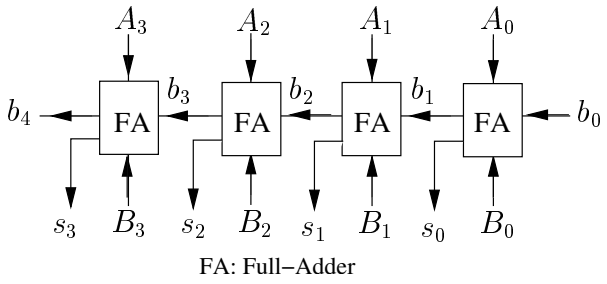


Figure 4.21 — Architecture d'un additionneur de 4 bits.

Dans notre implémentation, une valeur souple a une représentions en complément à 2. Au niveau de l'opération ou-exclusif entre une valeur souple quantifiée sur n bits et un bit b, les n bits sont d'abord inversés puis ajoutés à 1 lorsque $b = 1$. Cette opération exige donc un additionneur pour ajouter la retenue 1. Or, il est possible d'intégrer cette retenue lors du calcul des métriques de branche. La figure 4.21 présente l'architecture d'un additionneur de 4 bits. Elle est composée de 4 composants binaires FA (Full-Adder en anglais). Chaque composant FA produit une retenue b_1 et la somme s_0 en fonction des entrées A_0 et B_0 ainsi que la retenue b_0. Dans notre cas, il suffit alors de placer la retenue 1 à l'entrée b_0 de l'additionneur au moment où la métrique de branche est calculée. Ainsi, il n'est pas nécessaire d'ajouter un additionneur supplémentaire lors de l'opération ou-exclusif du décodage différentiel par rapport à l'architecture classique, d'où une diminution de complexité.

4.4.4 Architecture pour la détection du motif

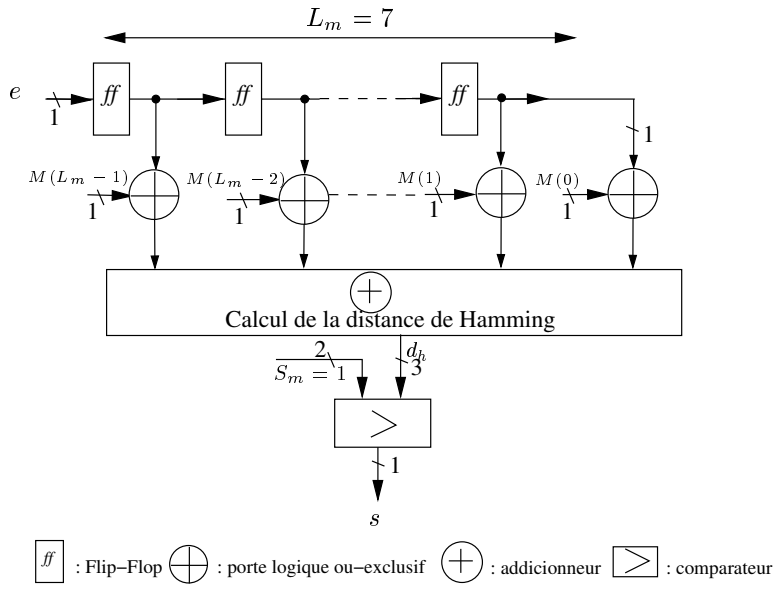


Figure 4.22 — Architecture du bloc *Décision*.

La détection du motif circulaire est utilisée pour insérer des dummy-errors. Puisque le motif est répétitif avec une période L_m, nous avons :

$$M_k^o = M_{k+L_m}^o \tag{4.12}$$

où le symbole M_k^o représente un élément du motif à l'instant k. Ce motif est apparu dans la redondance résultant de l'opération $V \oplus \bar{V}$ (voir l'expression 4.11). Dans ce cas, la détection du motif peut se faire dans une fenêtre glissante ayant la longueur L_m (figure 4.22). Nous prenons $L_m = 7$ et le seuil de détection $S_m = 2$ (voir Chapitre 2). La structure du motif comprend alors 7 bascules en série. La distance de Hamming entre les valeurs dans la fenêtre et celles du motif théorique, notées $(M_{L_m-1} \cdots M_0)$, est ensuite calculée et comparée au seuil S_m. Ceci est réalisé à l'aide de 7 portes logiques ou-exclusif et un arbre d'additionneur calculant la somme de 7 bits. Si la distance de Hamming d_h est inférieure au seuil S_m, le signal sortant s est activé ($s = 1$). Dans le cas contraire, il signal s est égal à 0. L'insertion d'erreurs à l'instant k dépend donc des données de l'instant $k-1$ à l'instant $k - L_m - 1$. Par conséquent, ce procédé implique que l'élimination du motif a lieu après une apparition de ce motif.

4.4.5 Architecture du décodeur SISO-D appliquant le calcul anticipé de l'information extrinsèque

Pour profiter de la propriété que le chemin survivant est souvent sur le chemin TAZ grâce à l'insertion de dummy-errors, il est possible de calculer l'information extrinsèque z lors du premier parcours du treillis, dit *calcul anticipé* de l'information extrinsèque. Si le chemin survivant est ramené sur le chemin TAZ pendant le premier parcours du treillis (sens *Aller*), l'information extrinsèque z peut être calculée à partir de la redondance Y_k et de la métrique de nœud α_{k-1}^1 sur l'état 1 d'après l'expression suivante (figure 4.23) :

$$z_a(k) = Y_k - \frac{1}{2}\alpha_{k-1}^1. \tag{4.13}$$

Dans ce cas, il n'est pas nécessaire de stocker toutes les métriques de nœud α dans le sens *Aller*, d'où une diminution du nombre d'accès mémoire et par conséquent de la consommation dynamique.

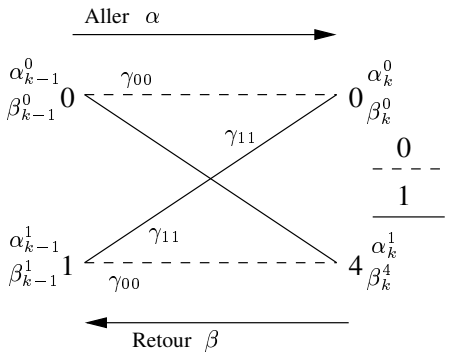


Figure 4.23 — Papillon du treillis associé au codeur ayant pour polynôme générateur $G[1, (1 + D^2 + D^3)/(1 + D + D^3)]$.

Le calcul anticipé dépend de l'estimation du chemin TAZ sur les séquences à l'entrée du décodeur SISO. Pour effectuer le calcul anticipé $z_a(k)$ à l'instant k, les conditions suivantes doivent être réunies :

1. la valeur entrante X_k est inférieure à 0 ;
2. la distance de Hamming entre les L_{dh} mots suivants (X,Y) et le chemin TAZ correspondant est inférieure au seuil préfixé S_{dh} ;
3. la métrique de nœud α_{k-1} à l'état 0 est égale à 0.

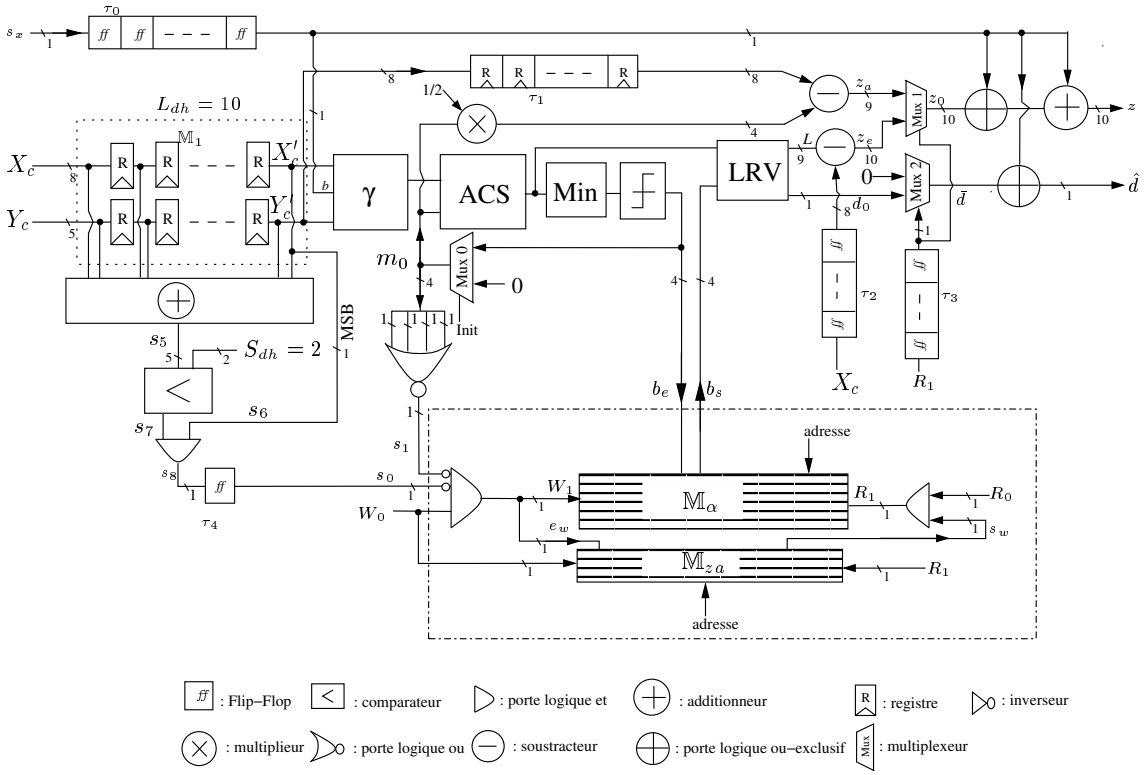


Figure 4.24 — Architecture d'un codeur SISO-D en tenant compte du calcul anticipé de l'information extrinsèque.

La figure 4.24 montre l'architecture globale d'un décodeur SISO-D (SISO Différentiel) en tenant compte du calcul anticipé de l'information extrinsèque z_a. Cette architecture est également basée sur les composants élémentaires : module γ, ACS, Min, et module LRV, comme pour le décodeur classique SISO. Tout d'abord, les données $(X_{c,k} \cdots X_{c,k-L_h d+1})$ sont stockées dans une mémoire, notée $\mathbb{M}_1$, ayant une longueur L_{dh}, soit 10 dans notre cas. En effet, cette mémoire $\mathbb{M}_1$ est composée de deux files de registres de 8 bits et de 5 bits respectivement pour les entrées X_c et Y_c.

La première tâche consiste à déterminer la distance de Hamming entre la séquence d'entrée et le chemin TAZ, c'est-à-dire, à calculer le nombre de 0s parmi les MSBs des registres pour le mot d'entrée (X_c, Y_c). Pour ce faire, Ces MSBs sont reliés à l'entrée d'un additionneur, dont le résultat s_5 est quantifié sur 5 bits ($2^5 > 20$). Ce résultat est alors comparé au seuil pré-fixé S_{dh} à l'aide d'un comparateur ($S_{dh} = 2$ dans notre cas). Si $s_5 < S_{hd}$, la sortie du comparateur s_7 est égale à 1. Pour vérifier si la valeur X'_c à l'entrée du module γ est inférieure à 0, il suffit d'observer le MSB du signal X'_c. Si ce MSB est égal à 1, alors, $X'_c < 0$ pour une représentation en complément à 2. Les signaux s_6 et s_7 sont reliés à l'entrée d'une porte logique *et* dont le signal sortant est noté s_8. La synchronisation nécessite l'ajout d'un délai τ_4 au signal s_8. Ce délai est équivalent au temps nécessaire pour calculer la métrique de branche à partir des valeurs X'_c et Y'_c, c'est-à-dire, τ_4 est égal à 1 cycle (voir la figure 4.10).

La métrique de nœud α_k doit être égale à 0 pour effectuer le calcul anticipé. La vérification n'est pas complexe à réaliser. Il suffit alors d'observer, à l'aide d'une porte logique *ou*, les bits de la valeur m_0 à l'entrée du module ACS. Si tous les bits sont égaux à 0, le signal s_1 est activé ($s_1 = 1$), sinon, il est désactivé ($s_1 = 0$). En résumé, les signaux s_0, s_1 et W_0 contrôlent l'écriture de la mémoire $\mathbb{M}_\alpha$. Le signal W_0 est équivalent au signal R/W du contrôleur décrit

dans la section 4.3.2. Le contrôle $W_0 = 1$ correspond au cas où $R/W = W$. Dans ce cas, l'écriture de la mémoire $\mathbb{M}_\alpha$ est contrôlée par le signal W_1. L'écriture en mémoire se produit si $s_0 = 0$, $s_1 = 0$ et $W_0 = 1$. De plus, l'activation de l'écriture est stockée dans une mémoire dédiée, marquée $\mathbb{M}_{za}$. Quant à la mémoire $\mathbb{M}_{za}$, son contenu s_w détermine la lecture de la mémoire $\mathbb{M}_\alpha$ avec le signal R_0 lors du décodage dans le sens *Retour*. Si $s_w = 1$ et $R_0 = 1$, alors $R_1 = 1$. La valeur b_s est lue de la mémoire $\mathbb{M}_\alpha$ pour calculer le LRV, tout comme le cas classique de l'algorithme MAX-Log-MAP.

Lorsque le calcul de l'information extrinsèque z_a est effectué au cours du premier parcours du treillis, il est alors nécessaire de stocker les valeurs z_a pour attendre le second parcours. En effet, la mise à jour de l'information extrinsèque z ne peut se faire qu'au cours du second parcours. Cela demande donc une mémorisation supplémentaire, qui est contraignant au niveau consommation. Pour contourner ce problème, il est possible de calculer l'information extrinsèque anticipée z_a dans le sens *Retour*, c'est-à-dire, à partir de la métrique de nœud β de la manière suivante :

$$z_a(k) = Y_k - \frac{1}{2}\beta^4_{k-1}. \tag{4.14}$$

Dans ce cas, nous n'avons pas besoin de mémoriser les valeurs z_a obtenues à partir de la métrique de nœud α (figure 4.23).

Le calcul anticipé, quant à lui, est réalisé par soustraction de la valeur Y_c retardée d'un temps τ_1, soit 5 cycles. Ce délai τ_1 correspond au temps nécessaire à synchroniser les valeurs Y_k et β_{k-1}. En effet, la valeur β_{k-1} est représentée par m_0 de la figure 4.24. Un facteur 1/2 est appliqué à la valeur m_0 (*c.f.* l'expression 4.13) avant de calculer l'information extrinsèque anticipée z_a. Par ailleurs, l'information extrinsèque z_e calculée selon l'algorithme Max-Log-MAP est obtenue grâce à la soustraction entre le résultat L du module LRV et la valeur entrante X_c retardée d'un temps τ_2, soit 15 cycles. De même, ce dernier correspond au temps nécessaire pour produire le LRV correspondant, y compris le temps de propagation dans la mémoire $\mathbb{M}_1$. Grâce à la mise en place d'un multiplexeur (Mux 1 de la figure 4.24), le résultat z_a est sélectionné si la lecture de mémoire $\mathbb{M}_\alpha$ n'est pas produite lors du parcours du treillis dans le sens *Retour*. Dans le cas contraire, la valeur z_e est sectionnée pour la sortie z.

Pour reconstituer la décision finale $\hat{d}$, il faut effectuer une opération ou-exclusif entre le signal d'entrée s_x retardé d'un temps τ_0, soit 15 cycles, et le signal d_1. La même procédure est appliquée à l'information extrinsèque z_0. Lorsque la valeur du signal s_x retardé est égale à 1, tous les bits de la valeur $\bar{z}$ sont inversés et une retenue 1 est ajoutée au résultat. Ceci évite de modifier l'amplitude de la valeur ayant une représentation complément à 2. Dans le cas contraire, la sortie de l'information extrinsèque z prend la valeur z_0. L'architecture précédente contient une mémoire supplémentaire $\mathbb{M}_{za}$ de taille 864×1 bits pour stocker l'évènement d'écriture de la mémoire $\mathbb{M}_\alpha$. Dans ce cas, toutes les valeurs de la métrique de nœud α ne sont pas écrites dans la mémoire $\mathbb{M}_\alpha$.

4.5 Conclusion

Nous avons présenté des solutions architecturales pour un système de turbo-décodage différentiel dédié à la norme UMTS. Ces solutions architecturales sont directement issues des réflexions au niveau algorithmique préalablement détaillées dans ce manuscrit. Nous avons aussi présenté un système de décodage intégrant toutes les propositions validées au cours de cette thèse. Le système final a les caractéristiques suivantes. A la réception, il est d'abord nécessaire de sauvegarder les données venant de la transmission du canal. Différentes mémoires

sont alors mises en place. De plus, le traitement et la réception des données sont effectués en parallèle. C'est pourquoi, nous avons présenté l'organisation de l'écriture et de la lecture des mémoires grandissant un traitement en temps réel.

Ensuite, nous avons présenté l'architecture globale d'un turbo-décodeur. Elle est composée d'un contrôleur, d'un décodeur élémentaire SISO et des mémoires de données $\mathbb{M}_X$, $\mathbb{M}_{Y_1}$, $\mathbb{M}_{Y_2}$ ainsi que d'une mémoire $\mathbb{M}_z$ pour l'information extrinsèque. Le contrôleur permet de piloter l'itération du décodage en cours et de valider les résultats du décodage. Il interagit donc avec le décodeur SISO et les mémoires.

L'architecture d'un décodeur élémentaire SISO est ensuite détaillée. Cette architecture détermine la complexité matérielle et donc la consommation du système. Dans le cadre de notre étude, la normalisation dynamique des métriques de nœud est réalisée au sein de ce décodeur SISO. A partir de l'architecture classique d'un décodeur SISO, nous avons réalisé l'architecture d'un codage différentiel en insérant un codeur convolutif bidirectionnel à la réception. Pour ce faire, le codeur convolutif bidirectionnel, qui permet de parcourir le même chemin de codage lors des deux sens (*Aller* et *Retour*) de parcours du treillis, est construit à partir d'un codeur classique. Quant à l'insertion d'erreurs, elle est obtenue à l'aide d'un registre à décalage pour repérer la présence d'un motif d'erreurs. De plus, l'insertion d'erreurs est d'abord mémorisée dans le sens *Aller* et ensuite lue dans le sens *Retour*. Cette solution permet de suivre le même chemin lors du codage différentiel dans les deux sens. En outre, le traitement de *tail bits* est également présenté au niveau architectural. Pour le calcul anticipé de l'information extrinsèque, une solution consiste à sauvegarder l'évènement de l'écriture de la mémoire $\mathbb{M}_\alpha$ dans une mémoire supplémentaire de taille 864 bits. Dans ce cas, toutes les métriques de nœud α ne sont pas écrites dans la mémoire $\mathbb{M}_\alpha$.

Dans le chapitre suivant, les architectures proposées sont implémentées sur une carte XILINX Virtex-II Pro. Ainsi, nous analysons les différents résultats de synthèse de chaque architecture et faisons une comparaison sur des consommations respectives.

CHAPITRE

5 Prototypage et Impact sur la Consommation

DANS ce dernier chapitre, nous présentons l'implémentation d'une chaîne de communications numériques ainsi que l'expérimentation de mesures sur la consommation du circuit. Tout d'abord, l'environnent d'implémentation et de mesures est introduit. Afin de limiter les biais de mesure, nous avons mis en œuvre une technique de *reconfiguration dynamique* (section 5.1.1). Ensuite, l'architecture de la partie émettrice est présentée. Elle est constituée d'un générateur de données pseudo-aléatoires afin de générer les mots de code émis, d'un encodeur et d'un émulateur de canal (section 5.2).

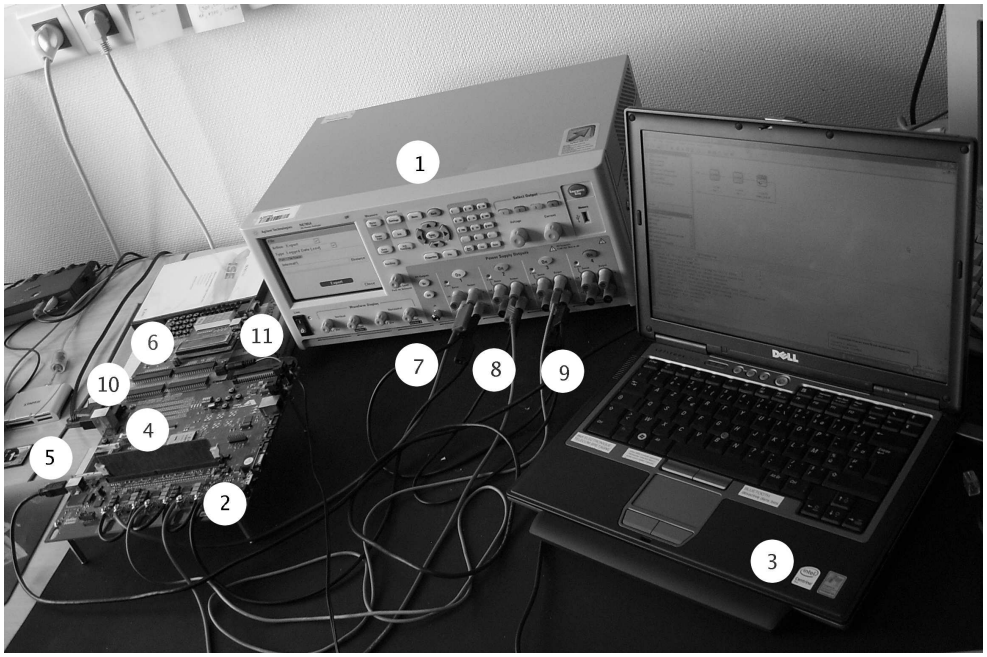
La section 5.3 est consacrée aux synthèses des différentes architectures proposées dans le chapitre précédent : TD (Turbo-Décodeur classique) TD-SM (Turbo-Décodeur appliquant la Saturation des Métriques de nœud), TD-CD (Turbo-Décodeur avec l'application du codage différentiel), TD-IE (Turbo-Décodeur appliquant l'Insertion d'Erreurs) et TD-CA (Turbo-Décodeur utilisant la méthode *Calcul Anticipé*). Les synthèses respectives sont détaillées. Nous faisons ensuite un bilan des différentes solutions architecturales en terme de complexité, de performance mesurée, et de mesure sur la consommation réelle des architectures.

5.1 Environnent d'implémentation et de prototypage

Dans cette section, nous décrivons l'environnement d'implémentation et de mesure de consommation. Puis, nous introduisons la technique *reconfiguration dynamique* d'un circuit FPGA. Enfin, nous détaillons la procédure de mesure de consommation.

5.1.1 Environnent d'implémentation et de mesure de consommation

Pour connaître la consommation réelle d'une application, il est nécessaire de l'implémenter sur un circuit électronique de type ASIC (Application Specific Integrated Circuit) et de faire des mesures. Or, l'implémentation sur un circuit ASIC est très coûteuse au niveau du temps de conception et la fabrication du circuit ASIC est très complexe. Dans un premier temps, nous avons choisi d'implémenter l'application sur un circuit FPGA, qui permet d'obtenir un résultat rapide sur la consommation réelle du circuit. Pour ce faire, nous avons implémenté une chaîne de communications numériques, et en particulier un turbo-décodeur à 8 états issu de la norme UMTS, sur un circuit de type FPGA Virtex II Pro de Xilinx.



1: Agilent DC power analyzer N6705A; 2: carte électronique XUP; 3: PC, ISE, XST, IMPACT, Planahead; 4: FPGA VirtexII Pro
5: Bus USB; 6: Extension DIO5; 7: Alim0=1.5V; 8: Alim1=2.5V; 9: Alim2=3.3V; 10: Connecteur; 11: Alim3=9.5V;

Figure 5.1 — Environnent d'implémentation et de mesure de consommation.

La figure 5.1 montre l'environnement d'implémentation et de mesure de consommation. Dans notre étude, nous utilisons la plate-forme XUP (abrégée de Xilinx University Programme en anglais) (voir le numéro 2 de la figure 5.1), dans laquelle un circuit FPGA (Virtex II Pro) (voir le numéro 4) est disponible. Nous allons uniquement introduire les principaux blocs de fonctionnement de la plate-forme utilisée. Plus d'informations sont disponibles dans le document technique de la carte [Xilinx 08].

L'horloge de la plate-forme, gérée par un bloc spécifique, peut s'adapter aux différents composants de la plate-forme. L'horloge du circuit FPGA, quant à elle, est gérée par une entité DCM (Digital Clock Manager en anglais). Trois tensions, *alim0*=1.5V, *alim1*=2.5V et

alim2=3.3V, (voir les numéros 7, 8 et 9) fournies par l'analyseur de consommation Agilent (voir le numéro 1), alimentent respectivement le cœur du circuit FPGA, ses périphériques et les périphériques de la carte. Les périphériques du circuit FPGA (mémoire, microprocesseur, etc) et ceux de la plate-forme (bus USB, porte PS/2, porte série RS-232, etc) (voir le numéro 5) sont respectivement gérés par deux blocs dédiés. Cette structure assure de bonnes communications entre le circuit et les périphériques. Dans le cadre de la thèse, nous nous focalisons sur la consommation du turbo-décodeur, qui dépend de la consommation du cœur du circuit FPGA.

Une carte périphérique DIO5 (voir le numéro 6) est connecté à la plate-forme à l'aide d'un connecteur (voir le numéro 10). Elle dispose de sa propre alimentation *alim3*=9.5V (voir le numéro 11). Comme les performances du turbo-décodeur dépendent du rapport signal à bruit, il est donc traité comme un paramètre d'entrée de l'architecture dans nos implémentations. Ce rapport dont la valeur est fixée par l'utilisateur est obtenu à l'aide des boutons disponibles sur la carte périphérique DIO5. L'afficheur LCD de cette carte permet de visualiser en temps réel le rapport signal à bruit et le taux d'erreurs binaires mesuré correspondant.

La carte périphérique DIO5 envoie d'abord le rapport signal à bruit au système implémenté sur le circuit FPGA. Le système récupère donc une valeur fixée, par exemple 1.5dB. En fonction de cette valeur, l'émulateur de canal ajoute un bruit ABBG correspondant aux messages générés par l'entité *émetteur*. Dans notre étude, nous avons considéré un mapping de type BPSK (Binary Phase Shift Keying en anglais). Des conversions binaire-symbole et symbole-binaire sont donc présentes avant et après l'émulateur de canal. Puis, le turbo-décodeur fournit le résultat de décodage à partir des valeurs bruitées. Enfin, le taux d'erreurs binaires calculé à partir du résultat de décodage est affiché en temps réel sur l'écran LCD. De même, le TEB est transmis par le connecteur.

Concernant la mise en œuvre du système, elle est réalisée selon le cahier de charge à l'aide de l'outil de CAOs (Conception Assistée par Ordinateur) de Xilinx (ISE, XST etc). Pour implémenter une application sur le circuit FPGA, le fichier bitstream, généré à l'issue du flot de conception, est chargé dans le circuit FPGA via une interface USB à partir d'un PC (voir le numéro 3 de la figure 5.1).

5.1.2 Reconfiguration dynamique d'un circuit FPGA

Notre objectif est d'étudier l'impact sur la consommation des différentes solutions architecturales. Comme nous avons plusieurs architectures, un système complet a été intégré sur le circuit FPGA pour chaque architecture de turbo-décodeur. Nous disposons donc de plusieurs fichiers bitstream et chaque fichier correspond à la chaîne complète de communications numériques dans laquelle une architecture de turbo-décodeur est considérée. Cette solution a un inconvénient majeur. Elle ne permet pas de différentier le coût de chacune des fonctions constituant la chaîne de communications. Dans ce cas, les résultats de mesure de consommation ne sont pas fiables.

Pour palier ce problème, une solution consiste à utiliser une technique de *reconfiguration dynamique*, qui permet d'implémenter un système sur deux parties : une partie statique et une partie dynamique. Les deux parties sont configurées l'une après l'autre par des fichiers bitstreams distincts. Il est alors possible de reconfigurer partiellement la partie dynamique de telle manière à adapter l'architecture considérée. Dans notre cas, la partie dynamique concerne le turbo-décodeur et la partie statique correspond aux autres composants de la chaîne de communications. Comme seule la partie turbo-décodeur est reconfigurée en fonction de l'architecture retenue pour chaque prototype, l'environnent de test pour le turbo-décodeur

devient identique. La reconfiguration dynamique est ainsi une technique efficace et exploitable pour mesurer, outre de consommation, une partie déterminée d'un système implémenté sur FPGA.

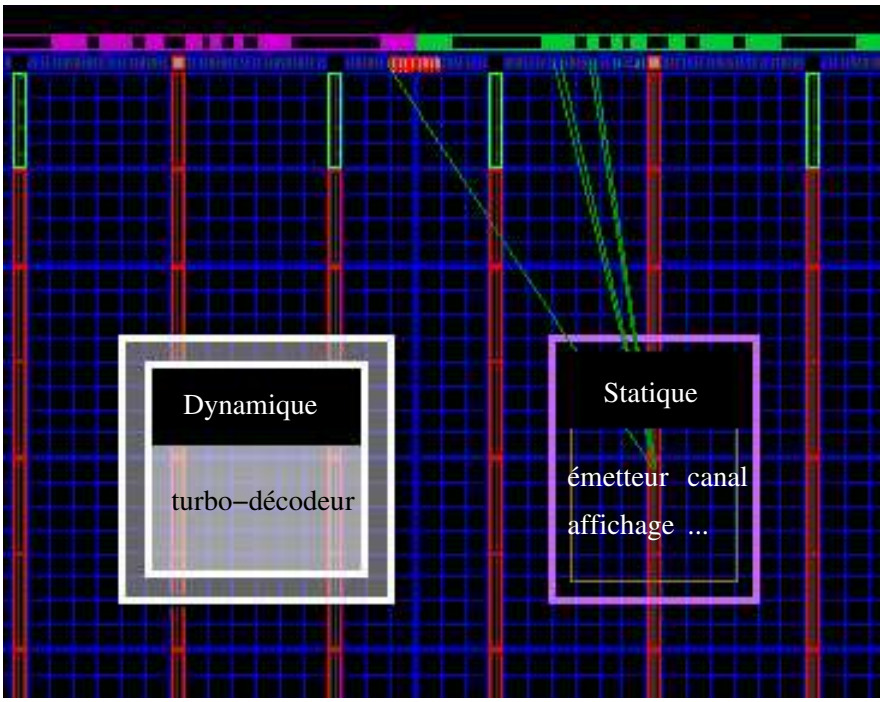


Figure 5.2 — Exemple de layout contenant une reconfiguration dynamique.

La figure 5.2 donne un exemple de layout. La zone d'implémentation du circuit FPGA est découpée en deux portions : une portion dynamique et une portion statique. La portion dynamique est reconfigurable selon l'architecture sélectionnée tandis que la portion statique n'est configurée qu'une seule fois. Dans notre cas, il s'agit d'implémenter le turbo-décodeur sur la portion dynamique. Le reste du système, *i.e*, l'émetteur, le mapping/demapping, le canal, et le calcul du TEB est implémenté dans la zone statique. Les deux zones communiquent à travers des bus dédiés. L'avantage majeur de cette technologie réside dans le fait que la portion statique conserve toujours le même placement-routage pour tous les prototypes, ce qui permet la comparaison des consommations de chaque prototype.

Pour exploiter aisément la technologie reconfiguration dynamique, Xilinx a proposé un outil appelé *planahead* dédié à cette tâche. Par ailleurs, nous disposons de 5 solutions architecturales dédiées aux approches détaillées dans le chapitre 4.

5.1.3 Description de la procédure de mesure

Pour évaluer la puissance consommée par le cœur du circuit FPGA, nous mesurons la valeur de courant fourni par la tension $Alim0 = 1.5\text{V}$. La puissance consommée est alors obtenue par le produit du courant I et de la tension $U = 1.5\text{V}$. Nous avons :

$$P = I \cdot U(\text{W}) \tag{5.1}$$

Lorsque le système implémenté s'exécute, l'analyseur de consommation Agilent (figure 5.1) fait un échantillonnage de courant et de tension auxquels nous nous intéressons, à savoir que

la fréquence d'échantillonnage est égale à 50KHz. Il fournit une valeur moyenne à chaque milliseconde, soit sur 50 échantillons. Le temps d'une mesure est fixé à 30s. Nous obtenons donc 30,000 données de mesure de courant et de tension sur le numéro 7 de la figure 5.1.

La procédure d'expérimentation comprend plusieurs étapes :

1. Implémenter respectivement les parties statique et dynamique à l'aide de l'outil XST. Nous obtenons les fichiers de configuration pour chaque décodeur SISO et l'ensemble du système.
2. Charger les fichiers respectifs de configuration sur les deux zones de configuration du circuit FPGA à l'aide de l'outil IMPACT.
3. Exécuter le système en permanence.
4. Saisir le rapport signal à bruit souhaité. Rappelons que le rapport SNR est variable.
5. Vérifier le bon fonctionnent en relevant la courbe de performance en terme de TEB en fonction du rapport signal à bruit. Le TEB peut être visualisé sur l'écran LCD.
6. Faire des mesures de courant au point 7 de la figure 5.1 à l'aide de l'analyseur Agilent pour les différents SNRs. Dans notre cas, 30,000 données sont relevées pour un turbo-décodeur et un SNR donné.
7. Calculer la valeur moyenne de la puissance P_{moy} à partir de la relation suivante :

$$P_{moy} = \frac{1}{30,000} \sum_{i=1}^{30,000} I_i \cdot U_i \tag{5.2}$$

   où I_i et U_i représentent respectivement des valeurs mesurées du courant et de la tension.
8. Charger un autre fichier bitstream correspondant à un autre turbo-décodeur pour reconfigurer partiellement la partie dynamique.
9. Répéter les étapes 4 à 8 jusqu'à la fin de mesure. Nous vérifions que la partie statique n'est pas modifiée au cours des mesures.

5.2 Implémentation d'un système de transmissions numériques

Dans cette section, nous présentons d'abord l'architecture d'un générateur de données pseudo-aléatoires. Ensuite, la partie émettrice de la chaîne de transmissions est détaillée au niveau architectural. Nous donnons également l'architecture d'un codeur convolutif à 8 états correspondant à la norme UMTS. Puis, un canal de transmission de type ABBG est implémenté. Enfin, les résultats respectifs de synthèse sont donnés à la fin de cette section.

5.2.1 Générateur des données binaires

Le générateur de données binaire est réalisé à l'aide d'un registre à décalage, dit LFSR (Linear Feedback Shift Register). La figure 5.3 présente la structure du LFSR que nous avons implémentée. Il s'agit d'un registre à décalage de 23 bits. La récursivité du registre est définie à l'aide du polynôme générateur suivant :

$$x^0 = x^{23} + x^{18}. \tag{5.3}$$

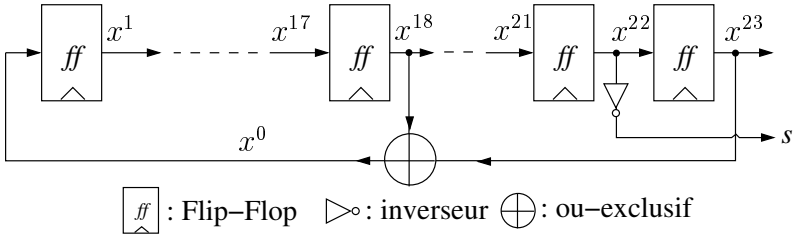


Figure 5.3 — LSR

En effet, les signaux $(x^1, x^2, \cdots, x^{23})$ du registre donne une valeur en représentation binaire. La valeur du registre a une périodicité qui dépend du polynôme générateur. C'est pourquoi le LFSR ne génère pas une séquence de données aléatoires. Si cette période est suffisamment grande, nous considérons que les données générées sont aléatoires. La structure de la figure 5.3 permet de générer une séquence de données avec une période de $2^{23} - 1$ selon [Alfke 96]. Concernant la sortie binaire, nous choisissons la valeur inverse du signal x^{22}.

5.2.2 Architecture de la partie émettrice (codeur)

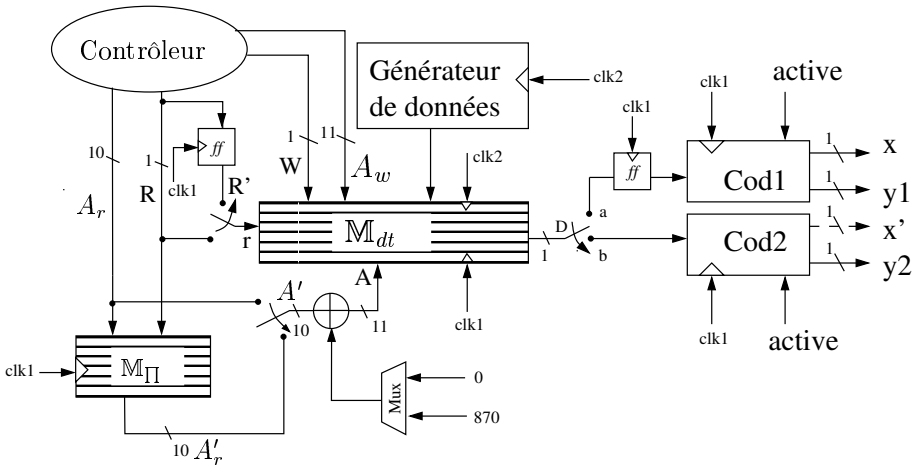


Figure 5.4 — Architecture de la partie émettrice.

Dans une chaîne de communications numériques, les messages numériques, aléatoires, indépendants, sont transmis de manière séquentielle. La figure 5.4 présente l'architecture globale de la partie aléatoire. Les données numériques sont produites de façon pseudo-aléatoire et indépendante par le générateur de séquences pseudo-aléatoires définies à la section précédente. L'horloge associée à la fréquence de génération est notée clk_2. Les données générées sont ensuite stockées dans une mémoire dédiée, notée $\mathbb{M}_{dt}$. Tout comme la gestion de mémoires décrite dans la section 4.2, le codage de données et la génération de données se produisent en parallèle. Nous avons donc besoin d'une mémoire dont la longueur de la mémoire est deux fois plus grande que celle de la trame générée. Ainsi, la taille de la mémoire $\mathbb{M}_{dt}$ est égale à $(2 \times 864) \times 1$ bits. Cette mémoire dispose de deux accès, dédiés à la lecture et l'écriture de données, qui sont cadencées par deux horloges distinctes clk_1 et clk_2. Les cycles de lecture et d'écriture sont similaires à ceux présentés dans le chapitre 4 (voir la section 4.2). Les opérations de lecture et d'écriture sont effectuées en parallèle. Quant à l'entrelaceur associé au codeur, il est équivalent à celui du turbo-décodeur, c'est-à-dire qu'il est remplacé par une

mémoire ROM, notée $\mathbb{M}_{\prod}$, dont la taille est égale à (870×10) bits. La lecture de la mémoire $\mathbb{M}_{\prod}$ prend le délai d'un cycle d'horloge.

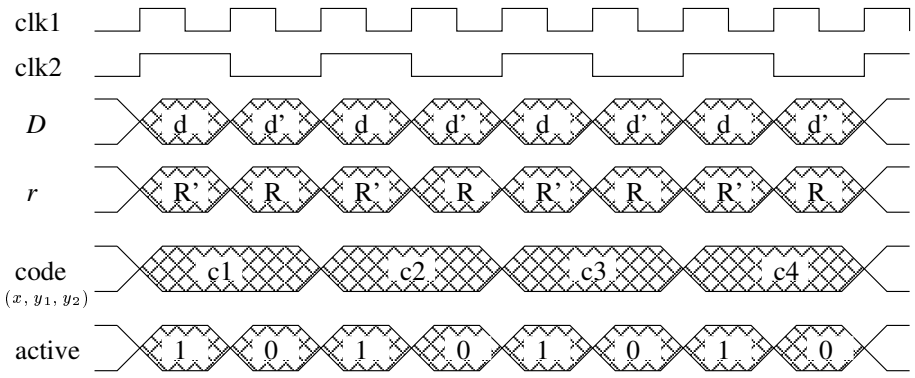


Figure 5.5 — Chronogramme de la génération du mot de code dans la partie émettrice.

Les signaux de contrôle sont délivrés par un contrôleur (*c.f.*, la figure 5.4). Les lectures de la mémoire $\mathbb{M}_{\prod}$ et de la mémoire $\mathbb{M}_{dt}$ sont respectivement contrôlées par les signaux R et r. De plus, les données sont lues à l'aide des adresses respectives A_r et A. Le signal r reçoit alternativement le signal R ou le signal R' un cycle de l'horloge clk_1 sur 2. Une bascule permet de synchroniser le signal R' et son adresse A'_r dans l'ordre entrelacé. Un chronogramme détaillant ce séquencement est donné dans la figure 5.5. Le contrôleur ainsi que l'entrelaceur délivre des adresses allant de 0 à 870. Or, la deuxième moitié de la mémoire $\mathbb{M}_{dt}$ est associée à des adresses allant de 870 à 1739. C'est pourquoi une constante 870 est ajoutée à l'adresse A' (*c.f.* la figure 5.4) pour accéder à cette partie. Ainsi, le bus de données D contient alternativement les valeurs d (ordre naturel) et d' (ordre entrelacé) de la mémoire $\mathbb{M}_{dt}$.

Pour produire un mot de code avec un rendement 1/3, deux codeurs convolutifs élémentaire, notés *Cod 1* et *Cod 2*, sont concaténés en parallèle. L'architecture d'un tel codeur sera détaillée dans la section suivante. Une bascule à l'entrée du codeur 1 permet de synchroniser les entrées des codeurs. Les valeurs d et d' sont respectivement codées par les codeurs 1 et 2. Nous obtenons alors un mot de code (x, y_1, y_2) à l'issue du codage. Ainsi, les 3-uplets, notés $c1,\ c2, \cdots$ (*c.f.* la figure 5.5), sont produits tous les deux cycles de l'horloge clk_1. Dans ce cas, le générateur ne produit pas des données pendant 6 cycles de l'horloge clk_2 à la fin du codage d'une trame. Dans notre cas, les fréquences clk_1 et clk_2 sont respectivement égales à 1MHz et 2MHz.

5.2.3 Codeur convolutif à 8 états

Le turbo-codage se compose de deux codeurs identiques. Nous détaillons donc l'architecture du codeur retenu. La figure 5.6 présente l'architecture d'un CCRS (Code Convolutif Récursif Systématique) ayant pour polynôme générateur $G[1, (1 + D + D^3)/(1 + D^2 + D^3))]$ dans l'émetteur. Trois bascules de type Flip-Flop sont en cascade, ce qui correspond à 8 états. Le signal d'activation, noté *active*, est égal à 1 tous les deux cycles de l'horloge clk_1. Le chronogramme correspondant est tracé sur la figure 5.5, qui montre que la valeur du signal *active* bascule alternativement entre 1 et 0 au cours d'un cycle de l'horloge clk_2. Lorsque le signal $active = 0$, l'horloge $clk1$ (figure 5.6) n'est pas transmise aux bascules du codeur. Dans ce cas, le codeur est dans l'état *arrêt*, c'est-à-dire, le codeur reste dans son état courant. Dans le cas contraire, l'entrée d est codée et l'état du codeur évolue en fonction de son état courant

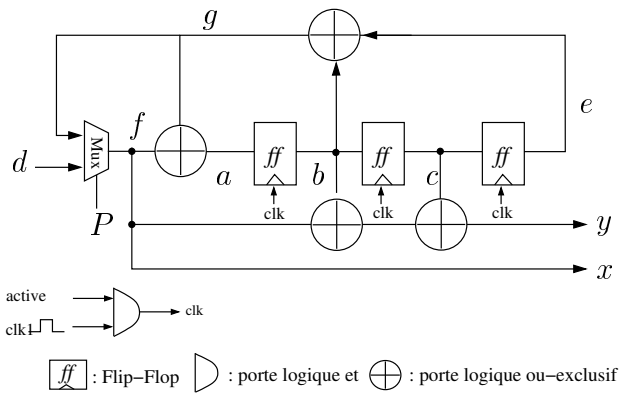


Figure 5.6 — Architecture d'un CCRS ayant pour polynôme générateur $G[1, (1 + D + D^3)/(1 + D^2 + D^3))]$.

et de cette entrée (figure 5.6). Un compteur est mis en place pour relever le nombre de bits systématiques. Si le compteur atteint la longueur de la trame, *i.e.*, 864 dans notre étude, le signal $P = 1$. Cela signifie que le codage de la trame est terminé et le codeur génère les *tail bits* à partir de l'état final du codage. Dans ce cas, le signal f reçoit le signal g. Dans le cas contraire, soit $P = 0$, le signal f reçoit le signal d, ce qui correspond au codage de l'entrée d.

5.2.4 Synthèse de la partie émettrice (encodeur)

Pour estimer la complexité matérielle d'une architecture donnée, nous nous intéressons en particulier aux ressources nécessaires, à savoir : Flip-Flop, LUT (Lookup Table) et BRAM (Bloc RAM). Nous donnons alors les nombres respectifs des ressources nécessaires à l'aide de l'outil de synthèse XST. Le tableau 5.1 montre le résultat de synthèse pour la partie émettrice implémentée. La fréquence maximale estimée f_{max} est également donnée. Cette fréquence peut néanmoins être variable selon le fichier de contrainte défini pas l'utilisateur lors de l'implémentation. Ainsi, l'architecture de la partie émettrice nécessite 132 Flip-Flops, 277 LUTs et 2 BRAMs. La fréquence maximale est égale à 174MHz. En pratique, la fréquence pour le générateur de données est mise à 1MHz et le codeur convolutif travaille avec une fréquence de 2MHz.

Tableau 5.1 — Synthèse de la partie émettrice.

synthèse	Flip-Flop	LUT	BRAM	f_{max} (MHz)
nombre	132	277	2	174

5.2.5 Implémentation de l'émulateur de canal ABBG

Après codage d'une trame, les symboles codés sont transmis à travers un canal de transmission. Il est donc nécessaire d'implémenter l'émulateur du canal retenu sur le circuit FPGA. La figure 5.7 présente l'interface de l'émulateur de canal ABBG implémenté. La variance du bruit σ_b est pré-calculée selon la puissance souhaitée, c'est-à-dire, le rapport signal à bruit, et stockée dans une mémoire, notée $\mathbb{M}_\sigma$. Sa taille est égale à $2^6 \times 17$ bits. Chaque mot de la mémoire est codé sur 17 bits dont 16 bits dédiés à sa partie fractionnaire. L'adressage de la mémoire A_σ nécessite 6 bits, soit 2^6 adresses disponibles. L'adresse 0 est réservée à un

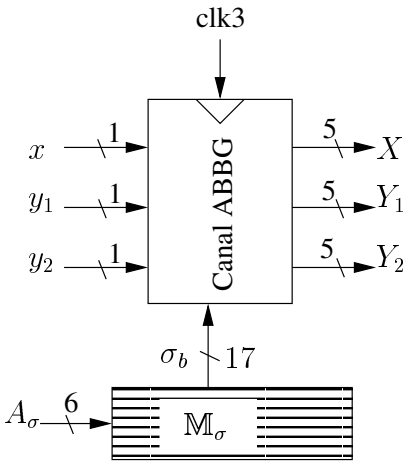


Figure 5.7 — Interface d'un canal ABBG

SNR=$+infty$ et le reste correspond à des SNRs allant de 0.25dB à 15.75dB avec un pas de 0.25dB. Les valeurs bruitées X, Y_1 et Y_2, quantifiées sur 5 bits, sont respectivement obtenues à partir des entrées x, y_1 et y_2, issues de la partie émettrice. Au cours de la transmission, un émulateur de canal ajoute un bruit ABBG aux messages codés. L'émulateur du canal gaussien que nous utilisons est détaillé dans [Boutillon 03a]. Dans cette section, nous présentons uniquement le principe de cette implémentation et les résultats de la synthèse associée.

Considérons le bruit comme une variable aléatoire V caractérisée par sa variance σ et une espérance nulle. Nous obtenons alors une distribution gaussienne $\mathcal{N}(0, \sigma)$ de la variable V. En simulation, la méthode Box-Muller [Knuth 98] est généralement utilisée pour générer une variable aléatoire gaussienne ayant une distribution normale $\mathcal{N}(0, 1)$. Cette méthode se compose de deux étapes. La première étape consiste à générer deux variables indépendantes v_1 et v_2, dont les valeurs sont comprises dans l'intervalle $[0, 1]$, à partir des expressions suivantes :

$$f(v_1) = \sqrt{-\ln v_1} \tag{5.4}$$

et

$$g(v_2) = \sqrt{2}\cos(2\pi v_2) \tag{5.5}$$

Le produit (*c.f.* l'expression 5.6) génère un échantillon v de la variable aléatoire V ayant une distribution normale $\mathcal{N}(0, 1)$:

$$v = f(v_1)g(v_2) \tag{5.6}$$

Il est difficile d'implémenter les fonctions $\ln(v_1)$ et $\cos(2\pi v_2)$ des expressions 5.4 et 5.5 à l'aide d'un processeur flottant de 32 bits. De plus, il est encore plus complexe de mettre en œuvre ces fonctions pour une cible matérielle tel qu'un circuit FPGA. C'est pourquoi une technique consiste à pré-calculer puis à mémoriser les valeurs $f(v_1)$ et $g(v_2)$ à partir des valeurs connues v_1 et v_2.

La figure 5.8 présente une architecture dédiée à la réalisation d'une variable aléatoire V ayant une distribution normale $\mathcal{N}(0, 1)$. Cette architecture repose à la fois sur la méthode Box-Muller [Knuth 98] et sur le théorème de la limite centrale [Tijms 04]. A partir des LFSRs de 29 bits, une adresse de 20 bits et une adresse de 8 bits sont respectivement fournies aux mémoires ROMs, dans lesquelles les valeurs $f(v_1)$ et $g(v_2)$ sont stockées. Le dernier bit, noté *Signe*, indique le signe d'échantillon généré. Nous donnons une notion (m, n) indiquant qu'une valeur est quantifiée sur m bits dont n bits dédiés à la partie fractionnaire. Les valeurs $f(v_1)$

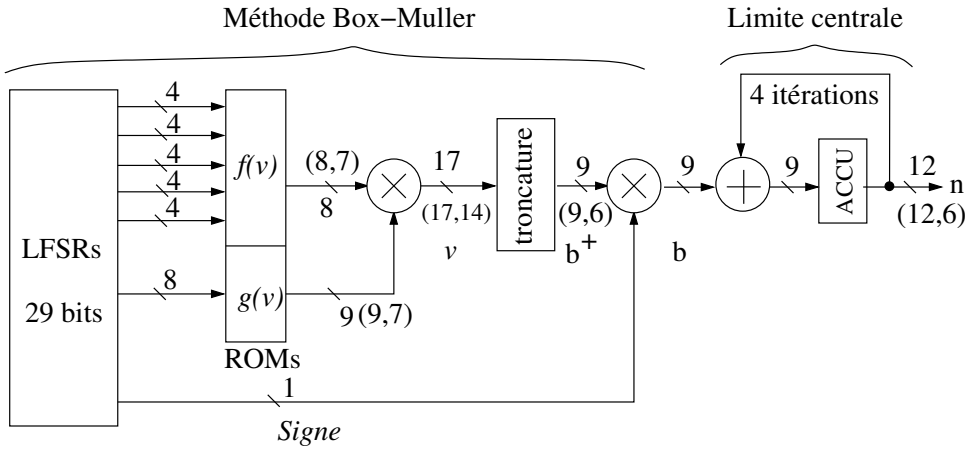


Figure 5.8 — Mise en œuvre d'une variable aléatoire ayant une distribution normale $\mathcal{N}(0,1)$.

et $g(v_2)$ sont respectivement quantifiées sur 8 bits et 9 bits sous des formats $(8,7)$ et $(9,7)$. La valeur x quantifiée sur $(17,14)$ bits est ensuite tronquée à une valeur sur $(9,6)$ bits. Ainsi, nous obtenons un échantillon b à partir de la relation suivante :

$$b = (1 - 2 \cdot Signe) \times b^{+}, \tag{5.7}$$

où b^{+} est une valeur positive.

La multiplication par le signal $Signe$ permet d'obtenir une densité de probabilité centrée sur 0 en utilisant une représentation en complément à 2. Dans ce cas, la probabilité sur la valeur 0 est deux fois plus grande qu'une distribution normale, car le complément à 2 de la valeur 0 est égal aussi à 0. Pour résoudre cette contrainte, le théorème de la limite centrale est appliqué.

Si V est une variable aléatoire caractérisée par son espérance m_v et sa variance σ_v, une variable aléatoire V_N peut être définie à partir de l'expression suivante :

$$V_N = \frac{1}{\sigma_v \sqrt{N}} \sum_{i=0}^{N-1} (v_i - m_v) \tag{5.8}$$

où v_i avec $i = 0 \cdots N-1$ sont N réalisations de la variable aléatoire V. Le théorème de la limite centrale dit que si N tend vers infini, la variable aléatoire suit une distribution normale $\mathcal{N}(0,1)$. Dans notre cas, il s'agit d'accumuler les échantillons b sur 4 itérations pour obtenir une réalisation n (voir la figure 5.8).

Pour obtenir un canal ABBG, il est ensuite nécessaire de multiplier la variance du bruit σ_b (figure 5.7) selon la puissance souhaitée avec la variable aléatoire ayant une distribution normale $\mathcal{N}(0,1)$. Puis, le bruit est ajouté aux messages codés. La donnée résultante est ensuite tronquée avant d'être traitée par le décodeur élémentaire SISO. Dans notre cas, les éléments bruités X, Y_1 et Y_2 sont saturés et quantifiés sur 5 bits sous le format $(5,3)$.

D'après le tableau 5.2, une telle architecture d'émulateur de canal nécessite 3542 Flip-Flops, 4208 LUTs et 1 BRAM. La fréquence maximale f_{max} atteint 138MHz. En pratique, nous avons mis 3MHz pour la fréquence du canal, ce qui permet de générer 3 échantillons de bruit pendant un cycle d'horloge à 1MHz, soit la fréquence de transmission.

Tableau 5.2 — Synthèse du canal ABBG.

synthèse	Flip-Flop	LUT	BRAM	f_{max} (MHz)
nombre	3542	4208	1	138

5.3 Synthèse des décodeurs élémentaires SISOs

Dans cette section, nous présentons successivement les résultats de synthèse pour les différentes architectures proposées dans le chapitre 4. Dans un premier temps, nous rappelons l'architecture globale du récepteur. Dans un deuxième temps, nous donnons les résultats pour l'architecture classique, dite architecture TD (Turbo-Décodeur). Ensuite, la saturation des métriques de nœud, dite architecture TD-SM (Turbo-Décodeur avec Saturation des Métriques de nœud), est appliquée au turbo-décodeur et la synthèse associée est présentée. Les autres architectures utilisent l'architecture TD-SM par défaut. Nous détaillons ensuite la synthèse de l'architecture contenant le codage différentiel, dite architecture TD-CD (Turo-Décodeur avec Codage Différentiel). Puis, l'architecture du décodeur tenant compte de l'insertion d'erreurs, notée TD-IE (Turbo-Décodeur avec Insertion d'Erreurs) est synthétisée. Enfin, la synthèse de l'architecture TD-CA (Turbo-Décodeur avec Calcul Anticipé) utilisant le calcul anticipé de l'information extrinsèque est détaillée. Pour toutes ces implémentations, la fréquence de fonctionnement pour le turbo-décodeur est fixée à 50MHz.

5.3.1 Architecture globale du récepteur

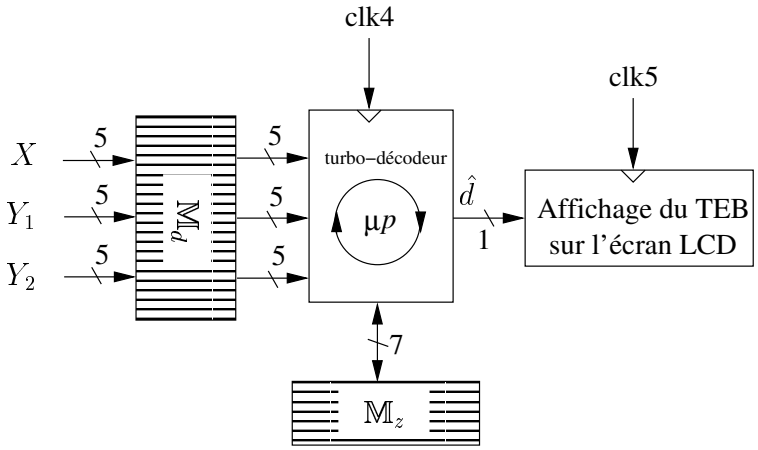


Figure 5.9 — Environnement de travail pour un turbo-décodeur

Dans le récepteur, l'architecture du turbo-décodeur est connectée à différents éléments : les mémoires de données $\mathbb{M}_d$, la mémoire de l'information extrinsèque $\mathbb{M}_z$ (figure 5.9). Le processus de turbo-décodage traite les données provenant des mémoires $\mathbb{M}_d$ et $\mathbb{M}_z$ et transmet l'information extrinsèque z à la mémoire $\mathbb{M}_z$ au cours des itérations. En parallèle, le TEB est calculé et affiché en temps réel à l'aide d'un écran LCD.

Les mémoires d'informations reçues X, Y_1 et Y_2 sont considérés comme des mémoires externes au décodeur SISO. La mémoire de l'information extrinsèque est également externe au décodeur. Cela signifie que nous ne prenons en compte que le traitement dans un décodeur élémentaire SISO. Ce contexte est celui retenue pour les synthèses. Durant nos expérimentations, les blocs autres que ceux associés au turbo-décodeur sont considérés comme la partie statique.

Ils sont implémentés qu'une seule fois dans la partie statique de la zone d'implémentation du circuit FPGA. Ainsi, le placement-routage de cette partie est contraint de façon définitive dès que l'implémentation est terminée. Par contre, la mémoire αLIFO nécessaire au stockage des métriques de nœud lors du premier parcours du treillis est considérée comme un composant interne au décodeur élémentaire SISO. le turbo-décodeur est considéré comme la partie dynamique. Il sera reconfiguré à chaque implémentation selon les architectures proposées dans le chapitre 4. En résumé, l'environnement de travail est exactement équivalent pour tous les décodeurs SISOs considérés grâce à la reconfiguration dynamique. Le placement-routage de la partie statique n'est pas modifié durant l'implémentation d'un décodeur SISO. Autrement dit, ces implémentations n'ont pas d'impact sur la partie statique.

Il est à noter que le turbo-décodeur et le périphérique LCD sont respectivement alimentés par des horloges clk_4 et clk_5, *i.e.*, $clk_4 = 50$MHz et $clk_5 = 3$MHz dans notre étude.

5.3.2 Synthèse de l'architecture TD

La première implémentation concerne l'architecture standard TD. Pour cette implémentation, les métriques de nœud, quantifiées sur 8 bits, ne sont pas saturées après normalisation. La taille de la mémoire αLIFO est égale à $(864 \times (8 \times 8))$ bits pour une trame de 864 bits. A partir des résultats fournis par l'outil XST, le tableau 5.3 présente la synthèse de l'architecture TD (Turbo-Décodeur). Nous constatons que cette architecture nécessite 633 Flip-Flops, 1398 LUTs et 4 blocs de mémoire. La fréquence maximale est égale à 53MHz. Cette architecture est la référence pour la suite de ce travail.

Tableau 5.3 — Synthèse de l'architecture TD.

synthèse	Flip-Flop	LUT	BRAM	f_{max} (MHz)
nombre	633	1398	4	53

5.3.3 Synthèse de l'architecture TD-SM

L'architecture TD-SM a la même interface pour les accès des signaux de contrôle et des bus de données que l'architecture TD, ce qui permet d'intégrer facilement ce décodeur SISO au projet précédent. Mais, le cœur de l'architecture TD-SM est modifié de manière à prendre en compte la saturation des métriques de nœud à chaque étage de décodage. Comme expliqué précédemment, les métriques de nœud sont alors saturées et puis quantifiées sur 5 bits au cours du décodage. Par conséquent, la taille de la mémoire αLIFO est diminuée à $(864 \times (5 \times 8))$ bits pour une trame de longueur 864 bits. De plus, la saturation, réalisée à l'aide des portes logiques, s'effectue à l'intérieur de la boucle du calcul récursif des métriques de nœud. Le chemin critique est donc plus lent par rapport à l'architecture TD-SM. Nous obtenons ainsi une fréquence maximale 52MHz. Le tableau 5.4 donne les résultats de synthèse de l'architecture TD-SM. Nous obtenons 437 Flip-Flops, 1206 LUTs et 3 BRAMs pour cette architecture.

Tableau 5.4 — Synthèse de l'architecture TD-SM.

synthèse	Flip-Flop	LUT	BRAM	f_{max} (MHz)
nombre	437	1206	3	52

5.3.4 Synthèse de l'architecture TD-CD

A partir de l'architecture TD-SM, un ré-encodage des données reçues est effectué à l'aide d'un codeur bidirectionnel défini dans le chapitre 4, avant de procéder au décodage proprement dit. Il est à noter que le décodeur SISO est basé sur l'architecture TD-SM. Comme le codage ne nécessite pas de retard, cette architecture, dite TD-CD, n'a pas de retard par rapport à l'architecture TD-SM. D'après le tableau 5.5, l'architecture TD-CD nécessite 459 Flip-Flops, 1357 LUTs et 3 BRAMs. La fréquence maximale f_{max} est de 52MHz.

Tableau 5.5 — Synthèse de l'architecture TD-CD.

synthèse	Flip-Flop	LUT	BRAM	f_{max} (MHz)
nombre	459	1357	3	52

5.3.5 Synthèse de l'architecture TD-IE

A partir de l'architecture TD-CD, des erreurs supplémentaires sont insérées au cours du codage différentiel, ce qui permet de ramener au maximum le chemin survivant au chemin TAZ. L'insertion d'erreurs ne concerne que la détection de motif répétitif, qui est réalisée à l'aide du registre à décalage, il n'y a donc pas d'augmentation en terme de LUT. En parallèle, les erreurs insérées sont mémorisées lors du premier parcours du treillis. Pour ce faire, une mémoire supplémentaire est nécessaire par rapport à l'architecture TD-CD. Au niveau synthèse, nous obtenons alors 465 Flip-Flops, 1357 LUTs et 4 BRAMs pour cette architecture (voir le tableau 5.6). La fréquence maximale f_{max} est de 54MHz.

Tableau 5.6 — Synthèse de l'architecture TD-IE.

synthèse	Flip-Flop	LUT	BRAM	f_{max} (MHz)
nombre	465	1357	4	52

5.3.6 Synthèse de l'architecture TD-CA

Pour profiter au maximum du chemin TAZ, l'information extrinsèque peut être calculée de manière anticipée. Ainsi, les accès (lecture et écriture) de la mémoire αLIFO sont déterminés par les conditions d'anticipation. Dans ce cas, toutes les métriques de nœud ne sont pas écrites dans la mémoire. De même, la lecture de la mémoire n'est effectuée que si l'information extrinsèque du bit considéré n'est pas calculée de manière anticipée. Ainsi, nous obtenons moins d'accès à la mémoire αLIFO par rapport aux architectures précédentes. Cependant, cette approche a un coût de mise en œuvre comme nous allons le voir. En effet, l'écriture de la mémoire αLIFO est sauvegardée dans une mémoire dédiée, d'où une mémoire de plus par rapport à l'architecture TD-IE. Au niveau synthèse, le tableau 5.7 montre que l'architecture TD-CA nécessite 544 Flip-Flops, 1487 LUTs et 5 BRAMs avec une fréquence maximale $f_{max} =$ 47MHz.

Tableau 5.7 — Synthèse de l'architecture TD-CA.

synthèse	Flip-Flop	LUT	BRAM	f_{max} (MHz)
nombre	544	1487	5	47

5.3.7 Bilan sur la complexité matérielle des différentes solutions architecturales

Grâce à la saturation des métriques de nœud, la complexité du module pour calculer le LLR est diminuée par rapport à l'architecture classique TD. D'après les résultats de synthèse, le module LLR de l'architecture TD nécessite 194 Flip-Flops, 17 additionneurs et 14 comparateurs. L'architecture TD-SM, quant à elle, nécessite 156 Flip-Flops, d'où une diminution de 38 Flip-Flops par rapport à l'architecture TD. Les nombres des additionneurs et des comparateurs ne sont pas changés. Mais, le nombre de mémoires est diminué de 4 BRAMS à 3 BRAMs.

Lorsque le codage différentiel est appliqué, l'architecture TD-CD nécessite une augmentation de 16 Flips-Flops, de 4 comparateurs, d'un compteur et d'une porte ou-exclusif par rapport à l'architecture TD-SM. Au niveau du nombre de mémoires, 3 BRAMs sont obtenues pour cette architecture. L'architecture TD-IE permet d'insérer les erreurs supplémentaires au cours du codage à la réception. Cette fonction nécessite alors d'ajouter une mémoire supplémentaire de 870 bits pour stocker les erreurs insérées lors du premier parcours du treillis (voir le chapitre 4). C'est pourquoi nous obtenons 4 BRAMs d'après la synthèse. En outre, une augmentation d'un comparateur, de 7 additionneurs et de 21 Flip-Flops est constatée par rapport à l'architecture TD.

Concernant l'architecture TD-CA, qui permet de calculer l'information extrinsèque de manière anticipée, une mémoire supplémentaire de 870 bits est nécessaire pour sauvegarder l'écriture des métriques de nœud à chaque bit considéré (voir le chapitre 4) par rapport à l'architecture TD-IE. Ainsi, nous obtenons 5 BRAMs à l'issue de la synthèse. De plus, la mémoire $\mathbb{M}_1$ de la figure 4.24 et la gestion associée ainsi que les retards τ augmentent sa complexité. Par rapport à l'architecture classique TD, nous constatons alors une augmentation respective de 246 Flip-Flops, de 28 additionneurs, de 1 compteur et de 4 comparateurs pour réaliser cette approche.

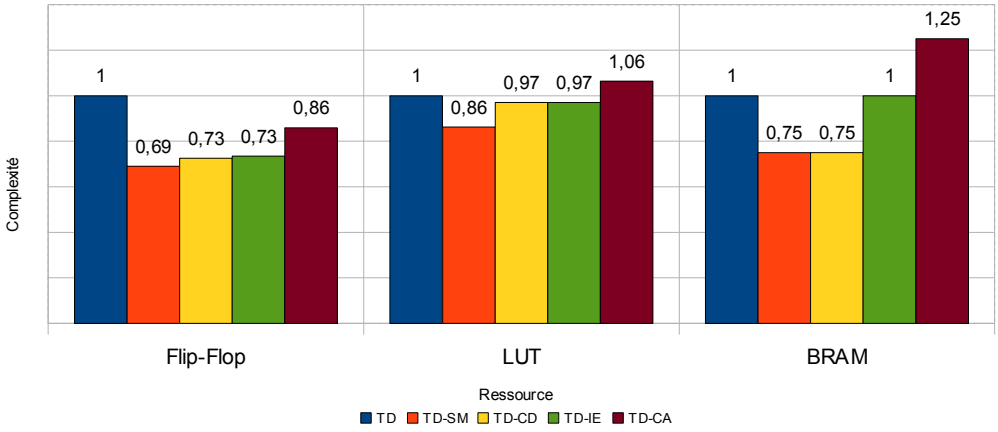


Figure 5.10 — Bilan de la complexité pour les différentes solutions architecturales.

La figure 5.10 récapitule la complexité des solutions architecturales en terme de Flip-Flop, de LUT et de BRAM à partir des différentes synthèses réalisées. Prenons l'architecture TD comme l'architecture de base. Nous obtenons des réductions des ressources Flip-Flop, LUT et BRAM de 31%, de 14% et de 25% pour l'architecture TD-SM. De plus, des réductions de 27%, de 3% et de 25% sont obtenues pour l'architecture TD-CD. Pour l'architecture TD-IE, les nombres de Flip-Flops et de LUTs sont respectivement diminués de 27% et de 3%. Il n'y a pas

d'augmentation en terme de BRAM. La complexité de l'architecture TD-CA augmente par rapport à l'architecture classique TD. En effet, nous obtenons des augmentations respectives des ressources LUT et BRAM de 6% et de 25%. Par contre, une diminution de 24% est obtenue en nombre de Flip-Flops. Cette architecture peut alors présenter un surcoût en terme de consommation des ressources supplémentaires allouées.

5.3.8 Bilan en terme de performance des différentes solutions architecturales investiguées

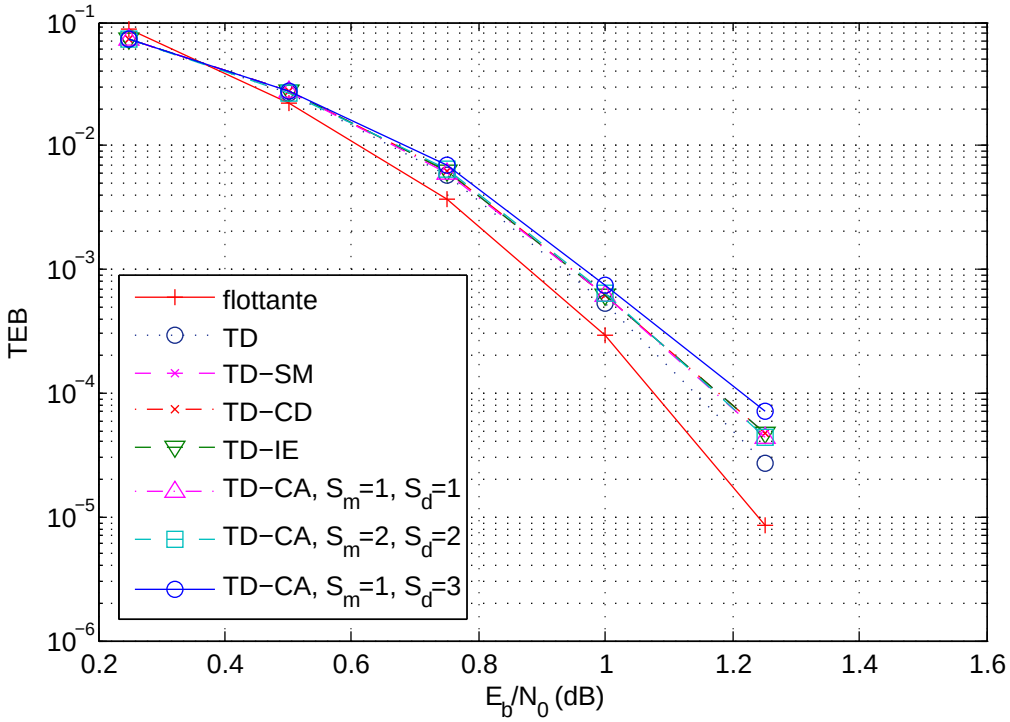


Figure 5.11 — Bilan en terme de performance des différentes solutions architecturales investiguées.

La figure 5.11 présente les performances du turbo-décodeur en terme de TEB pour les différentes architectures implémentées. La courbe, dite *flottante*, représente les performances obtenues par les simulations fonctionnelles (langage C) avec le générateur de donnée de la figure 5.3. Les données ont alors une représentation en valeur réelle. Pour les autres courbes, dites version *quantifiée*, les données sont quantifiées sous le format $(Q_m, 3)$, *i.e.*, $Q_m = 5$ bits pour les éléments X, Y_1 et Y_2 et $Q_m = 7$ pour l'information extrinsèque z. Le pas de quantification est égal à 2^{-3} au cours du décodage pour les architectures quantifiées. Leurs performances sont obtenues à l'aide des prototypes implémentés sur le circuit FPGA. Il s'agit donc de courbes mesurées et non simulées comme celles obtenues dans les chapitres précédents.

Par comparaison de référence, soit la courbe flottante, nous obtenons 0.07dB de dégradation pour un TEB de l'ordre de 10^{-4} lorsque les données sont quantifiées dans le cas de l'architecture TD. De plus, une dégradation supplémentaire de 0.04dB est constatée lorsque la saturation des métriques de nœud est appliquée (voir la courbe de l'architecture TD-SM).

Comme prévu (*c.f.* l'explication théorique du chapitre 2), le codage différentiel et l'insertion d'erreurs n'introduisent pas de perte en terme de performances (voir les courbes TD-SM, TD-CD et ID-IE).

Rappelons que les symboles S_m et S_d représentent respectivement les seuils pour repérer un motif répétitif et pour estimer le chemin survivant à l'aide de la distance de Hamming. Lorsque le couple (S_m, S_d) est égal à $(1, 1)$ ou $(2, 2)$, il n'y pas de dégradation des performances par rapport à la courbe TD-IE. En revanche, nous obtenons 0.06dB de dégradation supplémentaire pour un TEB=10^{-4} lorsque $(S_m, S_d) = (1, 3)$. En conclusion, les courbes mesurées lors du prototypage des différentes architectures sont conformes à la précision. En effet, des dégradations inférieures à 0.1dB ont été observées.

5.3.9 Bilan sur la consommation

Rappelons que la technique *reconfiguration partielle* est appliquée au cours de l'implémentation. Il n'est alors pas nécessaire de modifier le placement-routage de la partie statique du circuit FPGA, mais uniquement la partie dynamique. Pour mesurer la consommation du circuit associée aux différentes solutions architecturales, il suffit alors de reconfigurer la partie dynamique de manière à s'adapter à l'architecture visée. Cette procédure assure que les résultats de mesure s'appuient sur le même contexte de travail pour le turbo-décodeur.

Nous nous intéressons en particulier à la consommation du cœur du circuit FPGA sur lequel l'application est implémentée.

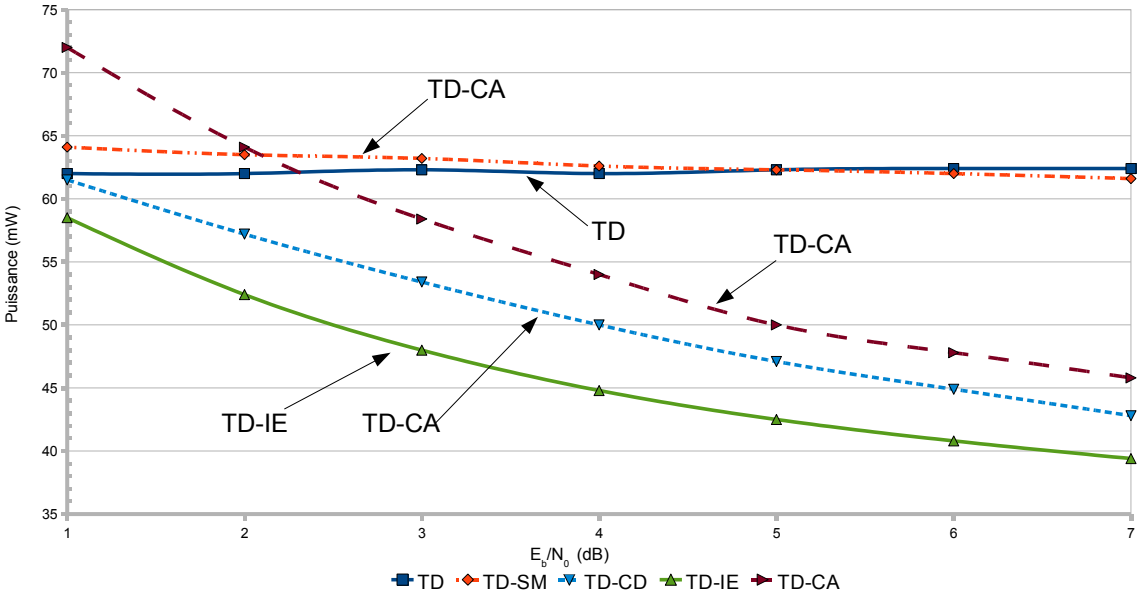


Figure 5.12 — Bilan sur les puissances respectives des solutions architecturales.

La figure 5.12 donne les résultats de mesure sur les puissances consommées par les différentes solutions architecturales en fonction du rapport signal à bruit. Les légendes correspondent aux solutions architecturales retenues. Nous considérons les puissances consommées par l'architecture classique TD comme référence. Les valeurs des puissances sont ensuite exprimées en pourcentage dans la figure 5.12. Nous obtenons une puissance quasi-stable de 62mW, indépendante du rapport signal à bruit lorsque la solution architecturale TD est retenue.

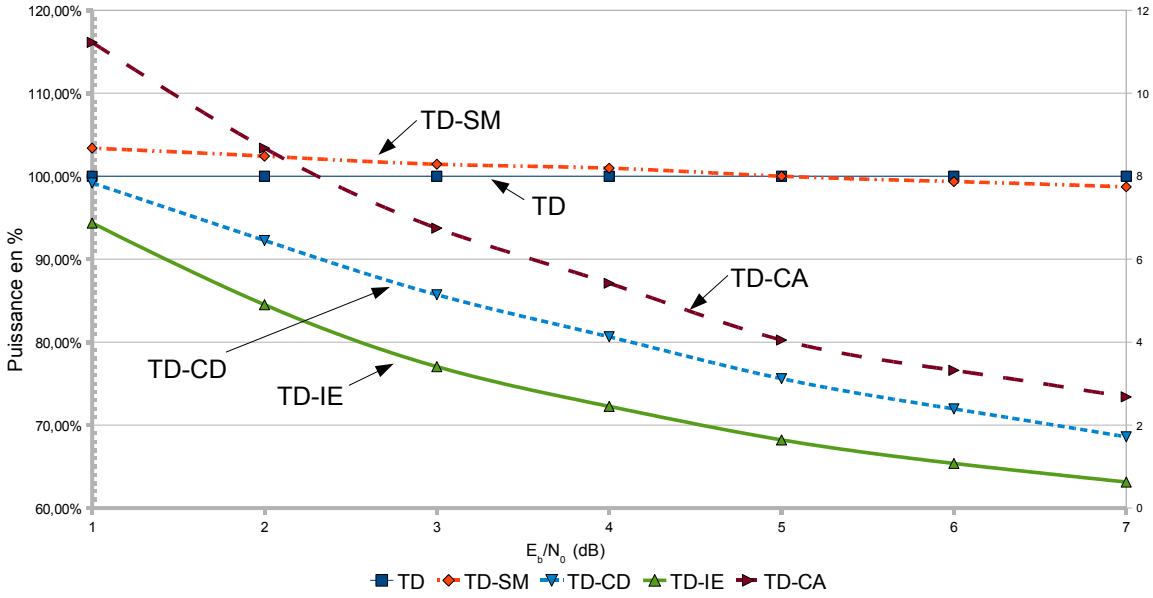


Figure 5.13 — Bilan sur les puissances respectives en pourcentage des solutions architecturales.

D'après les courbes TD-SM, la puissance consommée par l'architecture TD-SM augmente légèrement par rapport à la référence lorsque le rapport signal à bruit est faible. Nous obtenons une puissance de 64.1mW correspondant à SNR=1dB, soit une légère augmentation de 3%. Cependant, cette puissance diminue lorsque le rapport signal à bruit est élevé. Par exemple, une puissance de 61.6mW est obtenue pour un SNR=7dB, d'où une diminution de 1%.

Les courbes TD-CD correspondent à la puissance du turbo-décodeur lorsque la solution TD-CD est retenue. Conformément à nos approches, la puissance du turbo-décodeur est fonction du rapport signal à bruit, c'est-à-dire, elle diminue lorsque le rapport signal à bruit augmente. Cela est également vrai pour les architectures TD-IE et TD-CA (voir les courbes TD-IE et TD-CA des figures 5.12 et 5.13) que nous allons voir dans la suite. Nous obtenons des puissances de 61.5mW et de 42.8mW correspondants respectivement à des SNRs valent de 1dB et de 7dB. Nous vérifions alors que la diminution de la puissance n'est pas notable lorsque le rapport signal à bruit est faible, *i.e.*, seulement 1% pour un SNR=1dB. Dans le cas contraire, la diminution est significative. Nous obtenons alors une diminution de 31% pour un SNR=7dB. Concernant l'architecture TD-IE, nous obtenons des puissances de 58.5mW et de 39.4mW correspondants respectivement des rapports signal à bruit allant de 1dB à 7dB. Alors, seule une diminution de 6% est obtenue lorsque le rapport signal à bruit est égal à 1dB. En revanche, nous obtenons 37% de diminution de consommation lorsque ce rapport est de 7dB. Comme nous avons présenté dans la section précédente, la dernière architecture TD-CA représente une augmentation de complexité due aux mémoires ajoutées. Cette augmentation est contraignante au niveau consommation. Nous obtenons une puissance de 72mW, soit une augmentation de 16% pour un rapport signal à bruit de 1dB. Cependant, lorsque ce rapport devient significatif, par exemple, 7dB, nous obtenons tout de même une puissance de 45.8mW, soit 27% de diminution par rapport à l'architecture classique TD. Par ailleurs, en mesurant les puissances fournies par les alimentations $Alim_2$ et $Alim_3$ (voir les numéros 8 et 9 de la figure 5.1), nous obtenons des puissances respectives stables de 86.4mW et de 856.6mW, qui correspondent en effet aux puissances des périphériques du circuit FPGA et de la carte.

5.4 Conclusions

Nous avons d'abord présenté l'environnement de mesure de consommation du circuit FPGA dans ce chapitre. La technique *reconfiguration dynamique* a été utilisée pour nos implémentations. Ensuite, l'architecture de la partie émettrice a été présentée. Cette architecture correspond à la norme UMTS, qui nécessite deux codeurs convolutifs récursifs systématiques à 8 états en concaténation parallèle. Au niveau de la synthèse sur cible FPGA, nous nous intéressons en particulier aux ressources nécessaires en terme de Flip-Flop, de LUT et de BRAM pour chaque architecture considérée.

A l'aide de l'outil de synthèse XST, l'architecture de la partie émettrice nécessite 132 Flip-Flops, 277 LUTs et 2 BRAMs. Un émulateur de canal ABBG a été implémenté dans notre étude. Cette implémentation est basée sur la méthode Box-Muller et le théorème de la limite centrale. L'architecture d'un tel émulateur de canal nécessite 3542 flip-Flops, 4208 LUTs et 1 BRAM.

Du côté récepteur, les architectures proposées dans le chapitre 4 sont toutes implémentées et les différentes synthèses sont données. La complexité des solutions architecturales a été étudiée d'après les résultats de synthèse. En prenant l'architecture TD comme référence, l'architecture TD-SM représente des réductions respectives de 31%, de 14% et de 25% en terme de nombre de Flip-Flops, de LUTs et de BRAMs. Quant à l'architecture TD-CD, elle montre des réductions respectives de 27%, de 3%, de 25%. Pour l'architecture TD-IE, nous obtenons des réductions de 27% et de 3% en terme de Flip-Flop et de LUT. Par apport à l'architecture TD, nous obtenons 14% de réduction sur le nombre de Flip-Flops pour l'architecture TD-CA. En revanche, des augmentations respectives de 6% et de 25% sont obtenues sur le nombre de LUTS et de BRAMs. Au niveau des performances, les courbes mesurées de TEB sont conformes aux résultats attendus.

Au niveau de la consommation, nous avons montré les puissances consommées par les différentes solutions architecturales de turbo-décodeur. De plus, la puissance de turbo-décodeur classique est stable, indépendante du rapport signal à bruit. En revanche, la puissance de turbo-décodeur différentiel est fonction du rapport signal à bruit. Lorsque le rapport signal à bruit augmente, la puissance associée diminue. En terme de puissance , nous avons obtenu des diminutions de 37% et de 27% correspondant respectivement aux architectures TD-IE et TD-CA.

Conclusion et perspectives

Conclusion

Dans un système de communications numériques, il s'avère que les codes correcteurs d'erreurs sont des techniques efficaces pour lutter contre le bruit introduit lors de la transmission du message. En particulier, les turbocodes inventés en 1993 ont des performances proches de la limite de Shannon. C'est la raison pour laquelle ils ont été adoptés par de nombreux standards de télécommunications (UMTS, CDMA2000, DVB-RCS, IEEE802.16). L'utilisation des turbocodes pose néanmoins de nouvelles contraintes dues au décodage itératif. Parmi ces contraintes, la maîtrise de la consommation d'un turbo-décodeur devient primordial. En effet, des études ont montré que le budget énergétique d'un turbo-décodeur dans un système de communications peut atteindre 50% de la consommation globale du récepteur [Bougard 06]. C'est pourquoi, le décodage de canal constitue un problème majeur au niveau consommation. Or, pour une technologie CMOS, la consommation d'un circuit se décompose principalement en deux sources : la consommation statique (proportionnelle à la surface de silicium utilisée) et la consommation dynamique (proportionnelle au taux de transition des portes logiques ou activité). Deux types d'exploration sont donc possibles pour réduire la consommation : diminuer la surface et maîtriser l'activité.

L'algorithme MAP est l'algorithme de référence pour le décodage de codes convolutifs. Cependant la complexité de cet algorithme rend sa mise en œuvre peu réaliste. Nous avons ainsi considéré sa version simplifiée Max-Log-MAP dans ce projet. De nombreuses études algorithmiques et architecturales ont été menées pour tenter de maîtriser la consommation d'un turbo-décodeur. Depuis 1995, les recherches au niveau architectural se sont focalisées selon deux axes : réduire les accès mémoires et diminuer la surface du circuit. En effet, il est important de noter dès à présent que l'aspect mémorisation joue un rôle primordial pour la gestion de la consommation des architectures de turbo-décodage. En clair, l'architecture d'un décodeur SISO a déjà bien été exploré. Cependant, nous avons investigué deux techniques complémentaires favorisant la maîtrise de la consommation. f Nous avons ainsi réexaminé les architectures de décodeur dédiées aux turbocodes convolutifs pour réduire la consommation. Notre première contribution a consisté à montrer que la taille de la mémoire des métriques de nœud pouvait être réduite en normalisant et en saturant les métriques de nœud et ce, sans perte significative de performance. Un atout de cette approche réside dans le fait qu'elle est complémentaire avec les autres techniques d'optimisation existantes, a savoir fenêtre glissante et critère d'arrêt. En effet, dans chaque étage du treillis, les métriques de nœud ayant des valeurs élevées par rapport aux autres ne contribuent pas de manière significative à la décision finale. La saturation de ces métriques n'introduit donc pas de dégradation significative en terme de performance. Ainsi, la taille de la mémoire dans laquelle les métriques de nœud

sont stockées diminue de façon significative. La propriété itérative de l'algorithme de turbo-décodage implique que ce type de décodage soit très robuste face aux erreurs résiduelles. Des erreurs de décodage liées à la saturation au cours d'une itération peuvent alors être corrigées lors des itérations suivantes. C'est la raison pour laquelle la dégradation en terme de performance est minimale. Nous obtenons une diminution de 43% sur la taille de mémoire au prix d'une perte de performance de 0.1dB.

La saturation des métriques de nœud favorise surtout une diminution de la consommation statique. Par ailleurs, il est possible de maîtriser la consommation dynamique d'un turbo-décodeur en diminuant son activité. En effet, lors du turbo-décodage, la plupart des erreurs sont corrigées au cours des premières itérations. Il est donc a priori possible d'exploiter cette propriété pour proposer une diminution de l'activité d'un turbo-décodeur au cours de son processus itératif. Pour ce faire, nous avons étudié un message particulier : le message tout à zéro. Dans ce cas, le décodeur ne traite que le vecteur du message erroné. Si toutes les erreurs sont corrigées ou en absence d'erreurs, le chemin survivant reste sur le chemin tout à zéro lors du décodage. Le taux de transition binaire au sein du décodage diminue. Donc, l'activité d'un turbo-décodeur concerne à la fois le calcul intensif au sein des décodeurs élémentaires, mais également la gestion des accès aux mémoires. Nous nous sommes attachés à apporter des solutions sur les deux aspects. Notre travail a abouti à la proposition d'une technique de décodage que nous avons appelé turbo-décodage différentiel. Cette technique constitue la contribution majeure de cette thèse. Son principe consiste à reconstruire un mot de code en réception puis à appliquer un processus de décodage sur la différence avec le mot reçu. Pour ce faire, un ré-encodage à partir des valeurs reçues a été proposé. Cette technique a été préalablement appliquée aux codes en blocs et à l'algorithme de Viterbi. Lorsqu'elle est appliquée aux codes convolutifs, elle permet d'extraire les vecteurs d'erreur à partir des éléments reçus. Dans un premier temps, nous avons appliqué cette technique à l'algorithme Max-Log-MAP. L'objectif de l'étude était de tenter de ramener au maximum le chemin survivant sur le chemin tout à zéro. Cependant, notre étude a montré qu'un simple ré-encodage ne suffisait pas pour atteindre cet objectif. En effet, l'apparition d'un motif répétitif à l'issue du ré-encodage ne permet pas au chemin survivant de converger vers le chemin tout à zéro. Pour limiter cet effet, nous avons proposé d'insérer des dummy-errors au cours de l'étape de ré-encodage. Cette insertion permet de générer une séquence de motifs similaires au motif répétitif. Ainsi, une opération ou-exclusif lors du codage différentiel a permis de supprimer les motifs répétitifs. Grâce à cette insertion, notre etude a montré que le chemin survivant a été ramené au chemin tout à zéro au cours du décodage.

La technique de ré-encodage, qui a aboutit à un parcours du treillis favorisant le chemin tout à zéro, favorise une maîtrise des accès mémoires et donc de la consommation dynamique résultante. C'est pourquoi, nous avons profité de cette approche pour diminuer le nombre d'accès mémoires. En effet, il est dès lors possible de calculer de manière anticipée l'information extrinsèque, sans avoir besoin de la contribution des métriques de nœud du second parcours du treillis. Il n'est alors pas nécessaire de sauvegarder les métriques de nœud lors du premier parcours du treillis et de les lire lors du second parcours. A l'aide de l'estimation du chemin survivant et du motif répétitif, l'ensemble des techniques proposées ne génère pas de perte significative en terme de performance. En revanche, nous obtenons une réduction de 48% de l'activité de l'architecture par rapport au turbo-décodage classique.

Ainsi, les techniques proposées permettent d'une part de réduire l'activité de l'algorithme de décodage Max-Log-MAP, et d'autre part, de limiter le nombre d'accès mémoires durant le processus de décodage itératif. Ces différentes investigations ont été mises en oeuvre au sein d'une architecture de turbo-décodage. Ce travail a nécessité la conception, l'intégration

et le prototypage sur circuit FPGA d'un turbo-décodeur pour la norme UMTS. Une chaîne complète de communications numériques a par ailleurs été implémentée sur une carte de prototypage. Ainsi, les investigations au niveau algorithmique ont été suivies de la définition des architectures de turbo-décodeur. Pour estimer la consommation réelle des architectures, des mesures ont enfin été faites pour une cible FPGA. Par rapport à une architecture classique de turbo-décodeur, nous obtenons une réduction de 31% de la consommation lorsque la technique ré-encodage est appliqué au turbo-décodeur. De plus, une réduction de 37% est obtenue lorsque des erreurs supplémentaires sont insérées au cours du ré-encodage. Pour la dernière solution architecturale, soit l'ensemble de techniques proposées sont appliquées, nous obtenons une réduction de 27% de la consommation.

Perspectives : algorithme et architecture

Au niveau algorithmique, le décodage différentiel a été appliqué au turbo-décodeur binaire. Dans un système de turbo-décodage différentiel utilisant le principe du calcul anticipé de l'information extrinsèque, la légère perte de performances est due au fait que l'information extrinsèque anticipée n'est pas obtenue par l'algorithme Max-Log-MAP. Cependant, il serait possible d'établir une relation entre les métriques de nœud et les symboles reçus à l'entrée du décodeur élémentaire SISO lors du calcul anticipé, telle que l'information mutuelle entre l'information extrinsèque anticipée et l'information extrinsèque explicite soit maximale. Par ailleurs, le principe du codage différentiel est indépendant du type de code. Cela signifie que le principe du codage différentiel et de l'ensemble des techniques proposées dans cette thèse, est aussi applicable pour des codes doubles binaires, comme ceux spécifiés dans la norme WiMax [IEEE 04]. Mais, dans ce cas, le comportement d'un tel turbo-décodeur reste à étudier au niveau des performances et de la consommation.

Du côté de la mise en œuvre, nous avons retenue une cible FPGA pour permettre un prototypage rapide de nos solutions architecturales. Cependant, cette implémentation ne nous a pas donné de résultats à la hauteur de nos attentes en terme de consommation. Pour mesurer de manière précise l'impact sur la consommation dynamique de nos approches, de futurs travaux consisteraient à implémenter les architectures sur un circuit ASIC. De plus, il est possible d'optimiser les architectures de manière à accentuer la diminution de la consommation. En effet, l'insertion d'erreurs n'est pas bidirectionnelle lors du codage différentiel. Pour contourner ce problème, nous avons proposé de sauvegarder des erreurs insérées dans une mémoire dédiée lors du premier parcours du treillis. Ensuite, au lieu de détecter le motif lors du second parcours, les erreurs sont directement lues dans cette mémoire. Or, la mémoire a un impact dans l'architecture d'un turbo-décodeur en terme de consommation. Il serait judicieux d'optimiser l'architecture pour que l'insertion d'erreurs soit bidirectionnelle. Cela implique que le vecteur d'erreurs inséré lors du premier parcours du treillis est identique à celui du second parcours, mais dans le sens inverse, comme c'est le cas pour notre codeur bidirectionnel. De même, une mémoire supplémentaire pour sauvegarder l'écriture des métriques de nœud est nécessaire. Il est pertinent de supprimer ces deux mémoires en terme de consommation lors de la mise en nœuvre de nos solutions architecturales.

Comme nos solutions architecturales sont indépendantes des études préalables favorisant la maîtrise de la consommation. Il est possible de les combiner avec des techniques existantes comme par exemple un critère d'arrêt ou une fenêtre glissante. Dans cette thèse, nous avons privilégié l'aspect consommation parmi les critères d'optimisation. Néanmoins, les futurs systèmes de communications comme WiMax, demande des débits de l'ordre du Gb/s. En réalité,

nous devons donc tenir compte de critères tels que le débit ou la fréquence. Au département électronique de Télécom Bretagne, des travaux de recherche [Muller 07] [Leroux 08] ont investigué le turbo-décodage à très haut débit ($>$ 1 Gb/s). Une autre perspective serait d'associer notre approche de codage différentiel avec ces travaux. Il s'agirait d'optimiser les architectures existantes pour favoriser un compromis entre la consommation et les débits. Enfin, le turbo-décodage différentiel prend en considération la consommation au niveau algorithmique. Cette technique peut donc être étendue aux techniques de turbo-communications, tel que la turbo-équalisation ou la turbo-démodulation. Dans ce cas, il faudrait également apporter des solutions pour diminuer l'activité des fonctions associées au décodage souple.

Bibliographie

[3GPP 99] 3GPP. *Universal Mobile Telecommunications System (UMTS).* Multiplexing and channel coding (FDD), vol. 3GPP TS 25.212 version 3.7.0, 1999.

[802.16a 03] IEEE Std 802.16a. *IEEE Standard for local and metropolitan area networks.* Available at http ://standards.ieee.org/getieee802/download/802.16a-2003.pdf, 2003.

[Alfke 96] Peter Alfke. *Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators.* XAPP 052 (Version 1.1), July 7 1996.

[Atluri 03] I. Atluri & T. Arslan. *Low power architectures for the MAP decoder with the optimised memory sizes.* IEEE MWSCAS '03, vol. E89-A, no. 3, pages 1520 – 1523, Sep. 2003.

[Bahl 74] L. R. Bahl, J. Cocke, F. Jelinek & J. Raviv. *Optimal decoding of linear codes for minimizing symbole error rate.* IEEE Trans. Inform. Theory, IT-20, pages 248–287, March 1974.

[Belady 66] L.A. Belady. *A study of replacement algorithms for a virtual-storage computer.* IBM Systems Journal, vol. 5, N.2, May 1966.

[Benedetto 97] S. Benedetto, D. Divsalar, G. Montorsi & F. Pollara. *A soft-input soft-output APP module for iterative decoding of concatenated codes.* Communications Letters, vol. 1, Issue 1, pages 22–24, Jan. 1997.

[Berlekamp 84] E. R. Berlekamp. Algebraic coding theory. revised edition, Aegean, 1984.

[Berlekamp 96] E. Berlekamp. *Bounded distance +1 softdecision Reed-Solomon decoding.* IEEE Transactions on Information Theory, vol. 42, no. 3, pages 704–720, May 1996.

[Berrou 93] C. Berrou, A. Glavieux & P. Thitimajshima. *Near Shannon limit error-correcting coding and decoding : Turbo-Codes.* Proc. of IEEE ICC '93, vol. 2/3, pages 1064–1070, May 1993.

[Berrou 07] C. Berrou & *al. codes et turbocodes.* Springer ISBN 2-287-32739-8, 2007.

[Bougard 06] B. Bougard. *Cross-Layer Energy Management in Broadband Wireless Transceivers.* PhD dissertation, March 2006.

[Boutillon 03a] E. Boutillon, J.L. Danger & A. Gazel. *Design of High Speed AWGN Communication Channel Emulator.* Kluwer Press, Analog Integrated Circuits and Signal Processing 34(2), pages 133–142, Feb. 2003.

[Boutillon 03b] E. Boutillon, W. Gross & P. Gulak. *VLSI Architectures For the MAP Algorithm.* IEEE Trans. on communication, vol. 51, no. 2, pages 175–185, Feb. 2003.

[Boutillon 07] E. Boutillon, C. Douillard & G. Montorsi. *Iterative Decoding of Concatenated Convolutional Codes : Implementation Issues.* IEEE Trans., vol. 99, no. 6, June 2007.

[Cain 84] J. Cain. *CMOS VLSI implementation of decoder* $r = 1/2$, $k = 7$. IEEE Nat. Aerospace Electron. Conf., NAECOM, vol. 1, pages 20–27, 1984.

[C.Berrou 01] C.Berrou, Y.Saouter, C.Douillard, S.Kerouédan & M.Jézéquel. *Interaction channel for digital terrestrial telecision.* ETSI EN 301 958, vol. 1.1.1, pages 28–30, Aug. 2001.

[C.Berrou 04] C.Berrou, Y.Saouter, C.Douillard, S.Kerouédan & M.Jézéquel. *Designing good permutations for turbocodes : towards a single model.* ICC'04, April 2004.

[CCSDS 98] CCSDS. *Recommandations for Space Data Systems Telemetry Channel Coding.* Blue Book, May 1998.

[Choi 06] H. Choi, J. Kim & I. Park. *Low power hybrid turbo decoding based on reverse calculation.* ISCAS 2006, pages 2053–2056, 2006.

[Dingninou 01] A. Dingninou. *Implémentation de turbocodes pour les trames courtes.* thèse, $N°$ d'ordre 817, pages 73–75, Juill. 2001.

[Elassal 03] M. Elassal & M. Bayoumi. *A low power turbo decoder architecture.* IEEE SIPS 2003, pages 27–29, Aug. 2003.

[Elias 54] P. Elias. *Error-Free Coding.* IRE, Trans. Information Theory, vol. PGIT-4, pages 29–37, Sep. 1954.

[Elias 55] P. Elias. *Coding for noisy channels.* IRE, Conv. REC., vol. 4, pages 37–47, 1955.

[ETSI 00] ETSI. *Interaction channel for satellite distribution systems.* ETSI EN 301 790, vol. 1.2.2, pages 21–24, Dec. 2000.

[Faggin 72] F. Faggin, M. Shima, M.E. Hoff, H. Feeney & S. Mazor. *The MCS-4 : An LSI Microcomputer System.* IEEE Region Six Conference, 1972.

[Fano 63] R.M. Fano. *a Heuristic Discussion of Probabilistic Decoding.* IEEE Transaction on Information Theory, vol. IT-9, pages 67–74, April 1963.

[Forney 73] G.D. Forney. *The Viterbi Algorithm.* Proc. IEEE, pages 61 :268–278, Mar. 1973.

[Gallager 62] R. G. Gallager. *Low Density Parity Check Codes.* IRE Trans. Inform. Theory, vol. IT, pages 21–28, January 1962.

[Garrett 01] D. Garrett, B. Xu & C. Nicol. *Energy efficient turbo decoding for 3G mobile.* Low Power Electronics and Design, International Symposium on, pages 328–333, August 2001.

[Gilbert 01] F. Gilbert, A. Worm & N. Wehn. *Low power implementation of a turbo-decoder on programmable architectures.* EDA Technofair Design Automation Conference Asia and South Pacific, vol. ISBN :0-7803-6634-4, pages 400 – 403, 2001.

[Glavieux 96] A. Glavieux & M. Joindot. communications nuériques introductions. Masson, 1996.

[Glavieux 05] A. Glavieux & *al.* Codage de canal, des bases théoriques aux turbocodes. Hermes science ISBN 2-7462-0953-5, 2005.

[Gnaëdig 03] D. Gnaëdig, E. Boutillon, M. Jezequel, V. Gaudet & G. Gulak. *On Multiple Slice Turbo Code.* The 3nd International Symposium on Turbo Codes and related Topics, pages 343 – 346, Sep. 2003.

[Gross 02] W.J. Gross, F.R. Kschischang, R. Koetter & R.G. Gulak. *A VLSI architecture for interpolation in soft-decision list decoding of Reed-Solomon codes.* Signal Processing Systems, 2002. (SIPS '02). IEEE Workshop on, vol. 15, pages 39–34, 16-18 Oct. 2002.

[H. Al-Zoubi 04] A. Milenkovic H. Al-Zoubi & M. Milenkovic. *Performance evaluation of cache replacement policies for the SPEC CPU2000 benchmark suite, ACM-SE 42.* Proceedings of the 42nd annual southeast regional conference, vol. ACM Press, pages 267–272, May 2004.

[Hagenauer 89] J. Hagenauer & P. Hoeher. *A Viterbi Algorithm With Soft-Decision Outputs And Its Applications.* Proc. of the IEEE GLOBECOM, pages 47.1.1–47.1.7, Nov 1989.

[Hagenauer 96] J. Hagenauer, E. Offer & L. Papke. *Iterative decoding of binary block and convolutional code.* IEEE Trans. on info. Theory, vol. IT-42, March 1996.

[Hekstra 89] A. P. Hekstra. *An alternative to Metric rescaling in viterbi decoders.* IEEE Trans. on communication, vol. 37, no. 11, pages 1220–1222, Nov. 1989.

[IEEE 04] IEEE. *IEEE Standard for Local and metropolitan area networks, IEEE Std 802.16-2004.* IEEE Standards Board, 2004.

[Intel 08] Intel. *Quad-Core Intel Xeon Processor 5400 Datasheet.* Aug. 2008.

[ITRS] ITRS. *available on line at http ://public.itrs.net.* International Technology Road map for Semiconductors.

[Jin 07] J. Jin & C. Tsui. *Low-Power Limited-Search Parallel State Viterbi Decoder Implementation Based on Scarce State Transition.* IEEE Trans. on VLSI, vol. 15, pages 1172–1176, Oct. 2007.

[Knuth 98] D.E. Knuth. *The art of computer programming.* ADDISON-WESLEY, 1998.

[Kubota 86] S. Kubota, K. Ohtani & S. Kato. *High-speed and high-coding-gain Viterbi decoder with low power consumption employing SST (scare state transition) scheme.* Electronics Letters, vol. 22, issue 9, no. 491-493, 24 April 1986.

[Kubota 93] S. Kubota, S. Kato & T. Ishitani. *Novel Viterbi Decoder VLSI Implementation And Its Performance.* IEEE Trans. Commun., vol. 41, pages 1170–1178, Aug 1993.

[Lee 05] D. Lee & I. Park. *Low-power log-MAP turbo decoding based on reduced metric memory access.* IEEE ISCAS, vol. 4, pages 3167–3170, May 2005.

[Leroux 08] C. Leroux. *Comception d'architectures parallèles de turbo-décodeurs de codes produits : De l'exploration à la mise en œuvre.* PH.D. dissertationn, Université de Bretagne-Sud, 2008.

[Leung 99] O. Leung, C. Yue, C. Tsui & R. Cheng. *Reducing Power Consumption of Turbo Code Decoder Using Adaptive Iteration with Variable Supply Voltage.* ISLPED'99, pages 36 – 41, Aug. 1999.

[Lin 07] D. Lin, C. Lin, C. Chen, H. Chang & C. Lee. *A Lower-Power Viterbi Decoder Based On Scarce State Transition And Variable Truncation Length.*

Digital Object Identifier 10.1109/VDAT.2007.373220, vol. 25-27, pages 1–4, April 2007.

[Liu 07] H. Liu, J.P. Diguet, C. Jego, M. Jézéquel & E. Boutillon. *Energy Efficient Turbo Decoder with Reduced State Metric Quantization.* IEEE sips 2007, Nov. 2007.

[Lopez-Vallejo 02] M. Lopez-Vallejo, S.A. Mujtaba & Inkyu Lee. *A low-power architecture for maximum a posteriori decoding.* IEEE, Signals, Systems and Computers, vol. 2, pages 47 – 51, Nov. 2002.

[MacKay 99] D. MacKay. *Good error-correcting codes based on very sparse matrices.* Information Theory, IEEE Transactions, vol. 45, pages 399–431, 1999.

[Maessen 00] F. Maessen, L. Van der Perre, F. Willems, B. Gyselinckx, F. Catthoor & M. Engels. *Memory power reduction for the high-speed implementation of turbo codes.* SCVT-200, pages 94 – 102, Oct. 2000.

[Mansour 03] M. Mansour & N. Shanbhag. *VLSI architectures for SISO-APP decoders.* IEEE Trans. VLSI, vol. 11, no. 4, pages 627–650, Aug. 2003.

[Masera 99] G. Masera, G. Piccinini, M. R. Roch & M. Zamboni. *VLSI architectures for turbo codes.* IEEE. Trans. VLSI Syst., vol. 7, pages 369–379, Sep 1999.

[Massey 69] J. L. Massey. *Shift-register synthesis and BCH decoding.* IEEE Trans. Inform. Theory, vol. IT, pages 122–127, january 1969.

[Montorsi 01] G. Montorsi & S. Benedetto. *Design of fixed-point iterative decoders for concatenated codes with interleavers.* IEEE Journal on Selected Areas in Communications, vol. 19, pages 871–882, May 2001.

[Moore 65] G. Moore. *Cramming more components onto integrated circuits.* Electronics, vol. 38, 1965.

[Muller 07] O. Muller. *Architectures multiprocesseurs monopuces génériques pour turbo-communications haut-débit.* PH.D. dissertationn, Université de Bretagne-Sud, 2007.

[Pietrobon 96] S. S. Pietrobon. *Efficient implementation of continuous MAP decoders and a new synchronization technique for turbo decoders.* Proc. Int., Symp. Information Theory and Its Applications, pages 586–589, Sep. 1996.

[Pietrobon 98] S. S. Pietrobon. *Implementation and performance of a turbo/MAP decoder.* Int. J. of Satellite Commun., vol. 16, pages 23–46, Jan.-Feb. 1998.

[Pyndiah 98] R.M. Pyndiah. *Near-optimum decoding of product codes : block turbo codes.* Communications, IEEE Transactions on, vol. 46, no. 8, pages 1003–1010, Aug. 1998.

[Raouafi 99] F. Raouafi, A. Dingninou & C. Berrou. *Saving memory in turbo-decoders using the Max-Log-MAP algorithm.* Turbo Codes in Digital Broadcasting, IEEE Colloquium, pages 14/1–14/4, 22 Nov. 1999.

[Robertson 95] P. Robertson, E. Villebrun & P. Hoeher. *A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain.* Proc. ICC 1995, vol. 2, pages 1009–1013, Jun. 1995.

[Sakurai 03] T. Sakurai. *Perspectives on Power-Aware electronics.* IEEE International Solid-State Circuits Conference, 2003.

[Schurgers 01] C. Schurgers, F. Catthoor & M. Engels. *Memory optimisation of MAP turbo decoder algorithms.* IEEE trans. VLSI system, vol. 9, no. 2, April 2001.

[Shannon 48] C. E. Shannon. *A Mathematical Theory of Communication.* Bell System Technical Journal, vol. 27, pages 379–423 and 623–656, Jul. 1948.

[Shockley 48] W. Shockley, J. Bardeen & W.H. Brattain. Electronic theory of the transistor. Science, 1948.

[Tarjan 06] D. Tarjan, S. Thoziyoor & N.P. Jouppi. *CACTI 4.0 : A technical report.* HP Research Laboratories Palo Alto, June 2006.

[Tijms 04] H. Tijms. *Understanding Probability : Chance Rules in Everyday Life.* Cambridge : Cambridge University Press, 2004.

[Tsai 05] T. Tsai, C. Lin & A. Wu. *A memory-reduced log-MAP kernel for turbo decoder.* IEEE ISCAS 2005, vol. 2, pages 1032 – 1035, May 2005.

[Viterbi 67] A. J. Viterbi. *Error bounds for convolutional codes and an asymptotically optimum decoding algorithm.* IEEE Transactions on Information Theory 13(2), vol. 4, pages 260–269, Apr. 1967.

[von Neumann 66] J. von Neumann. Theory of self-reproducing automata. University of Illinois Press, 1966.

[Wang 00] Y. Wang, C. Tsui & R. S. Cheng. *A Lower Power VLSI Architecture of SOVA-Based Turbo-code Decoder Using Scarce State Transition Scheme.* ISCAS 2000, pages 283–286, May 2000.

[Weiss 98] C. Weiss, C. Bettstetter, S. Riedel & D.J. Costello. *Turbo decoding with tail-biting trellises.* Proc. Signals, Systems and Electronics, URSI Int'l Symposium, pages 343–348, Sept.-Oct. 1998.

[Weiss 01] C. Weiss, C. Bettstetter & S. Riedel. *Code construction and decoding of parallel concatenated tail-biting codes.* IEEE. Trans. Info. Theory, vol. 47, Issue 1, pages 366–386, Jan. 2001.

[Welch 86] L. Welch & E. Berlekamp. *Error correction for algebraic block codes.* US Patent 4,633,470, 1986.

[Wilton 94] Steven J.E. Wilton & Norman P. Jouppi. *An Enhanced Access and Cycle Time Model for On-Chip Caches.* Western Research Laboratory (WRL) Research Report 93/5, vol. ACM Press, July 1994.

[Wu 00] Y. Wu, J. W. Ebel & B. D. Woerner. *Forward computation of backward path metrics for MAP decoder.* IEEE VTC 2000, 2000.

[Xilinx 08] Xilinx. *Xilinx University Program Virtex-II Pro Development System : Hardware Reference Manual.* UG069 (v1.1), April 9 2008.