# IMPLANTATION D'UN ALGORITHME DE DECODAGE

# A SORTIES PONDEREES MAP DANS UN FPGA

**Didier Le Ruyet, Christophe Alexandre, Christophe Le, Han Vu Thien**

*Conservatoire National des Arts et Métiers*
*Laboratoire Signaux et Systèmes*
*292 rue Saint Martin*
*75141 PARIS Cedex 03, France*
*téléphone : +33(0) 1 40 27 27 98*
*fax : +33(0) 1 40 27 29 94*
*leruyet@cnam.fr*

## 1. Introduction

L'algorithme MAP introduit par Bahl [1] réalise le décodage des codes convolutifs. Par rapport à l'algorithme de Viterbi qui détermine la séquence la plus probable, l'algorithme MAP associe à chaque bit décodé une estimation de la fiabilité de cette décision. Cet algorithme est utilisé en particulier pour le décodage des codes convolutifs concaténés en parallèle ou turbo codes. Dans [2], Pietrobon a montré qu'il était possible d'implanter un décodeur MAP dans plusieurs circuits logiques programmables. L'objectif de cet article est de présenter la conception et l'implantation d'un décodeur MAP dans un circuit programmable basé sur la méthode de décodage dite à double fenêtre glissante proposée par Viterbi [3]. Dans cet article, après avoir rappelé l'algorithme de décodage pondéré MAP, nous proposerons une architecture matérielle dans un circuit logique programmable. Finalement nous présenterons les différentes phases de validation et les performances obtenues.

## 2. L'algorithme de décodage à sorties pondérées MAP

Pour simplifier la description de l'algorithme MAP, nous allons considérer un codeur convolutif systématique de rendement ½ composé de M mémoires ($2^M$ états internes). Soit $u_k$ le k-ième bit d'information et $c_k$ le bit de redondance correspondant. A la sortie du canal de transmission, on reçoit la séquence suivante : $R_1^N = (R_1,...,R_k,...,R_N)$ avec $R_k = (x_k, y_k)$ où $x_k$ et $y_k$ sont les composantes reçues à l'instant $k$ associées respectivement aux bits $u_k$ et $c_k$.

Le décodeur MAP calcule le rapport des probabilités conditionnelles:

$$\Lambda(u_k) = \frac{P(u_K = 1 / R_1^N)}{P(u_K = 0 / R_1^N)}$$

Pour calculer $\Lambda(u_k)$, on introduit les densités de probabilités conditionnelles pour l'état $m$ aller $a_k(m) = P(S_K = m / R_1^k)$ et retour $b_k(m) = P(S_K = m / R_k^N)$. On peut montrer que :

$$\Lambda(u_k) = \frac{\sum_m a_{k-1}(m) g_k^1(m) b_k(f(1,m))}{\sum_m a_{k-1}(m) g_k^0(m) b_k(f(0,m))}$$

$g_k^i(m)$ est la métrique de branche à l'instant $k$ correspondant au passage de l'état $m$ à l'état $f(i,m)$ lorsque le bit d'information $u_k = i$.

$a_k(m)$ et $b_k(m)$ sont calculées de façon itérative :

$$a_k(m) = h_a \sum_i a_{k-1}(b(i,m)) g_k^i(b(i,m)) \quad \text{avec } a_0(0) = 1 \text{ et } a_0(m) = 0 \ \ \forall (m \neq 0)$$

$$b_{k-1}(m) = h_b \sum_i b_k(f(i,m)) g_k^i(m) \quad \text{avec } b_N(0) = 1 \text{ et } b_N(m) = 0 \ \ \forall (m \neq 0)$$

$f(i,m)$ est l'état d'arrivée correspondant à un état de départ $m$ lorsque le bit d'information $u_k = i$.

$b(i,m)$ est l'état de départ correspondant à un état d'arrivée $m$ lorsque le bit d'information $u_k = i$.

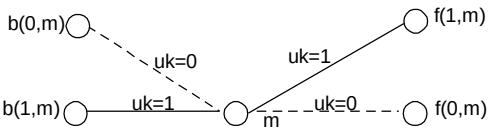


Figure 1. représentation du treillis

A la place de $\Lambda(u_k)$, on préfère calculer $L(u_k) = \log \Lambda(u_k)$ qui a l'avantage de remplacer les multiplications par des additions. En posant $A_k(m) = \log a_k(m)$, $B_k(m) = \log b_k(m)$ et $D_k^i(m) = \log g_k^i(m)$, on obtient finalement

$$L(u_k) = \max_m{}^*(A_{k-1}(m) + D_k^1(m) + B_k(f(1,m))) - \max_m{}^*(A_{k-1}(m) + D_k^0(m) + B_k(f(0,m)))$$

avec $A_k(m) = \max_i{}^*(A_{k-1}(b(i,m)) + D_k^i(b(i,m)))$,

$B_{k-1}(m) = \max_i{}^*(B_k(f(i,m)) + D_k^i(m))$,

et $\max^*(a_1, a_2, ..., a_j) = \ln\left(\exp(a_1) + \exp(a_2) + ... + \exp(a_j)\right)$.

Les $D_k^i(m)$ et $A_k(m)$ sont calculés à partir de k=1 jusqu'à k=N puis stockés en mémoire. L'algorithme doit attendre d'avoir reçu toute la séquence pour commencer le calcul de $B_k(m)$ puis $L(u_k)$.

## 3 Architecture matérielle

Dans [3], Viterbi a proposé une technique dite de la double fenêtre glissante permettant de réduire considérablement la taille de la mémoire nécessaire au décodage. Cette technique requière une taille mémoire indépendante de la longueur $N$ du bloc. Elle nécessite un calculateur pour $A_k(m)$ (calculateur de métrique d'état aller CMEA) et deux calculateurs pour $B_k(m)$ (calculateurs de métrique d'état retour CMER1 et CMER2). Pour le calcul $B_k(m)$, on exploite le fait que l'on peut commencer le calcul à n'importe quel instant ; les

premières valeurs calculées seront inexploitables mais après un certain temps L (L> 6M), les valeurs $B_k(m)$ sont aussi fiables que si nous avions effectué la récursion à partir de l'instant final. La figure 2 décrit le séquencement des différents calculateurs.

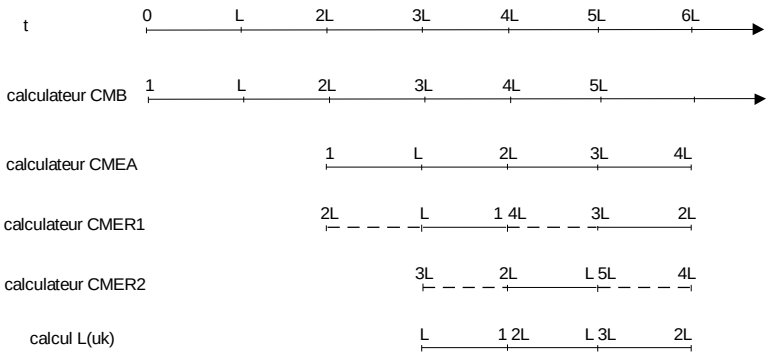


Figure 2 : séquencement des différents calculateurs.

Dans la figure 3, nous proposons un synoptique général du décodeur MAP utilisant la technique de la double fenêtre glissante. Les métriques de branche sont calculées à partir de t=1 puis mémorisées dans M1 et M2 $((L+2L).2^n$ cases mémoires). Les métriques d'état $A_k(m)$ pour k=1 à 2L sont calculées par le calculateur aller CMEA de t=2L à 4L puis stockées dans M3 $(2L.2^M$ cases mémoires). A partir de t=2L, le calculateur retour CMER1 effectue sa récursion sur les échantillons reçus de k=2L à 1. Cependant, seuls les $B_k(m)$ pour k=L à 1 sont utilisables pour le calcul de $L(u_k)$. On peut donc de t=3L à 4L calculer $L(u_k)$ pour k=L à 1. Le calculateur CMER2 fonctionne sur le même principe mais commence sa récursion à t=2L. La mémoire M4 permet de réordonner les informations $L(u_k)$.

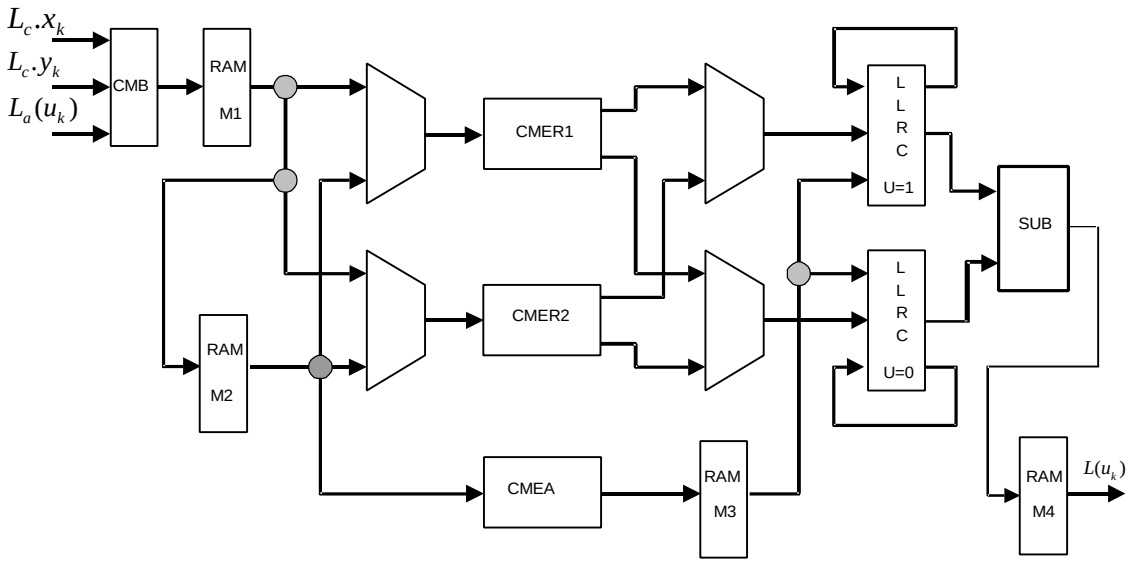


Figure 3 : synoptique général du décodeur MAP.

La fonction $\max^*(x_n)$ doit être simplifiée en vue d'une implantation : dans [4], il a été proposé d'approximer la fonction $\max^*(a_1, a_2)$ par : $\max^*(a_1, a_2) \approx \max(a_1, a_2) + f(a-b)$ avec $f(a-b) = cste$ si $(a-b) < k$ et $f(a-b) = 0$ sinon. C'est cette solution que nous avons retenue car elle est peu consommatrice de ressources et sans dégradation significative des performances. Pour le calcul de $L(u_k)$ on doit réaliser l'opération $\max^*$ sur $2^M$ paramètres. On peut montrer que : $\max^*(a_1, a_2, ..., a_j) = \max^*(\max^*(a_1, a_2, ..., a_{j-1}), a_j)$. Cette opération sera donc effectuée de façon itérative. Pour que la quantification ne dégrade pas les performances par rapport à un décodeur idéal (inférieure à 0.1dB) , nous avons choisi de quantifier les échantillons $x_k$ et $y_k$ sur 6 bits et $L(u_k)$ sur 8 bits et d'effectuer les calculs de $D_k^i(m)$, $A_k(m)$ et $B_k(m)$ sur 8 bits.

## 4 Aspects implantation

Dans notre application, nous avons choisi M=2 (4 états internes) et fixé L=32. Compte tenu de la capacité mémoire nécessaire à l'implantation de cet algorithme, notre choix s'est porté sur un circuit programmable intégrant de la mémoire dédiée. Nous avons choisi d'utiliser le circuit XCV100 de la famille Virtex de Xilinx qui comprend 10 blocs mémoires double port de 4192 bits. 6 blocs mémoires ont été utilisés (4 pour M1 et M2, 1 pour M3 et 1 pour M4). La conception du circuit a été entièrement réalisée en VHDL. Notre attention s'est particulièrement portée sur l'exploitation optimale des caractéristiques de ce circuit.
Tous les calculs doivent être effectués en moins d'une période d'horloge symbole. Pour implanter cet algorithme, nous avons utilisé une structure pipeline ; une horloge interne CLK cadence les différentes opérations. Un compromis vitesse/ complexité nous a amené au séquencement présenté figure 4.

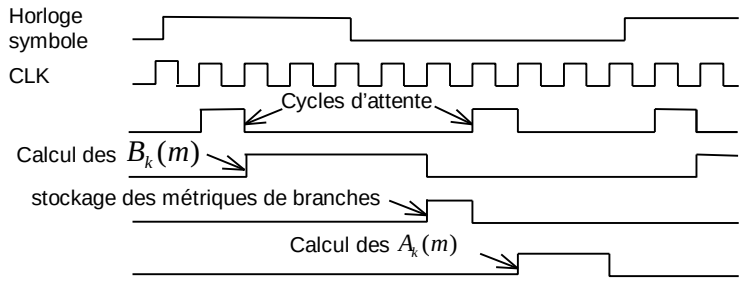


figure 4 :séquencement des différentes opérations.

10 périodes sont nécessaire pour réaliser toutes les opérations. Certaines opérations comme le calcul des métriques de branche sont effectuées en même temps que le calcul des $B_k(m)$.
La limitation principale de cette architecture est liée à l'implantation de la fonction $\max^*$ qui est effectuée de façon itérative pour le calcul de $L(u_k)$. Les ressources utilisées par les différentes fonctions du décodeur sont présentées table 1.

| | Bascules D | Générateurs de fonction(4 variables) | Slices (2 générateurs + 2 Bascules D) | Bloc mémoire 4 Kbits |
|---|---|---|---|---|
| divers | 76 | 98 | 50 | 1 |
| Calcul des métriques de branche | 21 | 152 | 75 | 4 |
| Calcul des $A_k(m)$ | 119 | 441 | 223 | 1 |
| Calcul des $B_k(m)$ | 177 | 601 | 305 | 0 |
| Calcul de $L(u_k)$ | 64 | 193 | 95 | 0 |
| total | 457 (19%) | 1485 (62%) | 748 (62%) | 6 (60 %) |

Table1 : ressources nécessaires aux différentes opérations.

Le taux d'occupation est donc de 62 %.

## 5 Tests et performances

Afin de valider la conception, nous avons réalisé un logiciel de simulation d'un ensemble codeur convolutif et décodeur MAP associé en langage C. Le décodeur MAP de ce programme de référence comporte les mêmes fonctions que le décodeur à réaliser (quantification, fenêtre glissante, fonction max,…). Nous avons vérifié à chaque étape de la réalisation, la cohérence des résultats obtenus en sortie pour une même séquence d'entrée.
Le circuit ainsi réalisé a été validé avec $f_{CLK}$ = 33MHz. Dans cette configuration, le débit maximum est égal à 3,3 Msymboles/sec. Ces performances sont légèrement supérieures à celles présentées dans [5] à complexité comparable.

## 6 Conclusion

Dans cet article, nous avons présenté une implantation d'un décodeur MAP dans un circuit programmable. Les premiers résultats obtenus montrent qu'il est tout à fait possible de réaliser un décodeur MAP 4 états fonctionnant à 3,3 Msymboles/sec. Cependant, pour un nombre d'états plus élevés (8 ou 16), l'implantation de la fonction $\max^*$ limite les performances. Une amélioration des performances implique de réduire le temps nécessaire au calcul de la fonction $\max^*$.

## Références


[1] L. Bahl, J. Cocke, F. Jelinek and J. Raviv, « Optimal decoding of linear codes for minimizing symbol error rate », *IEEE Trans. Inform. Theory*, vol. IT-20, pp. 284-287, Mar. 1974.
[2] S. S. Pietrobon, Implementation and Performance of a Turbo/MAP decoder », *Int. Journal of Satellite Communication*, vol. 16, n°1, pp. 23-46, Jan. 1998.
[3] A. J. Viterbi, « An intuitive justification and a simplified implementation of the MAP Decoder for convolutional codes », *IEEE Sel. Areas in Comm.*, vol. 16, pp. 260-264, Feb. 1998.
[4] W. J. Gross and P. G. Gulak, Simplified MAP algorithm suitable for implementation of turbo decoders », *Electronics Letters.*, vol. 34, pp. 1577-1578, 6th Aug. 1998.
[5] Small World Communications, « MAP04B Data Sheet », Oct. 1999, www.sworld.com.au.