CASTAGNINO Guillaume
Dossier de T.I.P.E.

La Modulation par Impulsion et Codage en téléphonie

MP* Lycée Thiers, Marseille. 2001-2002

LA MODULATION PAR IMPULSION ET CODAGE EN TELEPHONIE

INTRODUCTION

La Modulation par Impulsion et Codage (MIC) est l'opération qui décompose un signal analogique, ici en l'occurrence un signal de paroles, en une série d'impulsions électriques formant un signal numérique. Elle se compose de trois étapes essentielles : l'échantillonnage, la quantification et le transcodage, ainsi que de leur réciproque. On n'étudiera pas ici l'échantillonnage du signal, qui pourrait faire à lui seul l'objet d'un autre dossier ; on se limitera donc à l'étude de la quantification et du transcodage ainsi qu'à la recomposition du signal analogique.

La transmission numérique est très intéressante dans le domaine des télécommunications, car elle permet de propager un signal sur de très longues distances sans aucune altération. En effet, quel que soit le support (fibre optique, câble coaxial…), la dispersion électrique et l'apparition de bruit sont inévitables. Il devient alors nécessaire de régénérer le signal afin de le propager jusqu'à son but sans modifications rédhibitoires. Or, cette régénération est impossible sur un signal analogique, qui se disperse irrémédiablement dans le temps, puisqu'on ne connaît aucune information sur la tension d'origine. Par contre, dans la transmission numérique des données, les caractéristiques essentielles du signal sont connues : transition de niveaux à des instants synchrones avec l'horloge, tension constante en dehors des transitions. Le signal numérique altéré conserve alors suffisamment d'informations pour être reconstitué à l'identique du signal produit par la source, avec pour seul inconvénient un infime retard, dû à la régénération du signal, qui n'excède pas la période d'un bit (de l'ordre de 488 ns dans le réseau téléphonique local).

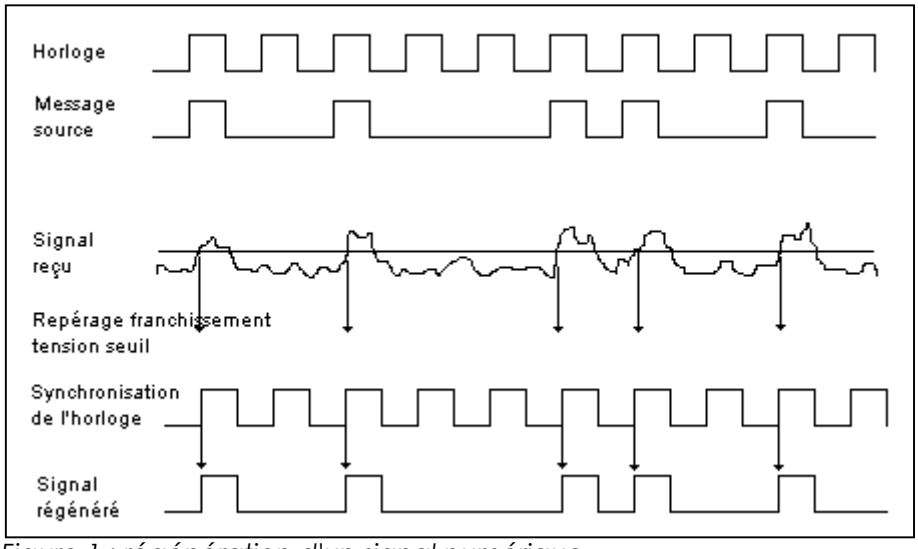


Figure 1 : régénération d'un signal numérique.

Un autre avantage de la transmission numérique est la détection d'erreurs. En effet, sous l'influence d'un fort bruit sur la ligne (dû principalement aux perturbations électriques diverses), il se peut qu'une valeur soit modifiée et introduise alors une erreur dans le signal. Mais, il existe des codages numériques qui permettent de détecter les erreurs et de les éliminer. La transmission en est d'autant plus fiable.

Un dernier point fort de la transmission numérique est de pouvoir appliquer des méthodes de compression, chose quasiment inapplicable à un signal analogique.

Il existe plusieurs types de matériels MIC. On les appelle des Terminaux Numérique d'Extrémité (TNE). Ils utilisent plusieurs normes qui ont évolué au fil des progrès techniques, notamment de la miniaturisation des composants. Ces normes sont le TN0, $TN1_a$, $TN1_b$, et actuellement, le TN1-2G (deuxième génération).

Il est à noter que toutes les opérations pratiquées dans le MIC sont soumises à des contraintes temporelles, dues au débit imposé et à la bande passante choisie. Ces valeurs sont communes à toutes les normes :

– La bande passante du téléphone est 300 à 3400 Hz, ce qui permet (théorème de Shannon, Nyquist) une fréquence d'échantillonnage Fe de 8000 Hz, soit une période Te = 125 µs.

– La transmission locale se fait sur une liaison multiplexée à 32 voies (c'est à dire 32 voies "mélangées" circulant en même temps dans le même fil). Comme il faut transmettre une donnée (un octet) par voie toutes les 125 µs, chaque voie bénéficie de 3,9 µs toutes les 125 µs pour transmettre un octet, la période d'un bit est donc Tb = 488 ns.

Les systèmes électriques sont par conséquent soumis à une contrainte de très grande rapidité, ce qui, nous le verrons, élimine des solutions qui pourtant s'avèrent plus simples à mettre en œuvre et moins coûteuses.

1). LA QUANTIFICATION

1.1). Principe.

On suppose au préalable que l'on dispose d'un signal de parole analogique échantillonné à la fréquence de 8000 Hz, c'est à dire que l'on a une série de valeurs prises toutes les 125 µs sur le signal. Ce sont les points noirs sur la courbe de la figure 2.

Le but de la quantification est de limiter le nombre de valeurs que peut prendre le signal. C'est en effet indispensable, car le nombre de valeurs que peut prendre chaque échantillon doit être fini, puisqu'on utilise un traitement numérique, incompatible avec l'infini. Il faut donc se donner un nombre de bits maximum pour chaque valeur, compatible avec les données à transmettre et le débit voulu. Dans le système MIC, le codage se fera sur 8 bits, ce qui correspond au débit idéal théorique de la ligne : 64 kbits.s^{-1}. On va ainsi constituer 256 plages de valeurs (2^8), étagées entre +Vsat et -Vsat, auxquelles on associera le même code binaire. Par conséquent, tous les échantillons ayant une amplitude comprise dans une même plage seront codées de la même manière. Cette mesure est celle du niveau inférieur de la plage (tension seuil).

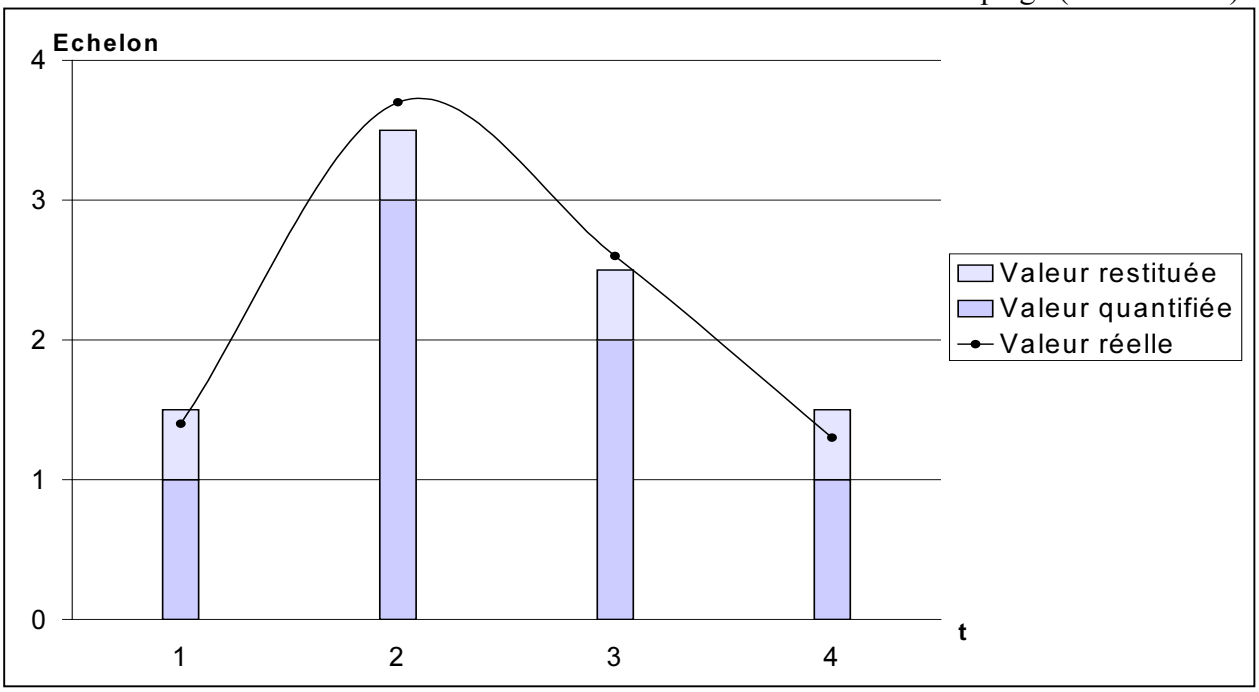


Figure 2 : Le principe de la quantification.

1.1.1). Un premier problème.

Cependant, on voit rapidement apparaître un premier problème : lors de la restitution du signal, un bruit important est produit. Il est matérialisé par la différence entre la valeur retenue pour le codage et la valeur effectivement échantillonnée. Ce bruit est nommé bruit de quantification. S'il est trop élevé, le signal d'information (la parole) sera inaudible. Pour le réduire, on peut augmenter le nombre de niveaux, mais on se limite en téléphonie à 256 échelons (codage sur 8 bits), soit 128 par polarité, puisque la ligne locale de l'utilisateur ne supporte pas des débits plus élevés que ceux actuellement utilisés en RTC (réseau téléphonique commuté), ce dernier étant imposé par des normes internationales.

On améliore facilement ce défaut en effectuant ce que l'on appelle une quantification linéaire. Au lieu de restituer la valeur de seuil de la plage, on augmente cette valeur d'un demi échelon (voir figure 2). Le bruit de quantification est alors limité au plus à E/2, où E est l'intervalle de tension entre la valeur seuil et la valeur maximale d'un échelon donné.

Le rapport signal/bruit caractérise le degré d'audibilité de l'information. On rappelle que plus il est élevé, meilleure est la restitution sonore. Pour référence, le rapport S/B d'un bon matériel Hi-Fi est d'environ 90 à 95 dB, celui d'un poste radio FM est d'environ 70 dB.

Observons maintenant le rapport signal/bruit si on utilise la quantification linéaire. Dans le pire des cas (bruit maximal), on a :

– Pour le premier échelon : $\frac{S}{B} = 20 \cdot \log \frac{E}{E/2} = 6dB$.

– Pour le nième échelon : $\frac{S}{B} = 20 \cdot \log \frac{n \cdot E}{E/2} = 20 \cdot \log n + 6dB$

On constate donc ici que S/B n'est pas constant, il augmente avec la tension échantillonnée, et qu'il est trop faible pour les signaux de petite valeur. Ceci ne favorise pas l'intelligibilité de l'information, car il existe un déséquilibre entre les faibles et les forts niveaux, ces derniers ressortant exagérément. De plus la trop grande faiblesse du rapport signal/bruit aux faibles niveaux entraîne une totale inintelligibilité de l'information à ces niveaux. Il apparaît par conséquent un deuxième problème à résoudre, sinon la communication sera impraticable.

1.1.2). Un second problème.

Il est impératif de corriger ce défaut, en trouvant une fonction de quantification qui, toujours étagée sur 256 échelons, va permettre de rendre le rapport signal/bruit constant. Or, augmenter le rapport signal/bruit aux faibles tensions revient à augmenter la précision de quantification aux faibles niveaux, puisqu'il faut réduire l'erreur. De plus, le rapport signal/bruit étant très important pour les fortes tensions, on peut se permettre de le réduire pour les niveaux hauts. On va donc utiliser une courbe à allure logarithmique, qui offre un pas de quantification réduit aux faibles tensions, et plus important aux forts niveaux. Pour cela, on a conçu la loi A, illustrée figure 3. Elle est utilisée dans tous les systèmes MIC.

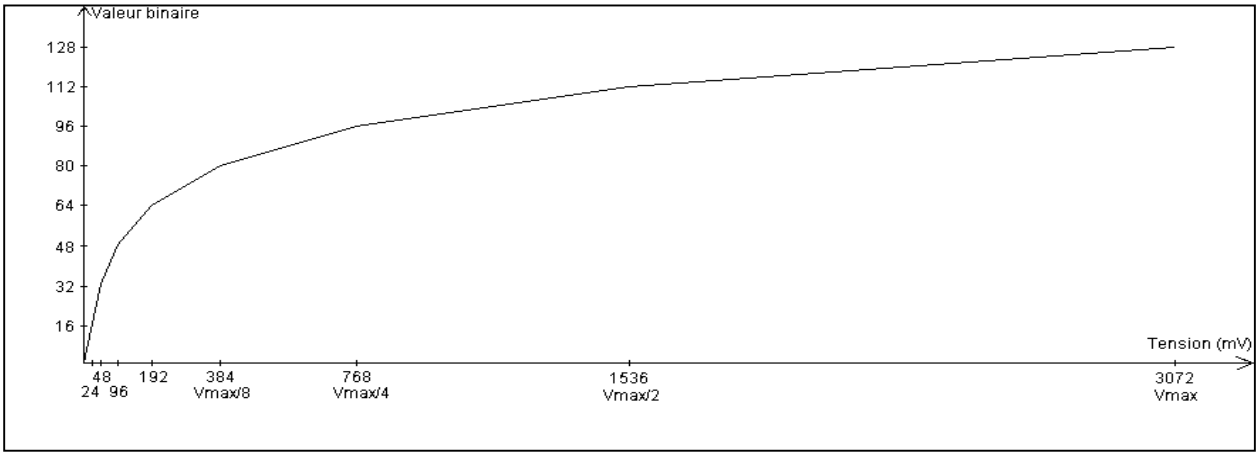


Figure 3 : la loi pseudo logarithmique.

La courbe est composée de 8 segments de droite, sur lesquels on pratique une quantification linéaire à 16 plages. Entre 0 et 24 mV, un échelon vaut donc 1,5 mV ; et entre 1536 et 3072 mV, il vaut 96 mV.

Effectivement, le défaut que possédait la quantification linéaire semble corrigé. Si on mesure le rapport signal/bruit, il est quasiment constant sauf pour les toutes premières plages du premier segment.

On le calcule en dB au niveau $n \in \{-128;127\}$ avec la formule : $\frac{S}{B}(n) = 20 \cdot \log \frac{u(n)}{E(n)/2}$,

où E(n) est l'amplitude d'un échelon au niveau n, et u(n) la tension seuil du niveau n.

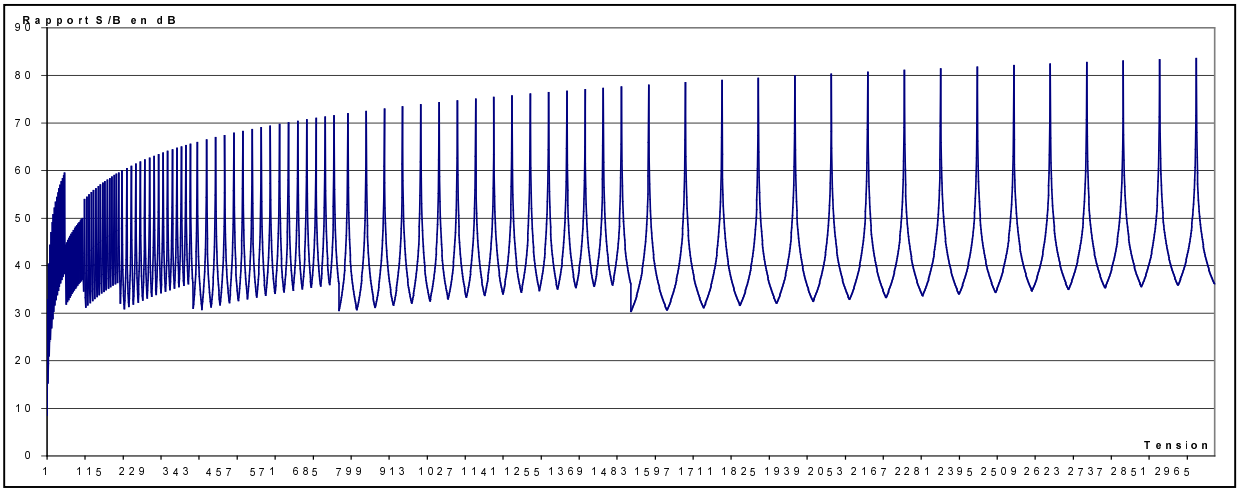


Figure 4 : le rapport signal/bruit du système MIC.

Le résultat est une meilleure intelligibilité de l'information contenue dans le signal, puisque les bas niveaux ne sont plus masqués par un bruit trop important et, le rapport signal/bruit étant à peu près constant, aucun niveau n'est démesurément favorisé par rapport aux autres. Il en résulte une meilleure homogénéité de la parole. C'est bien sûr cette méthode de quantification que nous allons retenir, puisque c'est celle utilisée couramment en télécommunications.

1.2). Mise en œuvre, principaux codeurs

1.2.1). Le codeur à pente.

Nous allons faire ici l'étude détaillée du codeur à pente. C'est un système très simple à mettre en œuvre, mais qui n'est plus utilisé en pratique, car trop lent en comparaison aux temps auquel doit se faire la conversion (quelques µs). Voici son schéma électrique :

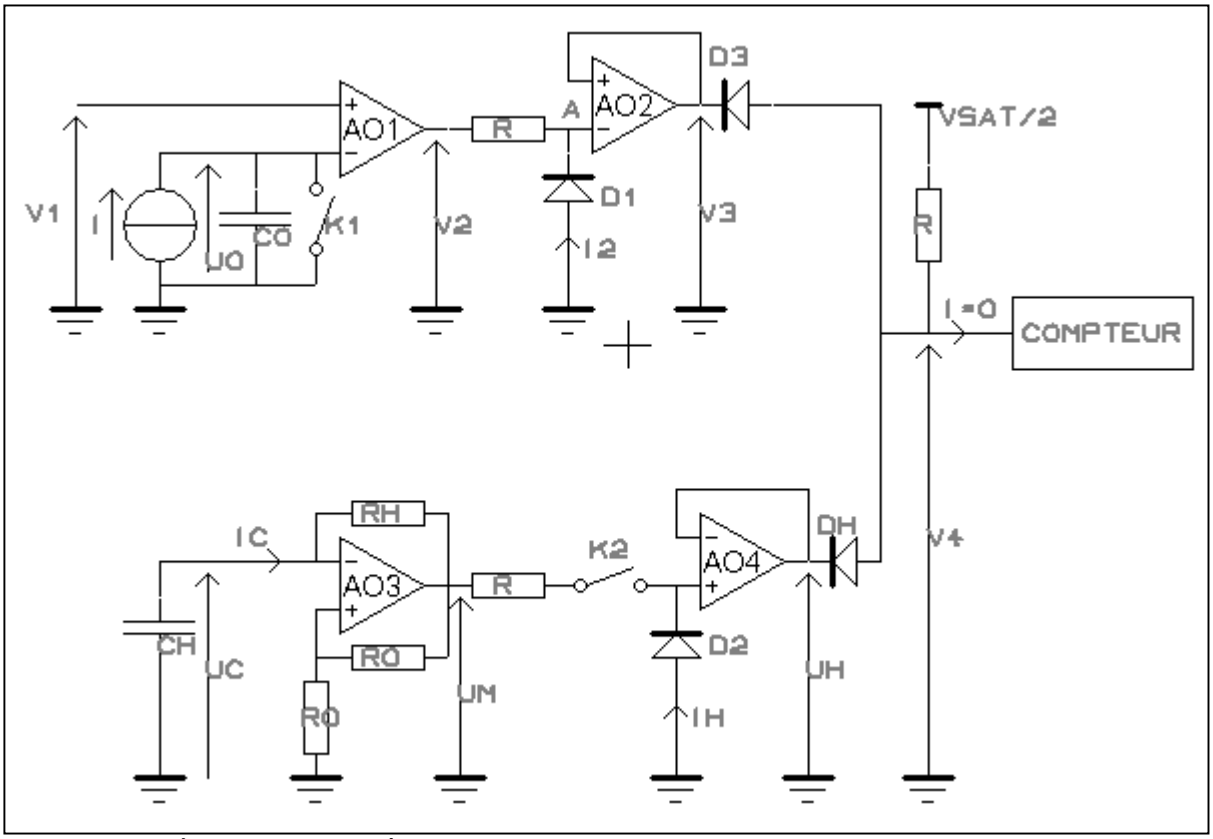


Figure 5 : Schéma du codeur à pente.

Le principe de ce codeur est de convertir la tension à quantifier en une tension de valeur constante pendant un temps proportionnel à la valeur de l'échantillon, et nulle ensuite (v3). On dénombre alors le nombre entier de périodes d'horloge immédiatement inférieur à la durée de cette tension (v4). Ceci permet de quantifier chaque valeur échantillonnée.

Ce système correspond à la norme TN0, qui utilise consécutivement 2 codeurs à pente. Le codage étant linéaire, 12 bits sont nécessaires pour coder les faibles niveaux avec suffisamment de précision. On pratique ensuite une compression pour retrouver la loi logarithmique sur 8 bits utilisée dans le système MIC. Le codage s'effectue en deux temps :

– Tout d'abord, on détermine sur quel échelon parmi 64 entre –Vmax et Vmax se trouve l'échantillon avec un premier codeur à pente (6 premiers bits).

– Dans un deuxième temps, on découpe l'échelon précédent en 64 et on cherche sur quel nouvel échelon se trouve l'échantillon avec un deuxième codeur à pente.

Pour ce codeur, les entrées sont, voir le schéma (figure 5), v1 est la tension à quantifier, uc est un signal d'horloge, à une fréquence connue.

a). Etude de $v2(t) = f(v1(t))$.

On connaît ici v1(t), qui est la tension à quantifier. Elle est maintenue constante pendant toute la durée de la quantification par un échantillonneur-bloqueur non étudié ici.

– A t = to$^-$, K1 est fermé, donc u0 = 0.

– A t = to$^+$, c'est le début de la conversion, on ouvre K1.

On a : $uo(t) = \int_{to}^{t} \frac{I}{Co} dt$, or I est constant (générateur de courant), d'où $uo(t) = \frac{I}{Co} \cdot (t - to) + A$, et A = 0 car on prend $uo(t = to) = 0$. Donc, $uo(t) = \frac{I}{Co} \cdot (t - to)$.

– L'AO1 est monté en comparateur, donc : $\begin{cases} v1(t) > uo(t) \Rightarrow v2(t) = Vsat \\ v1(t) < uo(t) \Rightarrow v2(t) = -Vsat \end{cases}$.

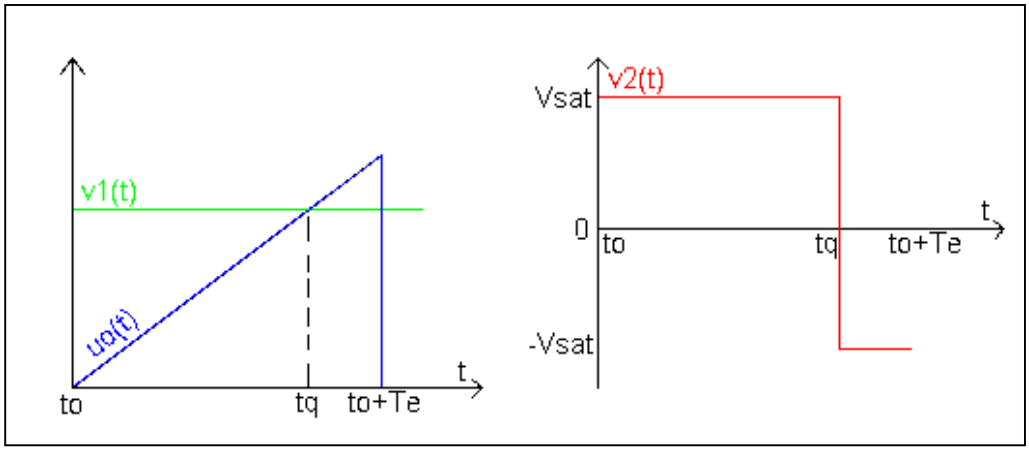


Figure 6 : illustration de v2(t) en fonction de v1(t).

b). Etude de $v3(t) = f(v2(t))$.

L'amplificateur opérationnel 2 est un suiveur.

– Hypothèse : D1 est passante, donc v3 = 0. Condition de validité : $i2 = -\frac{v2}{R} > 0 \Leftrightarrow v2 < 0$.

– Hypothèse : D1 bloquante, donc i2 = 0. Or, $i2 = \frac{uA - v2}{R} = 0$, donc uA = v2, d'où v2 = v3.

Condition de validité : -uA = v2 > 0.

– v3 est donc la partie positive de v2 : $\begin{cases} v2(t) > 0 \Rightarrow v3(t) = v2(t) \\ v2(t) < 0 \Rightarrow v3(t) = 0 \end{cases}$

c). Générateur du signal d'horloge uH.

uH est une tension en créneaux variant entre 0 et Vsat, d'une période TH bien déterminée. En vue d'un codage sur 12 bits, soit 11 bits par polarité, on va utiliser $Te = 2048 \cdot TH$.

d). Recherche de $v4(t) = f(v3(t), uH(t))$.

– Hypothèse : D3 et DH passantes, alors nécessairement, v3 = uH = v4 et $v4 < \frac{Vsat}{2}$. Or, v3 et vH sont à valeurs dans {0, Vsat}, donc v3 = uH = v4 = 0.

– Hypothèse : D3 et DH bloquées, alors le courant dans R est nul, donc $v4 = \frac{Vsat}{2}$. L'hypothèse est valide si v3 > v4 et uH > v4, donc v3 = uH = 0, $v4 = \frac{Vsat}{2}$.

– Hypothèse : D3 bloquée, DH passante, alors v4 = vH. Hypothèse valide si v3 > vH et $vH < \frac{Vsat}{2}$. Donc v4 = uH = 0, v3 = Vsat.

– Hypothèse : D3 passante, DH bloquée. Par symétrie, on peut conclure v3 = v4 = 0, vH = Vsat.

On peut récapituler ces résultats dans un tableau :

uH	v3	v4
0	0	0
0	Vsat	0
Vsat	0	0
Vsat	Vsat	Vsat/2

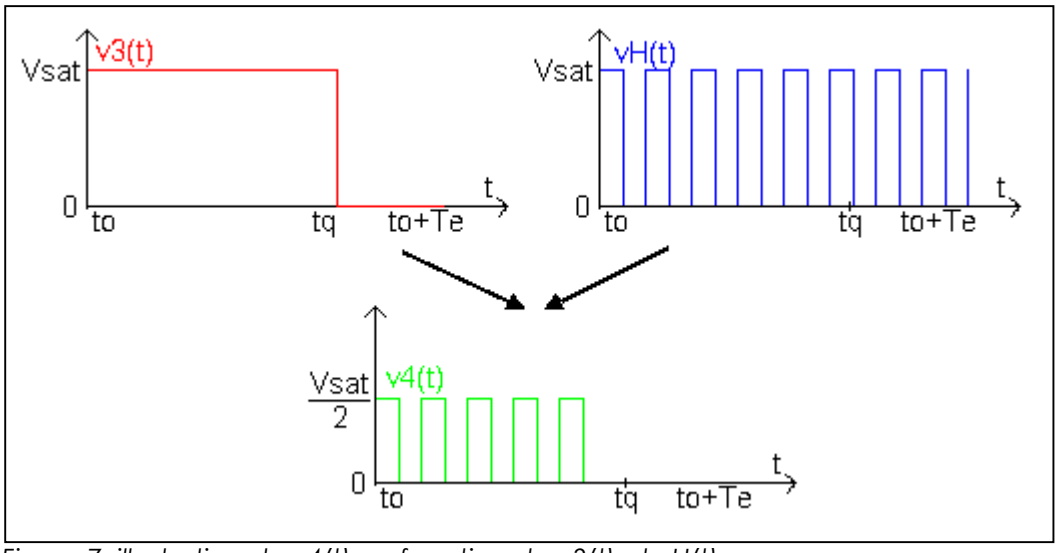


Figure 7, illustration de v4(t) en fonction de v3(t) et vH(t).

Finalement, le compteur binaire dénombre le nombre d'impulsions de v4(t) entre to et to + Te. Il ne reste plus qu'à transmettre ce code à un registre mémoire. On a alors obtenu les 6 premiers bits de la valeur à quantifier. On recommence la même opération pour les 6 derniers bits, déterminant ainsi la fin du code binaire. On pratique ensuite une compression de ces 12 bits pour respecter la loi logarithmique à 8 bits ; il ne reste plus qu'à transmettre la valeur à un registre à décalage qui délivre le code en ligne. La conversion est achevée.

1.2.2). Le codeur potentiométrique ou à résistances.

Ce type de codage est basé sur le principe de la dichotomie. En effet, on va comparer le tension à quantifier successivement à différentes tensions de référence, dont le choix est conditionné par le résultat des tests précédents. Chacun des résultats est stocké dans une zone mémoire ; à la fin des comparaisons, au nombre de 8, chacun des registres contient l'un des 8 bits de la valeur ainsi quantifiée. Il ne reste plus qu'alors à vider les registres l'un après l'autre avec une fréquence connue pour former le mot binaire.

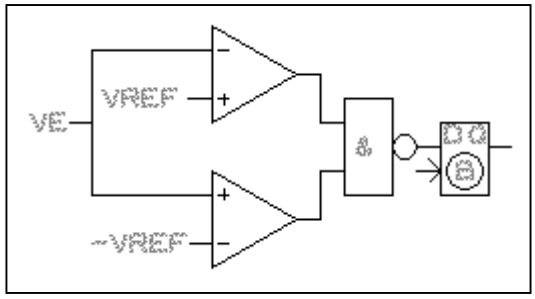
Ce codeur, utilisé dans les normes $TN1_a$ et $TN1_b$, effectue un codage potentiométrique type série. Dans ce système, les différentes tensions sont obtenues grâce à des résistances calibrées et à des injecteurs de courant pondéré (ICP). Le déclenchement des tensions de référence nécessaires pour les comparaisons et l'inscription des résultats dans les registres sont gérés par une logique de commande que nous n'étudierons pas ici.

Comme pour le codeur à pente, on doit maintenir la tension à quantifier constante pendant toute la durée du processus au moyen d'un échantillonneur-bloqueur. Il ne sera cependant pas nécessaire de pratiquer une compression après la quantification, car ce principe d'isolement de l'échelon correspondant à l'échelon par dichotomie se prête particulièrement bien à la compression directe. En effet, il est très facile de suivre dès le début du processus la loi logarithmique vue en 1.1.2).

Les opérations de codage se déroulent en 3 phases :– détermination du signe (1 bit).
– détermination du segment (3 bits).
– détermination du niveau dans le segment (4 bits).

On représente le mot binaire par la séquence A BCD EFGH, avec A le bit de signe, et B le bit de poids le plus fort, etc…

Le signe, connu par un comparateur à la masse, est mémorisé dans le registre A. On ne s'occupe ensuite plus que des valeurs positives pour la théorie, car la logique de commande inverse automatiquement les tensions en fonction du contenu du registre A.



La détermination du bit de poids fort B se fait simultanément avec la détermination du signe. C'est pourquoi, elle utilise le comparateur représenté ci-contre, qui compare des tensions en valeur absolue. La valeur de ce bit est très importante, car elle va conditionner le choix de toutes les tensions de référence. En effet, l'échantillon passe à l'entrée du circuit codeur par un amplificateur à gain commutable, qui peut prendre les valeurs 1 et 16. Suivant le contenu de B, on va commuter le gain de cet amplificateur. Initialement, le gain de l'amplificateur d'entrée est forcé à 16. On compare la tension à sa sortie à Vmax, ce qui revient à comparer l'échantillon (Ve) à Vmax/16. Si le résultat de cette comparaison est B = 1, c'est-à-dire $|Ve| > V\,\max/16$, le gain de l'amplificateur d'entrée est définitivement basculé à 1, dans le cas contraire, il est bloqué à 16. Cette multiplication de la tension d'entrée par 16 pour les faibles tensions permet de réduire le nombre d'injecteurs de courant pondéré (ICP) nécessaires pour effectuer toutes les comparaisons (8 au lieu de 12).

Pour les comparaisons des bits CD EFGH, on utilise le même comparateur que pour le bit A. Or, ce dernier compare les tensions à la masse. Par conséquent, on ne compare pas directement l'échantillon à la tension de référence. On fait la différence de ces deux tension (la logique de commande bascule le signe de la tension de référence pour qu'il soit opposé au signe de l'échantillon), et on compare le résultat à la masse. On obtient ainsi l'effet recherché.

Voici en détail la loi logarithmique de compression tenant compte de l'amplification des faibles tensions :

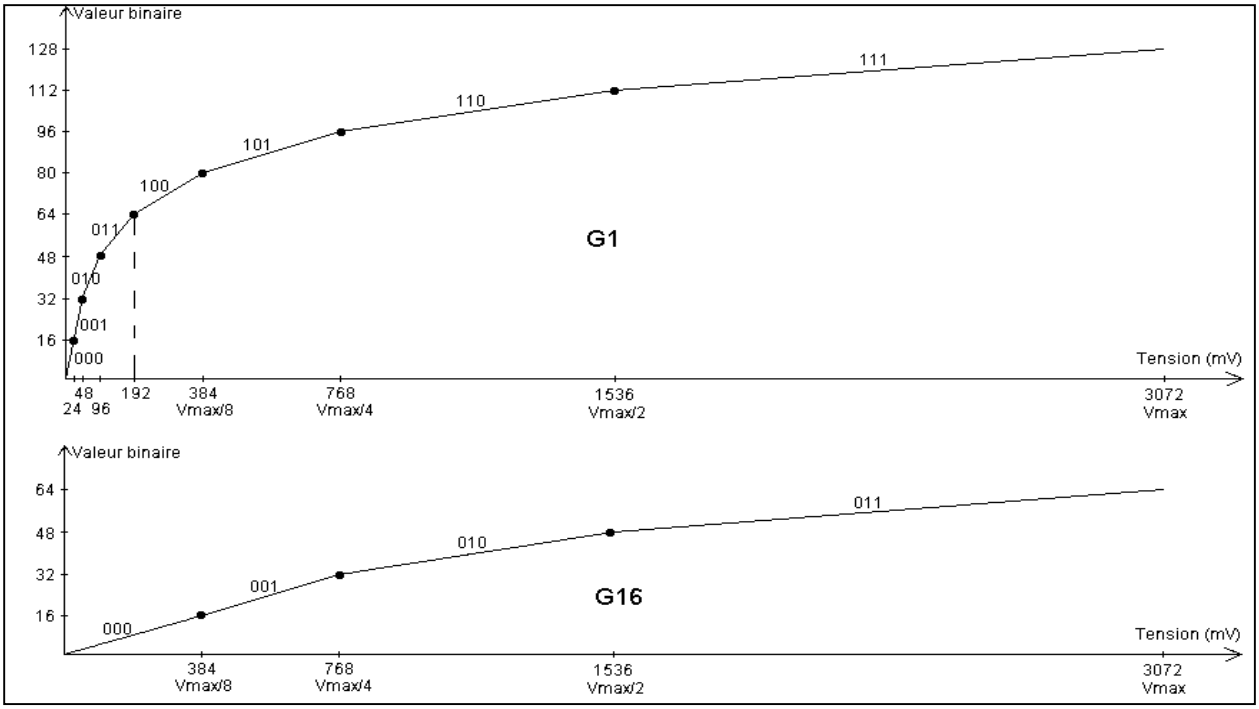


Figure 8 : Les courbes de codage suivant que le gain de l'amplificateur est à 1 ou 16 ; les séquences binaires correspondent à la séquence BCD.

Si l'amplificateur d'entrée est commuté à 1, on se basera sur la courbe G1 pour raisonner. Dans le cas contraire, on utilisera la courbe G16.

On peut remarquer que sur la courbe G1, la comparaison de Ve à Vmax/16 détermine bien le bit de poids fort, en regardant les séquences binaires de 3 bits figurant sur chaque segment, ce qui correspond bien à l'effet désiré.

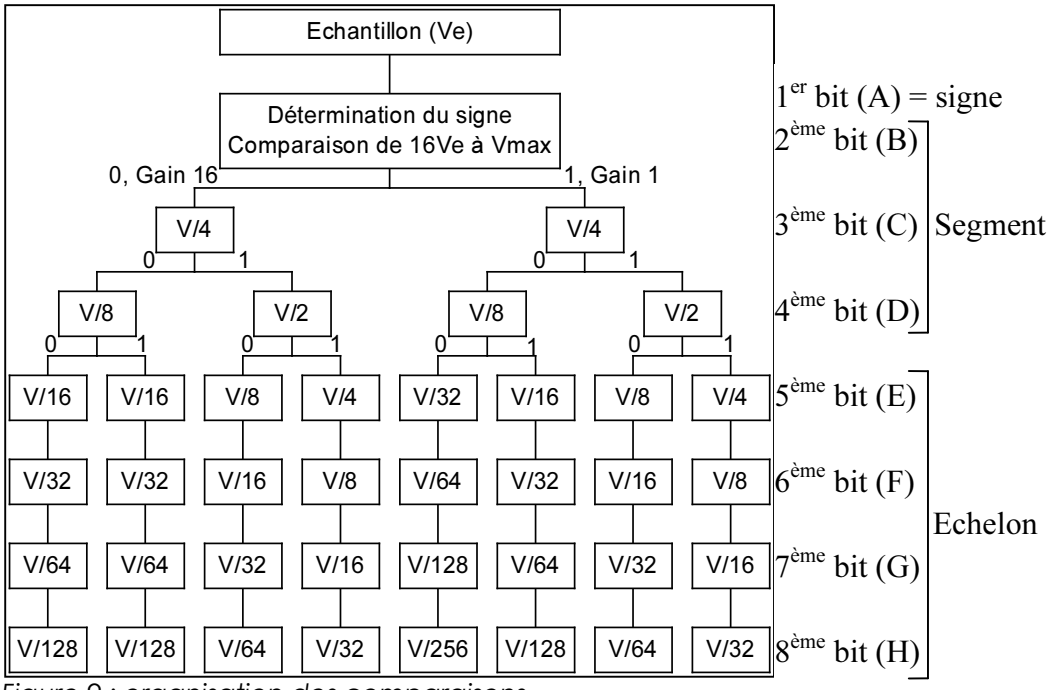


Figure 9 : organisation des comparaisons.

Le schéma ci-dessus montre l'organisation des tests suivant le résultat obtenu à chaque comparaison. Au stade où on en est, les bits A et B sont déjà déterminés. On va maintenant déterminer le bit C. Si on s'intéresse à la courbe figure 9, on s'aperçoit que dans les deux cas

(gain 1 et 16), la limite entre le bit C à 0 et à 1 est 768mV, soit exactement Vmax/4, ce qui correspond à la valeur donnée figure 9. Par conséquent, si la tension de l'échantillon est supérieure à 768 mV, on met C à 1, sinon à 0. On détermine le bit D de la même manière, en utilisant les valeurs indiquées figure 9 suivant le résultat des tests précédents.

Ainsi, par exemple, pour quantifier la valeur Ve = 900 mV, on a : A = 1 (positif), B = 1 (16Ve>V), C = 1 (Ve>V/4) et D = 0 (Ve<V/2). On a donc sur cet exemple le début du code binaire, A BCD = 1 110.

A ce stade, on connaît le signe de la tension à quantifier, et le segment sur lequel se trouve l'échantillon. On va donc maintenant déterminer quel échelon parmi les 16 du segment concerné correspond le mieux à l'échantillon. Jusqu'à présent, on a comparé la tension à quantifier à une tension de référence. Mais pour limiter le nombre d'ICP, on va ici plutôt sommer différentes tensions. Tout d'abord, on va utiliser, pour tout le temps de la détermination des 4 derniers bits, un tension piédestal du segment concerné. Elle correspond à la tension minimale du segment. Voici un tableau qui indique cette tension suivant la valeur de BCD (qui, rappelons-le, détermine le segment) :

	BCD	Valeur du piédestal du segment
G1	111	V/2
	110	V/4
	101	V/8
	100	V/16
G16	011	V/2
	010	V/4
	001	V/8
	000	0

A cette tension piédestal, on va sommer différentes tensions indiquées figure 9. A chaque fois, on compare cette somme de tensions à la tension à quantifier. Si la tension de l'échantillon est plus faible que la somme des tensions de référence, on retire la dernière tension ajoutée, c'est à dire qu'on éteint l'injecteur correspondant. La valeur du bit associé est mise à 0. Dans le cas contraire, on maintient l'injecteur et on met le bit à 1. On continue ensuite avec la tension suivante indiquée à la figure 9 en appliquant le même processus, et ainsi de suite.

On est sûr par cette méthode d'arriver à quantifier la tension, car on utilise une approximation par dichotomie, puisqu'on remarquera que les différentes tensions utilisées ici sont, pour la première, la moitié de l'amplitude du segment, la deuxième, le quart du segment, et ainsi de suite, la dernière correspondant à l'amplitude d'un échelon sur le segment (amplitude qui varie selon le segment).

Reprenons l'exemple précédent d'un échantillon de 900 mV. Le segment est BCD = 110, donc le piédestal est V/4.

– Le premier test est donc V/4 + V/8 = 1152 mV > 900 mV, donc E = 0, on désactive l'injecteur V/8.

– Deuxième test : V/4 + V/16 = 960 mV > 900 mV, donc F = 0, on désactive l'injecteur V/16.

– Troisième test : V/4 + V/32 = 864 mV < 900 mV, donc G = 1, on garde l'injecteur V/32 actif.

– Dernier test : V/4 + V/32 + V/64 = 912 mV > 900 mV, donc H = 0.

On possède maintenant le code de la valeur 900 mV, c'est à dire qu'on a quantifié cet échantillon : A BCD EFGH = 1 110 0010. Il ne reste plus qu'à vider les registres un à un, dans l'ordre, pour former une tension en créneaux, ie. un signal binaire.

Ce type de codeur présente l'inconvénient d'avoir huit opérations à effectuer les unes après les autre, imposant que la valeur de l'échantillon soit maintenue pendant toute la durée du codage. On a donc, grâce à la meilleure intégration des circuits, conçu une nouvelle génération de codeurs.

1.2.3). Le codeur série/parallèle sans maintien.

Il s'agit du codeur utilisé dans la norme du TN1-2G (deuxième génération). Comme pour le codeur utilisé dans le MIC $TN1_a$ et $TN1_b$, il utilise la loi logarithmique à 8 bits présentée en 1.1.2). La durée du codage est de 3,9 µs.

Le codage se fait en trois étapes :

1). La première étape est presque la même que pour le codeur potentiométrique (1.2.2).) : Pendant la durée des 4 premiers bits (1,95 µs), le gain de l'amplificateur à gain commutable est fixé à 8. Si $|Ve| < V\max/8$, le gain reste à 8. Sinon, il est commuté à 1. A la fin de cette phase, le bit de signe (A) et le gain sont mémorisés dans la logique de commande.

2). Pendant la durée des bits EF, on détermine la valeur du segment BCD. Pour cela, on applique l'échantillon Ve à un bloc de 16 comparateurs (Figure 10), dont les deux tensions de référence VH et VB sont fixées respectivement à 0 et Vmax ou -Vmax (selon le signe) par la logique de commande. Pour cette étape, tous les comparateurs ne sont pas utilisés : si le gain est à 1, on utilise les comparateurs 3, 5 et 9 (depuis la gauche), correspondant aux tensions V/8, V/4 et V/2 respectivement. Si le gain est à 8, on utilise les comparateurs 2, 3, 5 et 9, correspondant respectivement aux tensions V/16, V/8, V/4 et V/2.

Suivant le résultat donné par chacun des comparateurs, on connaît le segment sur lequel se trouve l'échantillon. Il ne reste plus qu'à convertir le numéro de ce segment en binaire. Les bits BCD sont mémorisés dans la logique de commande.

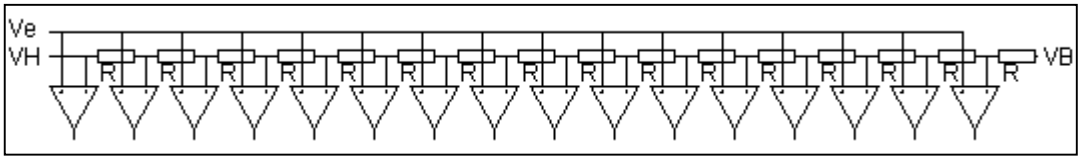


Figure 10 : la rampe de 16 comparateurs.

3). Dernière étape, pendant la durée des bits GH, on maintient la valeur de l'échantillon aux bornes de ce même bloc comparateur, et on met VH et VB respectivement aux tensions basses et hautes du segment identifié précédemment. Bien entendu, comme les résistances sont toutes égales et qu'il y a 16 comparateurs, on peut connaître sur quel échelon (tous égaux sur le segment) se trouve l'échantillon : si l'échantillon est supérieur à n tensions (par exemple sur les comparateurs 1 à 4), il se trouve sur le nième échelon ; la valeur des bits EFGH est alors n-1 en binaire (échelon 3, code 0011 sur l'exemple).

Le nième comparateur a bien pour tension de référence la tension seuil du nième échelon du segment. En effet, la tension de référence du nième comparateur est $VH + \frac{(n-1)\cdot(VB-VH)}{16}$, ce qui correspond bien au piédestal du nième échelon du segment concerné, avec $n \in [\![1;16]\!]$. On a donc un moyen de distinguer les 16 échelons du segment. Le code correspondant à l'échelon est le numéro en binaire (de 0 à 15) du comparateur associé à ce dernier.

A la fin du codage, on transmet le code ainsi élaboré dans un registre à décalage, qui construit le signal électrique numérique.

1.3). Simulation de la carte codeur.

Le but de cette partie est de concevoir en C une fonction d'émulation de la carte codeur. Le codeur à la norme TN1-2G est difficile à simuler à cause des comparateurs en parallèle. Nous allons donc reproduire le codeur à la norme $TN1_a$ $TN1_b$ qui se prête particulièrement bien à cet exercice.

Cette fonction doit prendre en argument une valeur quelconque (float), qui représente la tension échantillonnée à quantifier. La sortie sera un 'int' représentant les 8 éléments (donc un int compris entre 0 et 255). Il suffit, pour coder l'échantillon, de suivre l'ordre des comparaisons établi en 1.2.2), et d'envisager tous les cas.

On commence par définir les variables statiques, c'est à dire toutes les tensions de référence. Elles sont supposées intégrées à la logique de commande. On utilise des variables de type int[][], où le nième élément correspond aux tensions de référence à utiliser si on a BCD = n (on rappelle que le code final est A BCD EFGH). On retrouve ainsi facilement toutes les données correspondant à chaque cas.

```
// Table des différentes comparaisons suivant la valeur de (BCD)
static int lgc_ref[8][4] = {{Vmax/16, Vmax/32, Vmax/64, Vmax/128},
                    {Vmax/16, Vmax/32, Vmax/64, Vmax/128},
                    {Vmax/8, Vmax/16, Vmax/32, Vmax/64},
                    {Vmax/4, Vmax/8, Vmax/16, Vmax/32},
                    {Vmax/32, Vmax/64, Vmax/128, Vmax/256},
                    {Vmax/16, Vmax/32, Vmax/64, Vmax/128},
                    {Vmax/8, Vmax/16, Vmax/32, Vmax/64},
                    {Vmax/4, Vmax/8, Vmax/16, Vmax/32}};

// table des piedestaux suivant BCD
static int lgc_base[8] = {0, Vmax/8, Vmax/4, Vmax/2, Vmax/16,
Vmax/8, Vmax/4, Vmax/2};
```

Une fois ces variables définies, on peut écrire le codeur lui-même, qui dispose, par hypothèse, dans une zone mémoire de la logique de commande des tables de comparaisons définies plus haut. Les variables commençant par lgc sont des registres de la logique de commande :

```
int code (int val)
{
     int A, B, C, D, E, F, G, H; // Les 8 registres
     int lgc_gain = 16, lgc_signe, refer, segment;
     // Détermination de A et B simultanément
     A = (val >= 0);
     B = (lgc_gain * val >= Vmax || lgc_gain * val <= -Vmax);
     // Gestion du gain du signe dans la logique de commande
     lgc_gain = (B == 1) ? 1 : 16;
     lgc_signe = (A == 1) ? 1 : -1;
     // Détermination de C
     refer = Vmax / 4;
     C = (lgc_gain * lgc_signe * val >= refer);
     // Détermination de D
     refer = (C == 0) ? Vmax/8 : Vmax/2;
     D = (lgc_gain * lgc_signe * val >= refer);
     // Numéro du segment :  0..7
     segment = (B << 2) + (C << 1) + D;
     // Initialisation du piedestal
     refer = lgc_base[segment];
     // Détermination du segment
     if (lgc_gain * lgc_signe * val >= refer + lgc_ref[segment][0])
     {
          refer += lgc_ref[segment][0];
          E = 1;
     } else E = 0;
     if (lgc_gain * lgc_signe * val >= refer + lgc_ref[segment][1])
     {
          refer += lgc_ref[segment][1];
          F = 1;
     } else F = 0;
     if (lgc_gain * lgc_signe * val >= refer + lgc_ref[segment][2])
     {
          refer += lgc_ref[segment][2];
          G = 1;
     } else G = 0;
     H = (lgc_gain* lgc_signe * val >= refer + lgc_ref[segment][3]);
     return (A<<7)+(B<<6)+(C<<5)+(D<<4)+(E<<3)+(F<<2)+(G<<1)+(H);
}
```

Ce simulateur reprend exactement les étapes du codeur potentiométrique. val est l'échantillon passé en argument à cette fonction. Les variables de logique de commande sont pour les variables statiques, lgc_base et lgc_ref. La logique possède deux autres variables mises à jour à chaque opération de décomposition et codage ; ce sont lgc_signe, qui mémorise le signe de l'échantillon et lgc_gain qui représente la bascule de gain de l'amplificateur d'entrée (commandé par la logique). Le simulateur dispose aussi de huit registres A, B, C, D, E, F, G et H, que la commande peut interroger à volonté ; ces registres contiennent les bits correspondant à la valeur de l'échantillon. La variable refer est une variable temporaire contenant à chaque instant de la conversion la tension de référence pour la comparaison suivante. Elle simule la somme des injecteurs de courant. Enfin, segment est une variable crée après la détermination des 4 premiers bits contenant le numéro d'identification du segment.

2). LE TRANSCODAGE

2.1). Préliminaires.

2.1.1). Contraintes de transmission.

On va s'apercevoir que le signal numérique tel quel ne se prête pas à la transmission en ligne. Il faut le modifier pour pouvoir le transmettre. On va donc pratiquer un transcodage du signal, qui a pour but d'adapter le spectre de puissance du signal numérique au contraintes de transmission en modifiant la représentation électrique du signal binaire. Il faut respecter quelques critères :

– Le spectre de puissance ne doit pas avoir de composante continue : la valeur moyenne du signal doit être nulle. On a besoin de cette propriété car il est impossible de concevoir un canal de transmission sans aucun traitement électronique. La ligne de France Télécom comporte en effet plusieurs transformateurs qui ne transmettent pas la composante continue à cause du phénomène d'induction.

– Le spectre doit avoir un support le plus étroit possible. En effet, tout système de traitement possède une fréquence de coupure basse et une fréquence de coupure haute au delà desquelles il ne fonctionne pas correctement. Toute information se trouvant hors de la bande de fonctionnement du matériel est alors perdue.

– Le signal doit posséder une raie d'énergie à la fréquence d'horloge, ou fréquence de bit (Fb), pour permettre la régénération du signal.

Si ces critères sont respectés, le signal va pouvoir être propagé, régénéré, et récupéré. La communication est donc possible. On va aussi chercher un codage vérifiant ces critères qui soit le plus simple possible. La conversion en sera plus rapide, et par voie de conséquence, le débit plus élevé.

2.1.2). Quelques outils de calcul.

Au cours de cette partie, nous serons amenés à calculer des spectres de puissance de différents signaux numériques. Nous avons donc besoin d'une fonction permettant de calculer le spectre de puissance d'un signal échantillonné. Cette fonction est la densité spectrale de puissance (DSP), qui se calcule avec une formule simple : Soit x(t) le signal temporel, possédant une transformée de Fourier X(f), on définit la DSP par : $S_{xx}(f) = X(f) \cdot \overline{X}(f) = \left|X(f)\right|^2$.

On utilise la transformée de Fourier discrète (TFD), donnée par : $X(f) = \frac{1}{N} \cdot \sum_{k=0}^{N-1} x(k) \cdot e^{\frac{-2 \cdot i \cdot \pi}{N} \cdot f \cdot k}$.

Du point de vue algorithmique, nous allons implémenter en C l'algorithme de la FFT, c'est à dire l'algorithme de Cooley et Tuckey. C'est un algorithme fonctionnant sur un principe simple : il suffit de remarquer que $X(f) = \frac{1}{2} \cdot (P(f) + \omega_N^{-f} \cdot I(f))$, où $\begin{cases} P(f) = \frac{1}{m} \cdot \sum_{k=0}^{m-1} x(2k) \cdot e^{\frac{-2 \cdot i \cdot \pi}{N} \cdot f \cdot 2k} \\ I(f) = \frac{1}{m} \cdot \sum_{k=0}^{m-1} x(2k+1) \cdot e^{\frac{-2 \cdot i \cdot \pi}{N} \cdot f \cdot 2k} \end{cases}$, à condition que l'on ait N pair, et en posant $N = 2 \cdot m$. On remarque que P(f) et I(f) sont deux transformées de Fourier d'ordre moitié. Cet algorithme a cependant un inconvénient : il faut, qu'au départ, N soit une puissance de 2 pour pouvoir mener à bien tous les calculs.

Le format des nombres complexes et les prototypes des opérations dans $\mathbb{C}$:

```
typedef struct
{
     float re, im;
} cplx;

cplx zc (); // Le zéro
cplx mulc (cplx, cplx); // Multiplication
cplx addc (cplx, cplx); // Addition
cplx sousc (cplx, cplx); // Soustraction
cplx divcr (cplx, float); // Division par un réel
cplx conj (cplx); // Conjugaison
float module (cplx); //Module
```

L'algorithme de la FFT récursive prend en argument e, le vecteur des échantillons au format cplx définit plus haut, de longueur une puissance de 2 :

```
int fft (cplx *e, long n)
{
     cplx *p, *i, w, wk, x, y;
     int m = n / 2, k;
     if (n == 1) return 0;//fin de récursivité : fft([a]) = [a]
     if (2 * m != n) return 1;
     i = (cplx *) calloc (m, sizeof (cplx));
     p = (cplx *) calloc (m, sizeof (cplx));
     // Découpage de la série de données pairs/impairs
     for (k = 0; k <= m - 1; k++)
     {
               *(p+k) = *(e+2*k);
               *(i+k) = *(e+2*k + 1);
     }
     if (fft (p, m) == 1) return 1;
     if (fft (i, m) == 1) return 1;
     wk.re = 1; wk.im = 0;
     w.re = cos (2 * PI / (float)n);
     w.im = - sin (2 * PI / (float)n);
     // Combinaison de la fft des pairs/impairs
     for (k = 0; k <= m - 1; k++)
     {
               x = *(p+k); y = mulc (wk, *(i+k));
               *(e+k) = divcr (addc (x, y), 2);
               *(e+k+m) = divcr (sousc (x, y), 2);
               wk = mulc (wk, w);
     }
     free (i); free (p);
     return 0;
}
```

Au fur et à mesure, on définira les générateurs aléatoires de chacun des codes binaires testés, afin de calculer une moyenne statistique de la DSP de chacun de ces codes, pour pouvoir raisonner sur leurs avantages et inconvénients. Ces générateurs devront rendre une série aléatoire de longueur une puissance de 2, et utilisant le format 'cplx *', pour pouvoir exploiter les résultats avec l'algorithme fft décrit plus haut.

2.2). Etude de différents codes binaires possibles.

2.2.1). Le code binaire brut, ou NRZ.

Ce code binaire est celui que l'on voit usuellement. Les impulsions durent tout le temps impartit pour un bit, soit 488 ns. C'est pour cela qu'il s'appelle NRZ : Non Retour à Zéro. Voici un exemple que nous retrouverons pour chacun des autres codages, il s'agit du signal électrique correspondant à la séquence 100001011110000000011001 :

Le générateur aléatoire est ici sans grand intérêt, puisqu'un simple tirage à valeurs dans {0; 1} sans calcul de polarité suffit. Voici son code source en C :

```
int nrz (cplx *e, long n)
{
   int i;
   for (i = 0; i < n; i++)
   {
          (*(e+(8*i))).re = F * (rand()&1);
          (*(e+(8*i))).im = 0;
          *(e+(8*i)+1) = *(e+(8*i));
          *(e+(8*i)+2) = *(e+(8*i));
          *(e+(8*i)+3) = *(e+(8*i));
          *(e+(8*i)+4) = *(e+(8*i));
          *(e+(8*i)+5) = *(e+(8*i));
          *(e+(8*i)+6) = *(e+(8*i));
          *(e+(8*i)+7) = *(e+(8*i));
   }
   return 0;
}
```

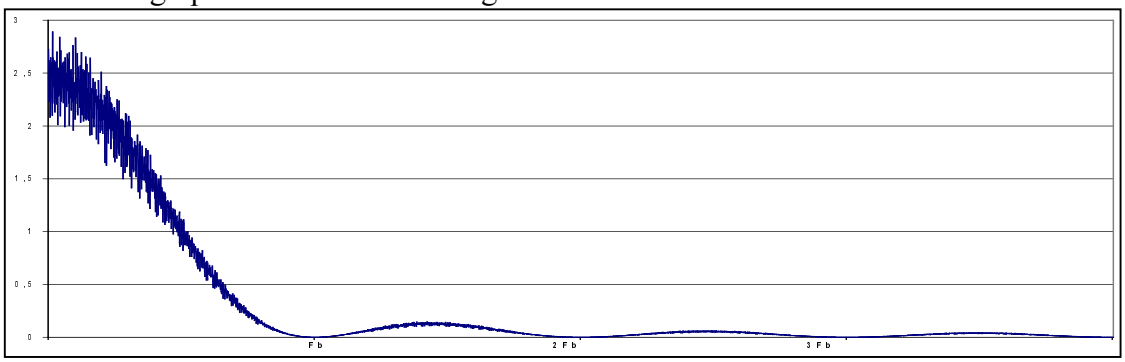
Les arguments de cette fonction sont : – n le nombres de bits de l'échantillon.
– f un facteur amplificateur du signal.

Le programme entre 8 fois la même valeur. Ainsi, on obtient l'échantillonnage d'un signal en créneaux aléatoire 8 fois par période d'un bit, c'est à dire 8 fois toutes les 488 ns. Ce choix n'est pas innocent car il suffit alors de rentrer en argument 'n' une valeur puissance de 2 pour obtenir un résultat de dimension une puissance de 2. De plus, cela permet d'obtenir un spectre de puissance fiable jusqu'à 4 fois la fréquence binaire. En effet, au delà de cette fréquence, on retrouve le symétrique du spectre, périodisé du fait de l'échantillonnage ; le résultat devient alors faux pour les hautes fréquences. En résumé, on simule ici un échantillonnage avec $Fe = 8 \cdot Fb = 16MHz$.

On utilise ce générateur aléatoire et la fonction 'fft' définie au 2.1.2). On obtient ainsi un vecteur contenant la DSP de ce signal, avec un pas de fréquence dépendant de la longueur de l'échantillon choisi. En effet, le pas est définit par $Fe/(8 \cdot n)$, où n est le nombre de bits.

On opère une moyenne sur un certain nombre de DSP de différentes séries aléatoires de même longueur pour obtenir une meilleure précision du spectre, car d'une part, le générateur n'est que pseudo aléatoire, et d'autre part, l'échantillon ne peut pas être infini. Le graphe suivant ainsi que tous les autres graphes de DSP à suivre sont faits à partir d'une moyenne de 10 séries de 256 bits (soit 2048 échantillons). On obtient une résolution suffisante, et une moyenne de 10 séries donne des résultats corrects : augmenter le nombre de séries n'apporte aucune information supplémentaire.

Voici le graphe de la DSP du codage NRZ :



On observe deux inconvénients majeurs :

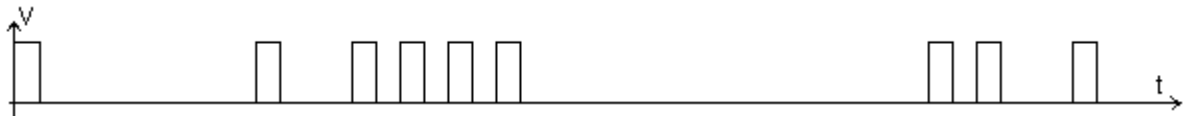
– Il existe une importante tension continue (aux basses fréquences), qui empêche le passage du signal par des transformateurs. Ce défaut est dû à la valeur moyenne du signal qui n'est pas nulle (Vmax/2).

– Le signal n'a aucune énergie à la fréquence binaire Fb. On est donc dans l'impossibilité de récupérer le signal d'horloge.

Si on observe le signal NRZ, on s'aperçoit que le manque d'énergie à la fréquence binaire, donc la perte du signal d'horloge, est en partie dû aux longues séries de 1. En effet, lorsque le signal comporte une importante série de 1, il n'y a plus de transition de niveaux, on a une tension constante égale à Vmax. On a par conséquent conçu un autre codage qui corrige ce défaut, c'est le RZ 50%.

2.2.2). Première amélioration, le RZ 50%.

Ce codage est très similaire au NRZ. Mais au lieu de maintenir la tension pendant toute la durée d'un bit, on ne la maintient que sur la moitié de la période binaire Tb, soit 244 ns. De là vient son nom : Retour à Zéro à 50%. En illustration, voici le signal associé à la même séquence binaire que précédemment :

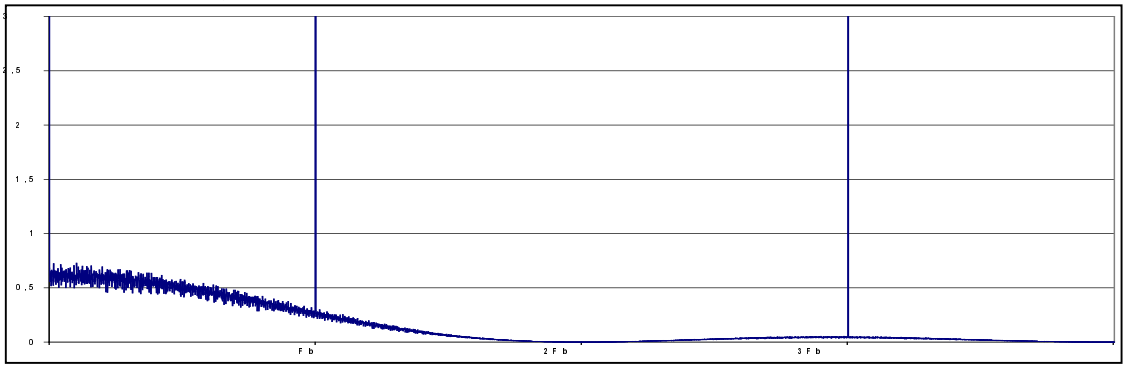


Suit son générateur aléatoire, toujours en C :

```
int rz (cplx *e, long n)
{
   int i;
   for (i = 0; i < n; i++)
   {
         (*(e+(8*i))).re = F * (rand()&1);
         (*(e+(8*i))).im = 0;
         *(e+(8*i)+1) = *(e+(8*i));
         *(e+(8*i)+2) = *(e+(8*i));
         *(e+(8*i)+3) = *(e+(8*i));
         *(e+(8*i)+4) = zc();
         *(e+(8*i)+5) = zc();
         *(e+(8*i)+6) = zc();
         *(e+(8*i)+7) = zc();
   }
   return 0;
}
```

On peut supposer qu'on ne perd plus le signal d'horloge avec ce codage, puisque les longues séries de 1 ne produisent plus une tension constante, mais une tension en créneaux de période Tb, soit exactement la période d'horloge.

Effectivement, voici sa DSP :

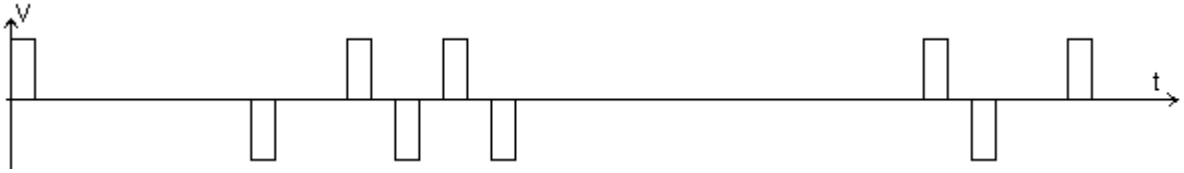


On peut donc bien, avec ce codage, récupérer l'horloge. Mais il subsiste toujours une importante composante continue. Ce type de signal n'est par conséquent pas transmissible en ligne : une grande partie de l'information serait perdue.

Cependant, si on observe attentivement un tel signal, on remarque aisément sans calcul que sa valeur moyenne est Vmax/4, sachant que les valeurs 0 ou 1 sont équiprobables. On veut donc ramener cette valeur moyenne à 0. Un signal purement binaire (à valeurs dans {0; 1}) ne suffit pas. On va donc élargir l'ensemble des valeurs que peut prendre le signal, en rajoutant une valeur négative. C'est le rôle du codage AMI.

2.2.3). Une évolution significative, le codage AMI.

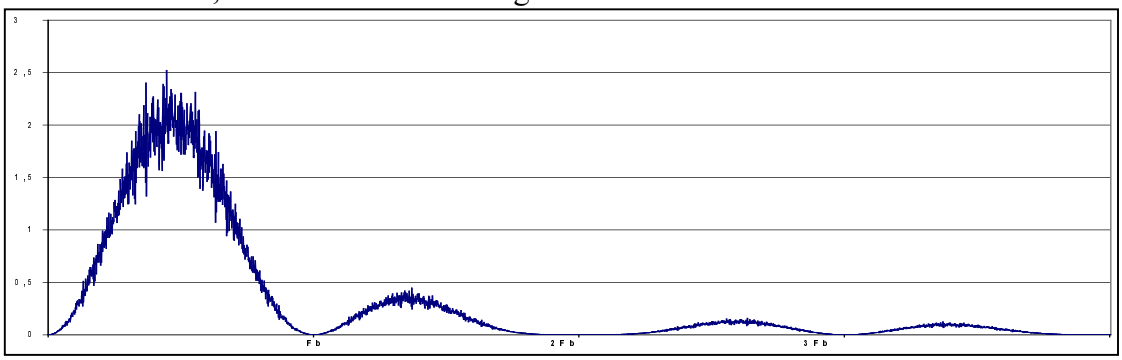
Dans ce codage, le Alternated Mark Inverted, On ne maintient la tension que pendant Tb/2, soit 244 ns, comme pour le RZ 50%. Cependant, si les 0 restent toujours au niveau 0V, les 1 sont maintenant alternés dans {-V; V}. On a donc un signal trivalent. Voici la même séquence binaire représentée par son signal AMI :



Voici maintenant son générateur aléatoire :

```
int ami (cplx *e, long n)
{
   int i, signe = 1;
   for (i = 0; i < n; i++)
   {
         (*(e+(8*i))).re = signe * F * (rand()&1);
         (*(e+(8*i))).im = 0;
         *(e+(8*i)+1) = *(e+(8*i));
         *(e+(8*i)+2) = *(e+(8*i));
         *(e+(8*i)+3) = *(e+(8*i));
         *(e+(8*i)+4) = zc();
         *(e+(8*i)+5) = zc();
         *(e+(8*i)+6) = zc();
         *(e+(8*i)+7) = zc();
         signe *= ((*(e+(8*i))).re != 0) ? -1 : 1;
   }
   return 0;
}
```

Voici sa DSP, calculée à l'aide de ce générateur :



Un tel message ne comporte plus de composante continue. On peut donc le transmettre en ligne. Cependant, la raie d'énergie n'est plus présente à la fréquence binaire. Mais en fait, ce n'est pas grave, en effet, si on redresse ce signal, on retrouve exactement un signal RZ 50%.

De plus, un autre avantage est que environ 80% de l'énergie de ce signal se trouve à une fréquence inférieure à Fb/2 (pic à 0,46 Fb). On peut donc filtrer le signal pour ne conserver que la bande de fréquence [0; Fb/2] sans perte d'information rédhibitoire. Ceci permettra ensuite de moduler plusieurs signaux en fréquence pour augmenter le débit binaire théorique de la ligne.

Cependant, ce signal possède un inconvénient : une longue séquence de 0 risque d'entraîner la perte de la raie d'énergie à Fb, donc une désynchronisation du signal d'horloge. Il faut donc maintenir une densité binaire suffisante pour éviter ce phénomène. C'est le rôle du codage HDB3.

2.2.4). Un codage vérifiant tous les critère, le HDB3.

HDB3 signifie Haute Densité Bipolaire d'ordre 3. C'est un cas particulier du codage HDBn, où on s'arrange pour qu'il n'y ait jamais plus de n bits nuls consécutifs.

Comme pour le AMI, les 1 respectent la bipolarité entre eux. Pour qu'il n'y ait pas quatre 0 consécutifs, on remplace le quatrième 0 par un 1. Pour le distinguer des véritables 1, on l'insère en viol de bipolarité, c'est-à-dire selon la même polarité que le 1 précédent. Cela permettra de reconnaître que c'est un 0 lors de la restitution. C'est donc un bit de "viol".

Cependant, cela risque d'introduire une composante continue. Donc on introduit un bit dit de "bourrage" selon une règle bien précise qui a pour but de faire respecter la bipolarité entre les bits de viol. Ainsi, si le nombre de 1 séparant deux suites de quatre 0 est nul ou pair, on remplace le premier bit de la deuxième séquence de quatre bits par un 1, qui a cette fois le même statut qu'un véritable 1, c'est-à-dire qu'il respecte la bipolarité des 1. On peut illustrer ce codage par la même séquence que précédemment :

Son générateur est plus compliqué, car il faut entretenir un compteur qui détermine si le nombre de 1 séparant deux séquences de quatre 0 est pair, et introduire le bit de bourrage rétroactivement :

```
int hdb3 (cplx *e, long n)
{
   int i, signe = 1, c0 = 0, c1 = 0;
   cplx val;
   for (i = 0; i < n; i++)
      {
         val.re = signe * F * (rand()&1);
         val.im = 0;
         if (val.re != 0)
```

```
            {
                  *(e+(8*i)) = val;
                  c0 = 0;
                  c1 += 1;
                  signe *= -1;
            }
            else
            {
                  c0 += 1;
                  switch (c0)
                        {
                  case 4 : if ((c1&1) == 0)
                        {
                              // bit de bourrage (retroactif)
                              (*(e+(8*(i-3)))).re = F * signe;
                              *(e+(8*(i-3))+1) = *(e+(8*(i-3)));
                              *(e+(8*(i-3))+2) = *(e+(8*(i-3)));
                              *(e+(8*(i-3))+3) = *(e+(8*(i-3)));
                              signe *= -1;
                        }
                        (*(e+(8*i))).re = (-signe) * F; // bit de viol
                        c0 = 0; c1 = 0;
                        break;
                  default : *(e+(8*i)) = val;
                   }
            }
            *(e+(8*i)+1) = *(e+(8*i));
            *(e+(8*i)+2) = *(e+(8*i));
            *(e+(8*i)+3) = *(e+(8*i));
            *(e+(8*i)+4) = zc();
            *(e+(8*i)+5) = zc();
            *(e+(8*i)+6) = zc();
            *(e+(8*i)+7) = zc();
      }
      return 0;
}
```

Ce code bénéficie des mêmes avantages que le AMI, mais en plus, il maintient une densité binaire beaucoup plus élevée : il n'existe pas dans ce type de codage des séquences nulles de plus de 3 bits. On élimine donc le risque de perdre la synchronisation de l'horloge. De plus, la forte densité des 1 permet après redressement du signal d'avoir un créneau quasiment complet, puisqu'il n'existe pas de suite de 0 prolongée.

Ce signal a un autre avantage de taille : tout viol de bipolarité autre que ceux introduits par le codage sont détectés comme des erreurs : on peut donc augmenter la fiabilité de la transmission sans pour autant introduire des codes correcteurs d'erreur très évolués.

En illustration, voici sa DSP :

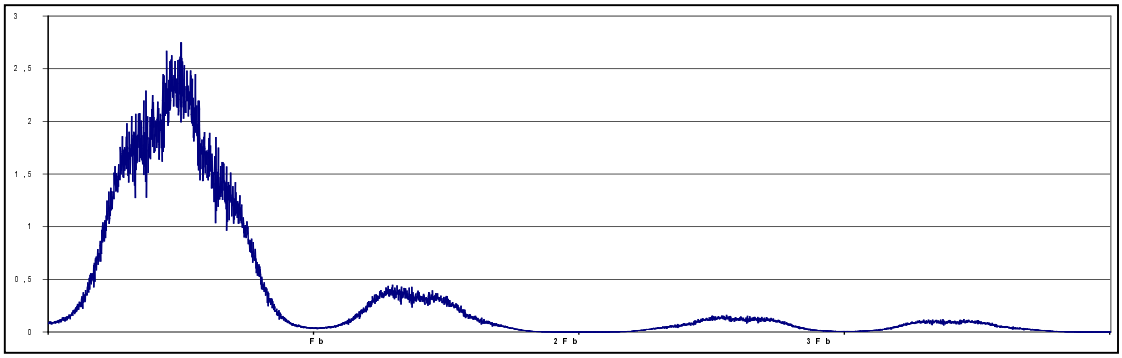


TABLE DES MATIERES

INTRODUCTION 2

1). LA QUANTIFICATION 4

1.1). Principe. 4

1.1.1). Un premier problème. 4

1.1.2). Un second problème. 5

1.2). Mise en œuvre, principaux codeurs 6

1.2.1). Le codeur à pente. 6

1.2.2). Le codeur potentiométrique ou à résistances. 9

1.2.3). Le codeur série/parallèle sans maintien. 12

1.3). Simulation de la carte codeur. 12

2). LE TRANSCODAGE 15

2.1). Préliminaires. 15

2.1.1). Contraintes de transmission. 15

2.1.2). Quelques outils de calcul. 15

2.2). Etude de différents codes binaires possibles. 17

2.2.1). Le code binaire brut, ou NRZ. 17

2.2.2). Première amélioration, le RZ 50%. 18

2.2.3). Une évolution significative, le codage AMI. 19

2.2.4). Un codage vérifiant tous les critère, le HDB3. 20

TABLE DES MATIÈRES 22