UNIVERSITÉ DE MONTRÉAL

CONCEPTION ET IMPLEMENTATION D'UN DÉCODEUR DÉDIÉ À UN MODULATEUR SIGMA-DELTA

MOHAMED AMINE MILED
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAITRISE ÉS SCIENCES APPLIQUÉES
(GÉNIE ÉLECTRIQUE)
NOVEMBRE 2007

© MOHAMED AMINE MILED, 2007

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-36924-1
Our file Notre référence
ISBN: 978-0-494-36924-1

NOTICE:
The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:
L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

UNIVERSITÉ DE MONTRÉAL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

CONCEPTION ET IMPLEMENTATION
D'UN DÉCODEUR DÉDIÉ À UN MODULATEUR SIGMA-DELTA

Présenté par : MOHAMED AMINE MILED
en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de :

M. SAVARIA Yvon,	Ph.D., président
M. SAWAN Mohamad,	Ph.D., membre et directeur de recherche
M. CARDINAL Christian,	Ph.D., membre

À ma famille …

Remerciements

Tout d'abord mes vifs remerciements s'adressent à mon directeur de recherche M. Mohamad Sawan, professeur à l'École polytechnique de Montréal, qui m'a permis, tout au long de ces deux dernières années, d'effectuer mes travaux dans d'excellentes conditions de travail au sein de son équipe Polystim et qui, par la même occasion, m'a soutenu financièrement.

Je tiens également à remercier M. Yvon Savaria, professeur à l'École polytechnique de Montréal, ainsi que M. Christian Cardinal, professeur à l'École polytechnique de Montréal, d'avoir acceptés respectivement de présider et d'être membre du jury de ce mémoire.

Je voudrais aussi remercier M. Ebrahim Ghafar-Zadeh, étudiant au doctorat dans l'équipe Polystim qui, par ses conseils, m'a offert l'occasion d'élargir mes connaissances et d'enrichir mon expérience. Je tiens à remercier également toute l'équipe Polystim, sans oublier, M. Réjean Lepage, M. Gaétan Décarie et Maxime Thibault, spécialistes techniques du département de génie électrique de l'École polytechnique de Montréal.

Je dois ma gratitude à mes très chers parents Faiza et Fadhel Miled et à ma sœur Emna pour leur soutient moral et encouragement indéfectible tout au long de ma recherche en dépit de la distance qui nous sépare. Je remercie tous mes proches et en particulier, Lotfi et Tarek Khaddar, mon cousin Youssef Khouja et sa femme pour leur aide omniprésente.

Pour finir, toute ma reconnaissance aux personnes qui ont contribué de prés ou de loin à la réalisation de ce projet de maîtrise.

Résumé

Les convertisseurs analogique-numérique (CAN) occupent une place prépondérante dans les circuits électroniques et le choix du convertisseur dépend considérablement de la nature de l'application. Dans le cadre de notre projet de détection des substances micro fluidiques par le biais de dispositifs miniaturisés tels que les laboratoires sur puces, les CAN Sigma-Delta ($\Sigma\Delta$) sont parmi les meilleurs candidats. D'abord parce que ces derniers permettent une précision élevée, en plus ils sont plus facilement intégrables dans un circuit intégré dédié. Cependant de tels convertisseurs exigent un certain traitement de signal spécialisé ce qui peut conduire à un temps de conversion élevé. Ainsi, dans ce projet, notre objectif est de réduire ce temps de conversion pour accélérer le traitement de signal.

Un convertisseur $\Sigma\Delta$ se compose principalement de deux grandes parties : le modulateur et le décimateur (ou décodeur). Le décodeur consiste à rendre l'information fournie par le modulateur interprétable par l'utilisateur. Pour réaliser cette partie deux approches sont possibles : 1) filtrage, 2) décodage. Le décodage permet d'atteindre une meilleure précision mais son implémentation matérielle est plus complexe que celle du filtrage et le délai de traitement des données est plus long. Notre choix s'est arrêté sur le décodage vu que la précision est d'une grande importance pour notre application, en plus nous avons proposé une technique de décodage dynamique permettant une réduction considérable du temps de traitement. Notons que l'utilisation d'un algorithme de décodage plutôt que le filtrage est justifiée par le fait que les séquences générées par un modulateur $\Sigma\Delta$ renferment un bon niveau de bruit de sorte que le filtrage peut supprimer une partie de l'information recherchée. C'est pour cette fin que le décodage est mieux adapté dans le cas où on s'intéresse à la précision.

Quant à l'architecture dynamique son avantage est d'éviter la redondance dans le traitement de l'information. Ceci est très bénéfique pour des applications dont la

fréquence de fonctionnement est faible. L'architecture proposée a permis une accélération du temps de traitement de 2 à 4 fois. Elle se base en partie sur un algorithme de décodage itératif qui a été récemment publié.

Pour valider l'algorithme proposé nous avons implémenté son architecture dans un composant programmable (FPGA). Les résultats montrent qu'une telle approche permet d'atteindre un gain moyen de 4.00 dB pour une séquence de longueur 8 bits et de 1.70 dB pour une séquence de longueur 80 bits.

L'architecture proposée a été implémentée sur la plateforme de développement d'Actel AFS-EVAL-BRD qui consiste en un nouveau composant mixte (analogique/numérique) programmable. En effet, cette plateforme donne accès à toutes les entrées sorties du FPGA en plus d'intégrer des outils de bases tels que des horloges, des régulateurs de tensions ainsi qu'à toute la fonctionnalité d'un FPGA comme le générateur d'horloge interne et le CAN intégré à ce dernier.

Abstract

Analog to Digital Converters have a considerable impact in electronic circuits. In fact, there are several types of ADCs and, according to every application; one type of ADC may be adapted better than others. Within the framework of our project intended to the detection of microfluidic substances based on a laboratory on chip platform, the ΣΔ converters are the best choice. These ΣΔ converters have a better precision than others and can be easily implemented. But such converters require specialized signal processing work, which can lead to a high conversion time. Thus, in this project, our goal is to reduce such conversion time in accelerating the signal processing tasks.

Basically a ΣΔ converter is composed of two parts: the modulator and the decimator or decoder. The decoding part consists of extracting the information related to the sampled signal from the ΣΔ modulator. Nevertheless, to carry out this part, two approaches can be chosen: 1) filtering, and 2) decoding. The decoding technique offers higher precision, but it is more complex and requires more time for data processing. The decoding technique is more adapted for our project than filtering, because sequences generated by the ΣΔ modulator contain much more information mixed with noise thus filtering can remove a part of the requested information to restore the sampled signal. Thus, decoding is more useful in our project since high precision is our main goal.

At the same time, using a dynamic architecture in the decoder is useful to avoid redundancy in the data processing. This is advantageous for low frequency applications. The proposed architecture offers data processing acceleration from 2 to 4 times. It is based on an iterative recently published decoding algorithm.

To validate the proposed decoding algorithm, we implemented its architecture on a Field-programmable gate array (FPGA). The results show that such an approach offers an average gain of 4.00 dB for an 8-bit sequence and 1.70 dB for an 80-bit sequence.

Actel AFS-EVAL-BRD prototyping kit was used to implement the proposed architecture. This platform provides access to all FPGA Inputs/Outputs in addition to configurable clock and voltage regulators; it allows access to all FPGA functions such as the clock generator and an integrated ADC available inside this mixed-signal (analog-digital) FPGA.

Table des matières

Liste des figures

Liste des tableaux

Liste des abréviations

ΣΔ	Sigma-Delta
ADC	Analog to Digital Converter
CAN	Convertisseur Analogique Numérique
CNA	Convertisseur Numérique Analogique
CNIS	Create New Intermediate Sequence
DAC	Digital to Analog Converter
DSP	Digital Signal Processor
FIR	Finite Impulse Response : filtre à réponse impulsionnelle
FPGA	Field-Programmable Gate Array
FSM	Finit State Machine
LoC	Lab-on-Chip
MEF	Machine à États Finis
MSE	Mean Square Error
NCR	Non Closed Run
RCS	Run and Correction Scan
SQNR	Signal Quantized to Noise Ratio
SNCR	Scan for Non Closed Run

Liste des annexes

Introduction

De plus en plus de dispositifs traitant l'information de façon numérique sont présents sur le marché permettant un traitement avancé de cette information, à titre d'exemples, nous pouvons citer le plus célèbre d'entre eux soit l'ordinateur personnel (PC) ou encore des circuits intégrés spécialisés tels que les microcontrôleurs ou les FPGA. Le point commun entre tous ces outils est le fait qu'ils ne peuvent traiter une information numérique codée que par des valeurs binaires.

Cependant la source d'un signal électrique est souvent analogique et non numérique comme les signaux électriques captés du corps humains qui ont de faibles amplitudes et qui sont extrêmement sensibles. Mais dans des applications bien particulières telles que les dispositifs basés sur la nouvelle technique laboratoire sur puce (Laboratory on chip – LoC), la précision est encore plus accrue. Une plateforme LoC, faisant l'objet de nos applications, et des mini laboratoires gravés sur des verres ou des polymères ou encore sur des puces et sous la forme d'un réseau de canaux microfluidiques. Les LoC permettent l'analyse ou la détection d'un échantillon, d'un mélange ou de toute autre substance liquide. Leurs principaux avantages sont leur facilité d'utilisation et la reproductibilité du résultat grâce à l'automatisation et la standardisation du processus, la vitesse de l'analyse et la faible consommation d'échantillons [1]-[5].

Un exemple d'utilisation d'un LoC est l'injection des médicaments dans le corps humain avec des doses bien précises à des moments bien déterminés. Une simple erreur de dosage ou de temporisation peut avoir des conséquences extrêmement graves. D'où l'importance de contrôler des micromachines avec une précision accrue [6]-[7]. Un tel contrôle est généralement effectué par d'autres composants qui sont soit un PC ou un circuit dédié au traitement de signal comme les FPGA. Cependant ces unités de

traitement de signal sont, en grande partie, de circuits numériques, autrement dit, ils ne peuvent fournir et recevoir que des 0 et 1. Et donc une interface s'impose entre l'unité de contrôle et les circuits analogiques. Ces interfaces sont connues sous le nom de convertisseurs analogique/numérique (CAN) (Analog to digital converters - ADC) et de convertisseurs numériques/analogiques (CNA) (Digital to analog converters - DAC). Les CAN reçoivent un signal analogique, font la conversion numérique et envoient le signal binaire obtenu suite à la conversion à l'unité de traitement. Ces derniers doivent être capables de convertir un signal avec une précision élevée et un temps de conversion le plus faible possible surtout pour les applications biomédicales. Dans notre application notre objectif est d'avoir un temps de conversion ne dépassant pas les 100 ns. À titre d'exemple le CAN que nous avons mis en œuvre pour l'application LoC est capable de détecter une variation de capacité de l'ordre de 1 fF. Ceci implique une variation très faible du signal analogique provenant du LoC, et donc une précision élevée s'impose du coté de l'ADC.

De nombreux travaux de recherche ont été publiés pour chacun des points cités précédemment. Certains d'entre eux tentent de minimiser les bruits et les artéfacts introduits par l'environnement extérieur en augmentant la résolution des CAN. Le revers de la médaille est le temps de conversion qui augmente en fonction de la résolution et par la suite la vitesse d'échantillonnage qui décroît. L'espace occupé par le CAN prend de plus en plus d'importance, vu qu'il est souhaitable de minimiser au minimum l'espace d'une puce électronique.

Plusieurs architectures de CAN ont été proposées au fil du temps dépendamment de l'évolution technologique. Toutes ces architectures peuvent être classées en ces 4 grandes familles de convertisseurs parmi elles:

- les convertisseurs à approximation successive dont le fonctionnement consiste en une approximation du signal d'entrée à des valeurs prédéfinies. Le signal obtenu

à la sortie n'est qu'une représentation de cette approximation. Plus le niveau d'approximation est élevé, meilleure est la résolution.

- les convertisseurs parallèles dont le fonctionnement se base sur la comparaison de la tension d'entrée à plusieurs seuils et une transformation du résultat de la comparaison par une logique combinatoire en un mot binaire.
- les convertisseurs basés sur les distributions de charges à l'aide de capacités qui se chargent et se déchargent dépendamment de la tension d'entrée.
- les convertisseurs Sigma-Delta dont le principe de fonctionnement se base sur le sur-échantillonnage du signal analogique. L'utilisation des filtres numériques et du décodage permet d'atteindre une résolution pouvant aller jusqu'à 20 bits pour ce type de CAN. Ces convertisseurs sont spécialement adaptés pour les signaux à faible fréquence et haute résolution. Ils ont, en plus, une grande précision et une linéarité remarquable.

Autre point culminant est l'introduction de l'analyse dynamique, approche devenue possible avec les moyens technologiques dont nous disposons actuellement et spécialement les FPGA et DSP qui permettent un traitement élaboré de l'information. Cette approche qui adopte une conduite particulière dépendamment de l'environnement dans lequelle elle évolue est de plus en plus convoitée par les chercheurs. Le but de cette approche est d'accélérer les tâches de traitement du signal. Ceci est très important dans les applications biomédicales vu que les fonctions biologiques se déroulent à des vitesses relativement faibles. Dans ce cas, il est souhaitable de ne pas refaire un ensemble de traitement identique si l'information traitée reste inchangée. D'où l'idée d'adopter une approche dynamique dans la phase de la conversion des données. Cette technique donnerait aux convertisseurs la possibilité d'adopter un traitement ou un autre pour une information donnée. Ajoutons que cette technique est déjà appliquée pour la gestion du flux des données sur Internet [8] ou pour les applications robotiques. En résumé, l'approche dynamique permet de donner une autonomie de décision à des

machines qui sont de plus en plus appelées à fonctionner dans des environnements inconnus dont le corps humain fait partie.

L'équipe de recherche Polystim étant spécialisée dans la conception des implants destinés à être utilisés dans le corps humain, ce projet s'inscrit dans le cadre de la mise en œuvre des microsystèmes basés sur un laboratoire sur puce dont la fonction est de déterminer la substance microfluidique et ses caractéristiques pour des applications médicales. Ce genre de microsystème nécessite des CAN à haute précision capable de détecter une variation de tension inférieure à 30 mV tout en réduisant les parasites introduits par la le LoC, mais souvent à très basse consommation d'énergie. Notre choix s'est arrêté sur les CAN Sigma-Delta. Cependant ce type de CAN nécessite un module de décodage des données.

Le présent projet de maîtrise concerne plus précisément le développement d'une architecture de décodage dynamique pour les CAN Sigma-Delta.

Nous introduisons dans le chapitre 1 les architectures des principaux convertisseurs analogique/numérique, leurs avantages et inconvénients et une comparaison permettant de mettre en évidence les avantages des uns par rapport aux autres.

Le chapitre 2 fera la lumière sur les convertisseurs analogique/digital Sigma-Delta ainsi que sur les différentes techniques de décodage qui accompagnent ces convertisseurs. L'algorithme de décodage utilisé ainsi que l'approche dynamique feront l'objet du 3ème chapitre. Quant au 4ème chapitre, il détaille l'implémentation de l'architecture de décodage proposée sur FPGA, l'architecture statique et la gestion dynamique des données y sont toutes les deux présentées. Le dernier chapitre est consacré à la présentation des résultats de l'implémentation que ce soit l'espace occupé, la puissance consommée ou encore la validation matérielle du fonctionnement du

décodeur développé. Une comparaison des résultats entre les approches statique et dynamique est présentée pour mettre en évidence les avantages de la gestion dynamique dont le principal atout est la réduction du temps de conversion. L'architecture développée a permis une réduction du temps de conversion de 2 à 4 fois dépendamment du signal échantillonné tout en assurant un bon rapport signal sur bruit.

Chapitre 1
Introduction aux convertisseurs analogique-numérique

1.1. Introduction

Un système électronique peut renfermer plusieurs composants et circuits électriques. Dans le cas d'un laboratoire sur puce, système qui fait l'objet de nos applications et présenté dans l'introduction du mémoire; nous retrouvons un étage d'acquisition de données biologiques, un autre qui transforme ces données en un signal électrique, une unité de traitement, un contrôleur de l'ensemble et un dernier module qui représente l'interface usager.

Dans l'application que nous avons développée, le laboratoire sur puce est un système qui reçoit les données biologiques et les transforme en un signal électrique, un FPGA traite le signal et enfin un PC représente la partie de contrôle et l'interface avec l'utilisateur. Ainsi une chaîne d'acquisition des données recueille les informations provenant de l'environnement extérieur et envoie ces informations sous une forme appropriée à une unité spécialisée pour leur exploitation [9]. Or, ces différents éléments ne peuvent pas communiquer entre eux directement, étant donné qu'ils traitent des informations de nature différente. La Figure 1.1 représente la chaîne d'acquisition des données qui montre les interactions entre ces différents modules.

Dans notre application ou nous cherchons à avoir les données les plus précises tout en réduisant le temps de conversions pour permettre au système de détecter n'importe quelle variation dès son apparition pour cette fin une approche dynamique s'impose.

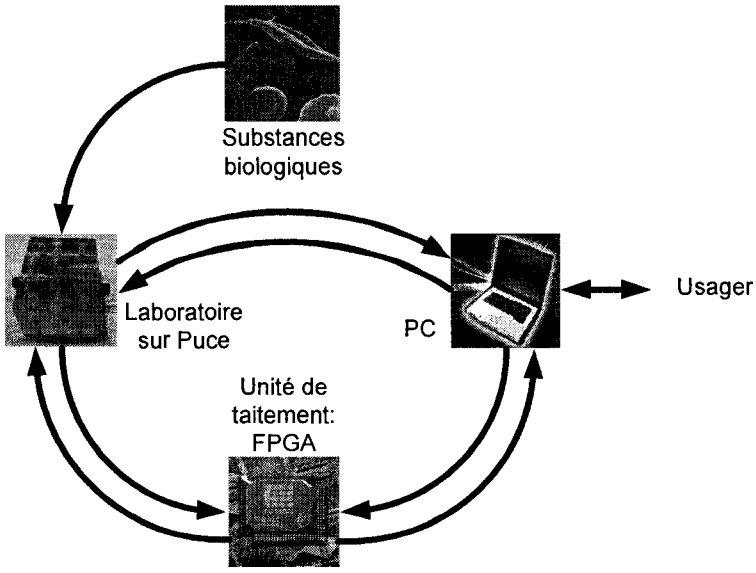


Figure 1.1 Chaîne d'acquisition des données

Nous donnons ci-dessous quelques exemples de données manipulées par chaque module dans le cadre du projet de laboratoire sur puce:

- le module biologique manipule des substances moléculaires.
- le laboratoire sur puce gère le flux de cellules biologiques et génère des signaux électriques analogiques.
- le FPGA traite les données numériques.
- Le PC traite, lui aussi, les données numériques et génère des données des graphiques ou des tableaux qui peuvent être visualisés sur son écran.

Les diverses parties de ce système nécessitent des composants qui permettent la liaison et la communication entre ces dernières. De tels composants sont les convertisseurs analogiques/numériques (CAN) et CNA.

Dans de ce chapitre, nous allons nous restreindre aux CAN en présentant leur principe de fonctionnement, ainsi que leurs principales familles.

1.2. Échantillonnage des données

L'échantillonnage est la première étape dans la transformation d'un signal analogique $x(t)$ en un signal numérique $x(n)$. Cette opération consiste à remplacer le signal analogique par une suite de valeurs ponctuelles

$$x(n) = x(nT_e) \; ; \; n = 0, \; 1, \; ..., \; \infty \tag{1.1}$$

où T_e est la période d'échantillonnage séparant les différentes valeurs et n le nombre de période d'échantillonnage.

Le signal échantillonné peut alors être approximé par un signal $x_e(t)$ qui est égal à $x(t)$ pendant de brefs instants de durée τ et de période T_e, tel que

$$f_e = \frac{1}{T_e} \tag{1.2}$$

où f_e est appelée aussi fréquence d'échantillonnage qui doit satisfaire les critère énoncés par le théorème de Nyquist-Shannon de sorte que la fréquence d'échantillonnage f_e d'un signal doit être égale ou supérieure au double de la fréquence maximale contenue dans ce signal, afin de convertir ce signal d'une forme analogique à une forme numérique.. $x_e(t)$ est nul autrement.

En pratique, le signal $x_e(t)$ est obtenu en échantillonnant le signal $x(t)$ avec un circuit échantillonneur/bloqueur (E/B) actionné par une impulsion de durée τ et de période T_e comme le montre la Figure 1.2. τ Défini le temps d'acquisition du signal analogique par l'échantillonneur bloqueur. Pour cette fin, elle doit être suffisamment grande pour permettre aux capacités des E/B de se charger et décharger.

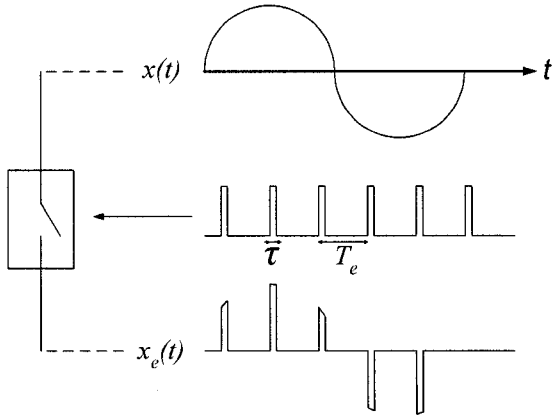


Figure 1.2. Principe de l'échantillonnage

1.3. Principe de fonctionnement des CAN

Un CAN convertit un signal analogique en un signal numérique. Cette opération se fait dans certains cas en deux étapes : la quantification puis le décodage.

La quantification transforme un signal analogique continu en une suite finie de valeurs numériques. La Figure 1.3 montre le principe de quantification : chaque état couvre un intervalle de valeurs analogiques de largeurs Q_N, appelé pas de quantification ou quantum. Lorsque les pas de quantification sont tous égaux (c'est loin d'être le cas étant donné que cela implique une reproduction parfaite des circuits analogiques correspondant à chaque niveau), la quantification est dite uniforme. C'est cette étape de quantification qui fixe en partie le temps et la précision de la conversion mais elle introduit une certaine erreur dans le résultat final. En fait si on cherche à avoir un code binaire de N bits il nous faudra 2^N -1 états de quantification. Un code binaire est une représentation codée du signal analogique.

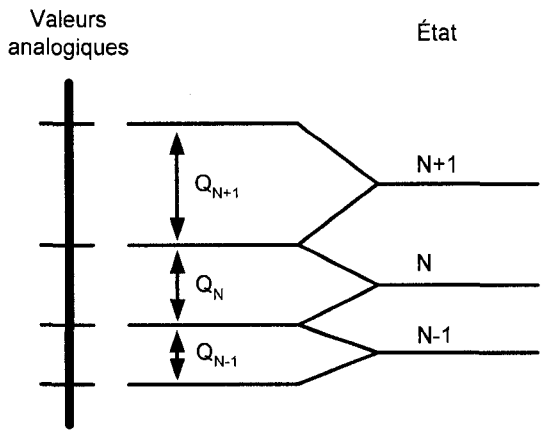


Figure 1.3. Quantification des données dans un CAN

Le décalage binaire et le complément à deux sont fréquemment utilisés dans la génération des codes dans les CAN. Citons également le code Gray qui est utilisé de temps à autre [10].

En outre, un CAN est défini par plusieurs paramètres dont la résolution, l'exactitude, la fréquence d'échantillonnage, le temps de conversion, la tension de références, son offset, le temps d'établissement... Mais puisque notre projet de recherche concerne uniquement la partie de décodage des CAN Sigma-Delta, seuls quelques paramètres ont un impact sur cette partie. Ces derniers sont introduits dans la section suivante.

- **Résolution**

C'est la plus petite variation de l'entrée qui fait changer la sortie numérique. Elle peut s'exprimer en pourcentage de la pleine échelle ou en nombre de bits.

$$Résolution = \frac{V_{max}}{2^N} = \Delta \tag{1.3}$$

V_{max} étant la tension maximale que peut détecter le CAN et N le nombre de bits représentant le signal de sortie.

- **Exactitude : Nombre effectif de bits ENOB**

Ce paramètre illustre la précision effective de votre convertisseur une fois les différentes erreurs additionnées. Le calcul de ENOB est basé sur la valeur du Rapport signal sur bruit et distorsion SINAD et est donnée en retournant la formule : SNR = 6.02•N +1.76 dB , ce qui donne : ENOB= (SINAD-1.76dB) / 6.02. Sachant que le SINAD est le rapport entre la valeur RMS du signal sinus de test et la valeur RMS de la somme des amplitudes de toutes les autres fréquences présentes dans le signal sauf la tension continue.

- **Erreur de quantification**

La quantification introduit une erreur (bruit) qui correspond à une différence entre la valeur analogique et la valeur numérique codée. Ce bruit varie en dent de scie et est donné par l'équation (1.4).

$$Erreur\ de\ quantification = \pm\frac{1}{2} LSB = \pm\frac{\Delta}{2} \qquad (1.4)$$

Le LSB étant le bit le moins significatif du mot binaire ou code binaire généré par le CAN.

- **Temps d'établissement**

C'est le temps nécessaire au convertisseur pour répondre à une variation pleine échelle du signal d'entrée. C'est un paramètre important si plusieurs signaux sont multiplexés pour être traités par le même CAN.

D'autres paramètres existent pour les CAN tels que l'erreur de décalage, l'erreur du gain, la distorsion harmonique, le rapport signal sur bruit (SNR) qui sont expliqués en détails dans [11]. Tous ces paramètres ont un impact sur le bruit de quantification et la résolution du convertisseur.

Suite à cette brève présentation du principe de fonctionnement des CAN nous exposons les grandes familles de ceux qui sont fréquemment utilisés, leurs avantages, inconvénients ainsi que leur domaine d'utilisation dans la prochaine section

1.4. Principaux types de CAN

Plusieurs types de CAN ont été développés par les chercheurs selon leur application. Certains s'adaptent le mieux pour des applications à hautes fréquences, d'autres pour des applications exigeant une faible consommation, dans d'autre cas c'est la précision et l'exactitude du CAN qui sont importantes. Pour justifier notre choix des CAN ΣΔ, il est important de présenter quelques principales architectures de CAN et leurs particularités ainsi que leurs domaines d'application, ce qui est le sujet de la section suivante.

1.4.1. CAN à approximation successive

La technique utilisée par cette famille de CAN consiste à approximer l'entrée analogique à des valeurs binaires en utilisant un CAN tel que montré dans la Figure 1.4. En fait, un registre de n bits renferme une valeur initiale « 0 ». Par la suite le bit de poids le plus fort est mis à « 1 ». Le contenu du registre (SAR) est envoyé à un CNA qui le convertit en une valeur analogique. Le signal analogique obtenu sera comparé au signal d'entrée. Si ce dernier est inférieur à la valeur fournie par le CNA, le bit ayant été mis à « 1 » est remis « 0 », et le bit suivant est mis à « 1 » et on refait la comparaison. Ainsi de suite jusqu'à ce qu'on atteigne le bit moins significatif (LSB). Ainsi un bit donné ne peut garder un « 1 » que si la valeur analogique fournie par le CNA est inférieure à la valeur du signal analogique d'entrée. En conséquence, il faudra n étapes de comparaison successives pour générer la valeur numérique du signal analogique.

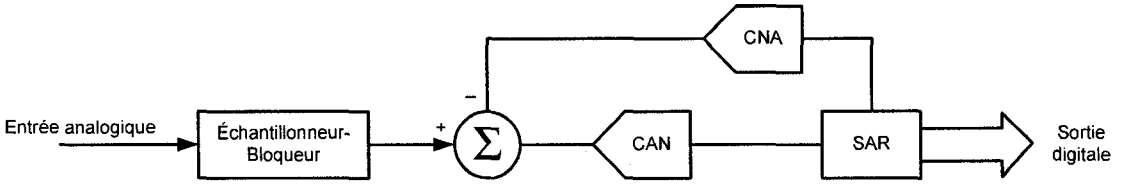


Figure 1.4. Architecture du CAN à approximation successive

Deux signaux caractérisent ce type de CAN : le signal de début de conversion et le signal de fin de conversion. Les CAN à approximation successive sont relativement rapides, précis et fournissent souvent une sortie de n-bit parallèles et quelques fois en série en commençant par le MSB sur une seule ligne de sortie.

Ce genre de convertisseurs est sensible au décalage du comparateur et à la linéarité du CNA. Le temps de conversion s'échelonne entre 1µs et 50µs pour une précision standard de huit à douze bits. Ce CAN parait peu sensible à la variation de la tension durant la période de conversion cependant cette dernière peut conduire à une perturbation si elle est inférieure au LSB [12]. Par contre les pics de tension très élevée et de très haute fréquence à l'entrée sont complètement désastreux et peuvent conduire à un résultat complètement erroné.

Cette famille de CAN est utilisée surtout pour les signaux non périodiques vu que nous pouvons activer la conversion à un instant désiré et ignorer le reste. Ils sont également adaptés pour les signaux à basse fréquence étant donné que la conversion prend assez de temps (n cycles de comparaison, n étant le nombre de bits à la sortie).

Lors de l'utilisation des SAR il faut absolument faire attention au crènelage (chevauchement spectral : phénomène qui est du à la présence de fréquences non souhaitées lorsque nous travaillons avec les hautes fréquences). L'utilisation des filtres passe-bas s'avère très utile dans des cas pareils pour supprimer les hautes fréquences indésirables.

Plusieurs recherches mettent en évidence l'impact du bruit sur les CAN de type SAR et traitent en détail ce phénomène [14]. D'autres recherches ont été effectuées et sont encore en cours pour améliorer les caractéristiques de ce genre de CAN comme la réduction de la puissance consommée en gardant une bonne résolution et une fréquence d'échantillonnage relativement élevée [15]-[17].

1.4.2. Convertisseur Parallèle

C'est de loin le convertisseur le plus rapide grâce à son architecture parallèle. L'architecture de ce dernier est présentée à la Figure 1.5.

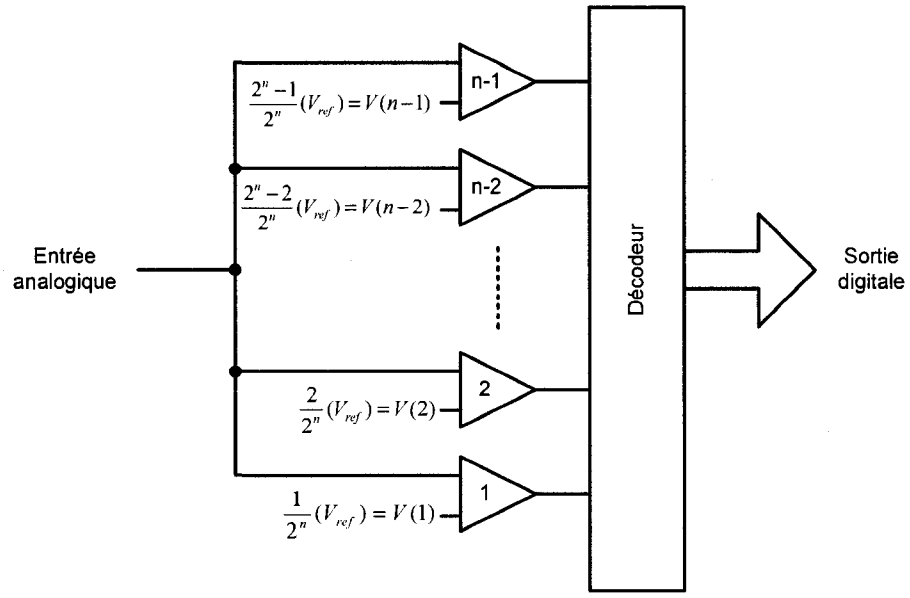


Figure 1.5. Architecture de base d'un CAN parallèle

Le fonctionnement de ce convertisseur repose sur la comparaison de la valeur d'entrée par rapport à différentes valeurs de référence à l'aide de plusieurs comparateurs connectés en parallèle. Il existe 2^n-1 comparateurs pour une résolution de n bits. Une des entrées du comparateur est connectée au signal d'entrée l'autre est connectée à la tension de référence *v(N)*. Cette dernière est générée à parti d'un réseau de résistances jouant le rôle d'un diviseur de tension À chaque comparateur est associé une tension de

référence différente selon la Figure 1.5. Les sorties de tous les comparateurs pour lesquels la tension d'entrée est supérieure à la tension de référence adéquate $v(p)_{0<p<n}$ reçoivent simultanément un « 1 » logique alors que les autres reçoivent un « 0 ». Le décodeur convertit les 2^n-1 sorties des comparateurs en un mot binaire sur n bits. Ce décodeur est un circuit combinatoire de transcodage. Ainsi le temps de conversion qui est le temps de propagation du signal à travers les comparateurs et le décodeur est de l'ordre de 10 ns à quelques 10^2 ns [12].

Les inconvénients d'un tel convertisseur sont l'espace important qu'il occupe dans une puce électronique pour avoir une grande résolution et nécessitant des répliques exactes de tous les comparateurs. Ainsi plus on ajoute un bit dans la résolution plus la surface est grande. À titre d'exemple pour une résolution de 8 bits on doit avoir 255 comparateurs et si on veut élever la résolution à 9 bits on doit multiplier cette surface par deux. Donc peu importe la technologie, bipolaire ou CMOS, l'espace occupé reste toujours important. Un autre inconvénient, non négligeable, c'est la charge d'entrée qui est relativement importante vu l'architecture parallèle de ce CAN et par suite une consommation assez importante. Cette capacité élevée en entrée limite la précision du CAN puisque l'entrée n'arrive plus à être chargée correctement pour procéder à sa conversion, mais généralement ceci ne peut avoir lieu que dans le cas des très hautes fréquences. De plus ce CAN n'est pas approprié pour des applications où le signal change de façon relativement lente puisqu'il échantillonnera la même valeur durant un intervalle assez long en fonctionnant en hautes fréquences ce qui conduit à un problème de redondance dans le signal [15].

1.4.3. Convertisseur Sigma-Delta

Tout au long de cette section nous décrirons brièvement le fonctionnement du convertisseur ΣΔ. Une présentation détaillée de ce dernier fera l'objet du $2^{ème}$ chapitre.

Principalement un convertisseur ΣΔ est formé de deux modules :

- Un module analogique qui convertit un signal analogique en une suite de bits.
- Un filtre numérique qui convertit cette suite de bits en une valeur numérique sur n bits dépendamment de la résolution du convertisseur.

Typiquement le principe de base est le sur échantillonnage, ainsi le rôle du filtre est de:

- Ramener la fréquence de sur-échantillonnage à celle de Nyquist, opération connue sous le nom de décimation.
- Éliminer le bruit du à la quantification, (la quantification est une opération effectuée à l'intérieur du module analogique et qui sera aussi décrite avec plus de détails dans le chapitre suivant).

En d'autres termes, les filtres ne font que la moyenne des bits d'une séquence donnée : Si on suppose que, suite à une entrée analogique donnée, le modulateur ΣΔ fournit une séquence de 8192 bits, la moyenne de tous ces bits n'est autre que la valeur du signal analogique. Cette approche est correcte cependant on n'a pas tenu compte du bruit introduit par le modulateur, d'où la nécessité d'un prétraitement du signal pour supprimer le bruit et pour aller chercher l'information utile [11].

Les filtres sont demeurés longtemps l'outil de prédilection pour les CAN ΣΔ. Cependant avec l'apparition des nouvelles technologies comme les DSP et les FPGA, de nouvelles approches ont été développées parmi ces dernières il y a les algorithmes de décodage de la séquence générée par le modulateur ΣΔ. Le décodage a pu voir le jour pour améliorer certains paramètres comme le SQNR (Signal Quantized to Noise Ratio) qui est le rapport signal sur bruit appliqué pour un signal numérique. L'inconvénient de ce décodage est que le traitement du signal requis nécessite énormément de ressources. L'idée qui a poussé les chercheurs à s'orienter vers le décodage est le fait que la séquence de bits générée par le modulateur ΣΔ renferme beaucoup plus d'informations

qu'un filtre numérique ne peut en extraire. En éliminant le bruit par l'opération de filtrage on enlève également une partie de l'information utile. D'où l'idée d'utiliser un décodage au lieu d'un filtrage. Cependant une certaine controverse s'est installée : certains chercheurs considèrent que le décodage est équivalent à la décimation et d'autres le considèrent comme une opération à part entière. Étant donné que cette technique demeure encore dans le domaine de la recherche, plusieurs algorithmes de décodage ont été proposés principalement pour les ΣΔ de premier ordre [31]-[34]. La limite au décodage réside dans la complexité d'implémentation des décodeurs. À l'état actuel de la recherche, les décodeurs sont assez efficaces pour les ΣΔ de 1er ordre et pour les signaux de basses fréquences ou constants. Pour des signaux périodiques, les filtres demeurent encore plus performants que les décodeurs. Cependant, une étude détaillée des filtres et des décodeurs est présentée dans le 2ème chapitre.

De plus nous trouvons plusieurs ordres de CAN ΣΔ [19]-[23]. L'ordre d'un ΣΔ correspond au nombre de boucles de rétroactions dans le modulateur (un modulateur de 1er ordre nous indique qu'il n'y a qu'une seule boucle de rétroaction dans le modulateur). À titre purement informatif, à la Figure 1.6 le modèle d'un ΣΔ de 2ème ordre est représenté, le nombre d'intégrateurs est égal à l'ordre du convertisseur.

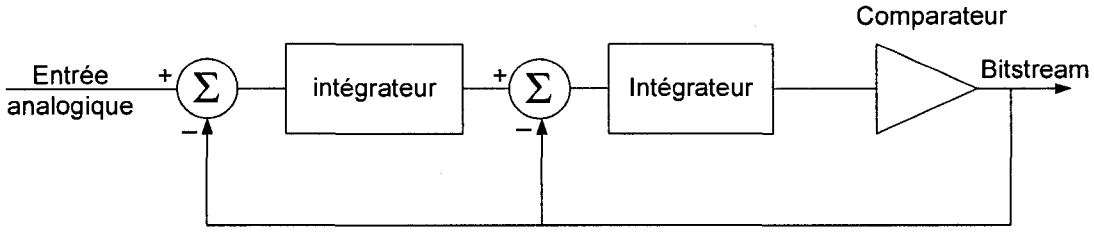


Figure 1.6. Modèle d'un convertisseur Sigma-Delta de 2ème ordre

1.4.4. Convertisseur pipeline

Cette structure s'adapte le mieux pour les circuits fonctionnant à haute vitesse. Le « pipelinage » est une approche qui a souvent attiré les chercheurs et les concepteurs parce qu'elle permet de lancer plusieurs opérations en même temps. L'exécution de chaque phase est assurée par une unité fonctionnelle élémentaire du processus appelée étage. La durée d'une phase correspond à un cycle d'opérations. L'idée du pipeline consiste à coordonner toutes les phases pour qu'elles fonctionnent en parallèle en exécutant différentes tâches. Ce même principe a été appliqué pour les convertisseurs.

Un premier module permet de générer les bits les plus significatifs, un deuxième les bits intermédiaires, alors que le premier continue à générer les bits les plus significatifs de l'échantillon suivant. Un troisième étage permet de compléter la série en générant les bits les moins significatifs alors que les autres étages continuent à convertir des échantillons plus récents. En fin du compte, passé le temps de latence au début qui nécessite trois cycles de conversion, à chaque cycle on a une nouvelle valeur du signal digital qui correspond à une entrée analogique.

Le principal inconvénient d'une telle architecture est la nécessité de la présence d'échantillonneurs/bloqueurs au niveau de chaque étage. La réalisation de ces derniers est plus délicate, ce qui représente la limitation majeure de cette architecture [10], [24]-[26].

1.5. Comparaisons des CAN

À ce stade il est utile de faire une comparaison entre les convertisseurs présentés dans ce chapitre. Le Tableau 1.1 résume les caractéristiques principales de ces convertisseurs.

Tableau 1.1 Caractéristiques des principales familles des CAN [27]

Famille du CAN / Caractéristiques	Parallèle	Pipe-line	SAR	ΣΔ
Fréquence d'échantillonnage	1*	2	3	4
Résolution	4	3	2	1
Latence	1	3	2	4
Multiplexage de plusieurs entrées	1	2	1	3
Conversion des signaux non périodiques	1	2	1	3

*1 : Très bon, 2 : Bon, 3 : Moyen, 4 : Mauvais

Comme nous pouvons bien le remarquer, aucun convertisseur ne monopolise la tête du classement dans toutes les caractéristiques et chaque CAN possède son propre domaine d'application. Ainsi les CAN parallèles sont utilisés pour des applications nécessitant une fréquence d'échantillonnage très élevée sans donner une grande importance à la résolution. Ils permettent également le multiplexage de plusieurs entrées analogiques. Par contre les CAN ΣΔ ont une résolution très élevée ce qui leur donne la possibilité d'être utilisés dans des applications nécessitant une grande précision pour des signaux analogiques de faible fréquence par rapport aux autres types de CAN. L'objectif de notre application est d'avoir des signaux très précis. Les CAN ΣΔ permettent d'atteindre cet objectif, cependant ceci n'est pas la seule et l'unique raison. L'architecture du laboratoire sur puce qui est un capteur capacitif est sous forme matricielle. Ce qui implique la nécessité d'implémenter plusieurs CAN. Ainsi il est primordial d'opter pour un CAN de faible complexité. Notre choix s'est fixé sur les CAN ΣΔ, parce qu'ils offrent une grande précision, mais en plus ils disposent de plusieurs configurations de complexité variable. Parmi ces configurations, il y a le CAN ΣΔ de premier ordre qui occupe un espace très faible, tout en offrant une très bonne précision. Toutefois, pour expliquer le fonctionnement d'un tel convertisseur et comprendre son architecture, une description plus détaillée de ce dernier est présentée dans le deuxième chapitre.

1.6. Conclusion

Tout au long de ce chapitre, nous avons présenté les différents types de CAN, nous avons décri brièvement leur architecture et leur principe de fonctionnement. Cependant, il existe quelques CAN qui sortent du commun vu qu'ils sont conçus pour des applications bien particulières. À titre d'exemple nous trouvons des CAN fonctionnant à une fréquence d'échantillonnage de 20 échantillons par seconde ou bien d'autres qui sont développé pour des applications vidéo. Certains CAN n'offrent pas une grande flexibilité par rapport à l'évolution technologique et sont voués à la disparition avec l'apparition de nouveaux procédés. Par contre d'autres s'adaptent parfaitement bien en améliorant leurs performances en utilisant d'autres approches, tel est le cas pour les CAN $\Sigma\Delta$ qui ne cessent de s'améliorer en développant des architectures offrant de meilleures performances. Mais avant tout il est primordial de bien comprendre le fonctionnement d'un CAN avant de procéder à son amélioration ce qui fera l'objet du deuxième chapitre dans lequel une description plus détaillée du convertisseur Sigma-Delta est présentée.

Chapitre 2
Le décodage dans les convertisseurs Sigma-Delta

2.1. Introduction

Suite à la présentation de principaux convertisseurs analogiques numériques au chapitre 1, nous nous intéressons dans ce chapitre aux CAN ΣΔ. Quel est l'intérêt d'utiliser ces derniers? Pourquoi sont-ils plus performants que d'autres dans des applications particulières et surtout pourquoi le décodage et le filtrage dans le cas de ces convertisseurs est d'une importance capitale?

Tout au long de ce chapitre, nous répondrons à ces questions et nous montrerons à travers des différents exemples que le décodage dans les CAN ΣΔ permet d'améliorer les performances de ces convertisseurs et en particulier ceux de 1^{er} ordre.

2.2. Convertisseurs Sigma-Delta

Comme nous l'avons déjà mentionné dans le chapitre 1, le principe de fonctionnement d'un convertisseur ΣΔ est le sur-échantillonnage. Le modulateur fournit en sortie un signal sur 1 bit. Le signal obtenu renferme des données relatives au signal analogique modulé mais aussi du bruit provenant du quantificateur. D'où la nécessité d'utiliser un filtre numérique pour supprimer ce bruit tout en déplaçant la fréquence du signal dans le domaine de Nyquist. La fonction du filtre numérique et des algorithmes de décodage est d'augmenter la résolution des CAN ΣΔ tout en éliminant l'erreur, ce qui est primordial dans des applications de pointe telles que les applications biomédicales.

La Figure 2.1 montre un diagramme bloc simplifié d'un CAN ΣΔ dans la partie (a) et représente en (b) le schéma typique de son modulateur.

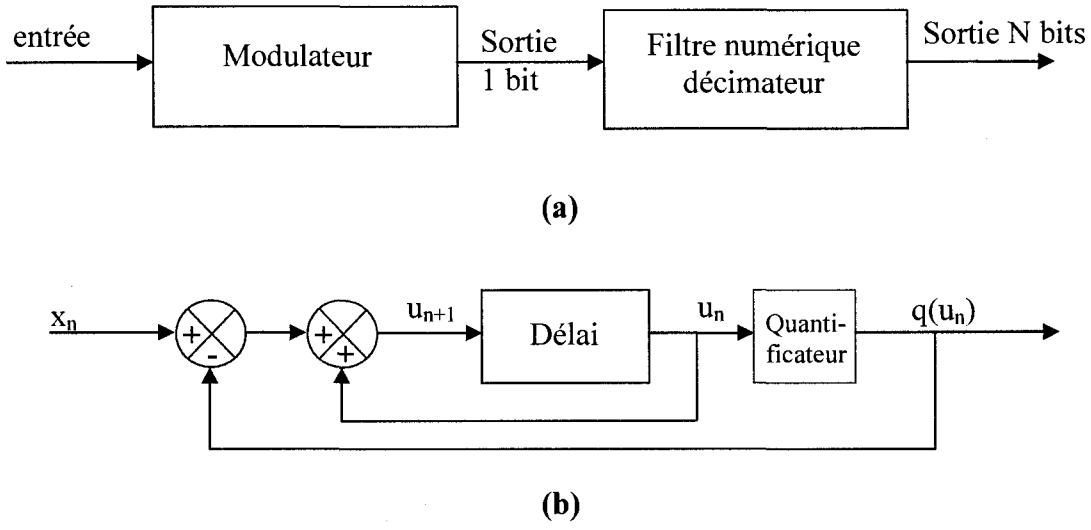


Figure 2.1. Convertisseur ΣΔ: (a) diagramme bloc, (b) contenu du modulateur

Notons que u_n correspond à l'échantillon n à un instant t du signal analogique.

La figure 2.2 montre le principe de fonctionnement d'un CAN ΣΔ. $+U$ et $-U$ sont les sorties du quantificateur comme montré dans l'équation (2.1).

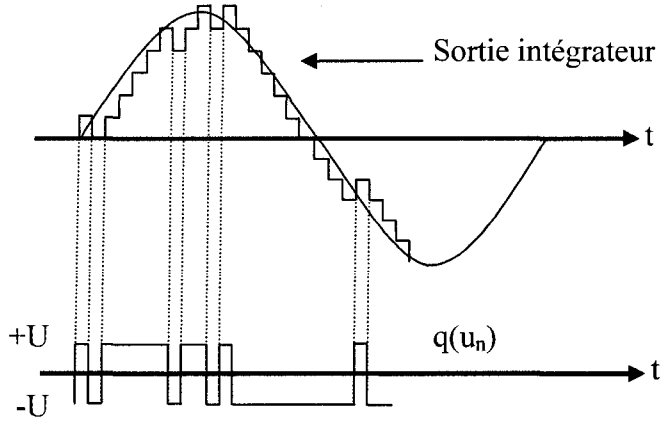


Figure 2.2. Principe de fonctionnement des convertisseurs ΣΔ

$$q(u_n)=\begin{cases}+U; & 0\le u;\\ -U; & u<0;\end{cases} \tag{2.1}$$

Comme montré dans la figure 2.1 les CAN ΣΔ sont formés de plusieurs blocs dont le plus important est le quantificateur. Ci-dessous, la définition mathématique d'un bloc quantificateur.

$$q(u)=\begin{cases}\left(\frac{M}{2}-\frac{1}{2}\right)\Delta; & \left(\frac{M}{2}-\frac{1}{2}\right)\Delta\le u;\\ \left(k-\frac{1}{2}\right)\Delta; & (k-1)\Delta\le u\le k\Delta \text{ avec } k=\left(-\frac{M}{2}+\frac{1}{2}\right),\ldots,\left(\frac{M}{2}-\frac{1}{2}\right);\\ \left(-\frac{M}{2}+\frac{1}{2}\right)\Delta; & u\le\left(\frac{M}{2}-\frac{1}{2}\right)\Delta;\end{cases} \tag{2.2}$$

où M est le nombre de niveaux existants dans un quantificateur, Δ la distance séparant deux niveaux successifs ce qui est l'équivalent de Q_N à la figure 1.3. Elle doit être la plus faible possible pour avoir le minimum d'erreur, u étant la valeur du signal analogique. Nous pouvons également définir l'erreur introduite par le bloc quantificateur comme suit :

$$\varepsilon = q(u)-u \tag{2.3}$$

cette caractéristiques q(u) est présentée à la figure 2.3 pour M = 8 et $\Delta=\frac{1}{4}$.

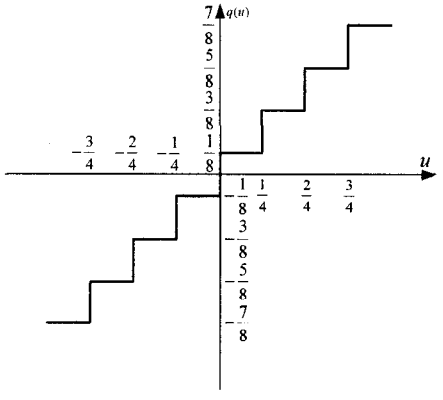


Figure 2.3. Principe de la quantification

Après avoir défini le fonctionnement de chaque partie du convertisseur nous pouvons retrouver l'équation qui régit le fonctionnement du modulateur ΣΔ de 1er ordre.

$$u_n = x_{n-1} - \varepsilon_{n-1} = u_{n-1} + x_{n-1} - q(u_{n-1}); \qquad n = 1,2,... \tag{2.4}$$

où x_n est l'échantillon n du signal analogique à l'entrée du modulateur et ε_n l'erreur de quantification

Et selon l'équation (2.2) on a :

$$\varepsilon_n = q(u_n) - u_n \tag{2.5}$$

d'où

$$q(u_n) = x_{n-1} + \varepsilon_n - \varepsilon_{n-1} \tag{2.6}$$

L'information qui précède n'est applicable que pour les ΣΔ de 1er ordre, c'est-à-dire ne contenant qu'une seule boucle d'asservissement. Il est à noter qu'un Sigma-Delta d'ordre n correspond à un convertisseur avec n boucles d'asservissement.

À titre d'exemple, un modulateur ΣΔ de 2ème ordre est régit par l'équation (2.7).

$$q(u_n) = \varepsilon_n + u_n = \varepsilon_n - 2\varepsilon_{n-1} + \varepsilon_{n-2} + x_{n-1} \tag{2.7}$$

Dans ce qui suit, nous nous limitons à un modulateur ΣΔ de 1er ordre avec un signal d'entrée DC qui constitue notre principal intérêt.

Ainsi le quantificateur, peut être représenté par l'équation (2.8) en remplaçant U par b dans l'équation (2.1) :

$$q(u_n) = \begin{cases} -b, & \text{si } u_n \le 0 \\ +b, & \text{si } u_n > 0 \end{cases} \tag{2.8}$$

Cette équation régissant le convertisseur du 1er ordre demeure la même, cependant le résultat de cette dernière dépend des valeurs initiales. Cette équation limite

également le nombre de séquences pouvant être générées par un modulateur ΣΔ. Pour expliquer cette théorie nous allons utiliser l'équation (2.9) qui représente le fonctionnement du modulateur ΣΔ de 1er ordre mais sous une autre forme [28].

$$U_0 \in (X-b, X+b) \Rightarrow U_n \in (X-b, X+b) \quad \text{pour tout } n>0 \qquad (2.9)$$

où U_0 est la valeur initiale de l'intégrateur et l'entrée du modulateur. L'équation (2.9) reste valide quelque soit n. X représente l'échantillon x_n de l'équation (2.4).

Cependant les différentes combinaisons de séquences pouvant être générées par un modulateur ΣΔ sont limitées. Pour déterminer le nombre maximal de codes ou de séquences pouvant être générés par un modulateur, ainsi que le code lui-même, nous pouvons nous référer à [29]. En revenant aux équations (2.4), (2.8) et (2.9) nous pouvons constater que X est décrémentée de b - X à chaque pas si $U_{n-1} > 0$, par contre si $U_{n-1} < 0$ elle est incrémentée de $b + X = 2b - (b - X)$. Ainsi nous remarquons que X est toujours décrémentée et que si $U < 0$ elle est incrémenté de $2b$. Bien sûr l'équation (2.9) doit être toujours valide. D'où la nécessité de la condition suivante :

$$X-b \le U_0 - n(b-X) + 2jb \le X+b \qquad (2.10)$$

où j est le nombre de bits négatifs, représenté par la valeur –b dans l'équation (2.8).

L'intervalle dynamique du quantificateur [-b, +b] peut être divisé en un ensemble d'intervalles dépendamment de U_0 ainsi toutes les entrées DC situées à l'intérieur d'un même sous-intervalle génèrent le même code, le seuil de transition entre chaque intervalle est défini par l'équation suivante :

$$X = \frac{pb - (2jb + U_0)}{p} \qquad (2.11)$$

où p est entier qui est donné par

$$1 \le p \le N-1 \qquad (2.12)$$

N étant le nombre de bits caractéristiques du CAN et j un entier naturel tel que $X \in (-b, +b)$.

Ainsi p et j déterminent le nombre de sous-intervalles ainsi que le seuil - c'est-à-dire la largeur des intervalles. Le tableau 2.1 met en évidence la théorie expliquée précédemment en montrant quelque exemple de point de transition (limites des intervalles) correspondant à un couple (p, j) déterminé.

Tableau 2.1. Points de transitions pour N=12 et U_0=0

p \ j	1	2	3	4	5	6	7	8	9	10
2	0									
3	1/3	-1/3								
4	1/2	0	-1/2							
5	3/5	1/5	-1/5	-3/5						
6	2/3	1/3	0	-1/3	-2/3					
7	5/7	3/7	1/7	-1/7	-3/7	-5/7				
8	3/4	1/2	1/4	0	-1/4	-1/2	-3/4			
9	7/9	5/9	3/9	1/9	-1/9	-3/9	-5/9	-7/9		
10	4/5	3/5	2/5	1/5	0	-1/5	-2/5	-3/5	-4/5	
11	9/11	7/11	5/11	3/11	1/11	-1/11	-3/11	-5/11	-7/11	-9/11

Les valeurs du Tableau 2.1 sont calculées à partir de l'équation (2.11). Ces valeurs sont normalisées par rapport à b. puisque $X \in (-b,+b)$ alors ces valeurs appartiennent à l'intervalle]-1,1[. Par exemple si p=5 et j=3 et U_0=0 alors nous aurons :

$$X = \frac{5b - 6b}{5} = -\frac{1}{5}b \tag{2.13}$$

le nombre maximum de codes générés est déterminé par les équations suivantes et ceci en calculant le nombre de combinaison possible de (p,j).

$$C_{\max} = \begin{cases} \frac{1}{2}N(N-1)+1 & ,U_0 \neq 0 \\ \frac{1}{2}N(N-1)(N-2)+1 & ,U_0 = 0 \end{cases} \tag{2.14}$$

Cependant le nombre réel de codes pouvant être générés pratiquement est inférieur à celui défini par l'équation (2.14) vu que certains paramètres peuvent conduire à un même point de transition [29]. Le tableau 2.2 présente quelques exemples de codes de 12 bits générés par un modulateur ΣΔ de 1er ordre

Tableau 2.2. Exemples de codes générés par un modulateur ΣΔ

Point de transition	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*
1	-	+	+	+	+	+	+	+	+	+	+	+
9/11	-	+	+	+	+	+	+	+	+	+	+	-
4/5	-	+	+	+	+	+	+	+	+	+	-	+
7/9	-	+	+	+	+	+	+	+	+	-	+	+
3/4	-	+	+	+	+	+	+	+	-	+	+	+
5/7	-	+	+	+	+	+	+	-	+	+	+	+
2/3	-	+	+	+	+	+	-	+	+	+	+	+
7/11	-	+	+	+	+	+	-	+	+	+	+	-
3/5	-	+	+	+	+	-	+	+	+	+	-	+
5/9	-	+	+	+	+	-	+	+	+	-	+	+
1/2	-	+	+	+	-	+	+	+	-	-	+	+
5/11	-	+	+	+	-	+	+	+	-	+	+	-
3/7	-	+	+	+	-	+	+	-	+	+	+	-
2/5	-	+	+	+	-	+	+	-	+	+	-	+
1/3	-	+	+	-	+	+	-	+	+	+	+	+
3/11	-	+	+	-	+	+	-	+	+		+	-
1/4	-	+	+	-	+	+	-	+	-	+	+	-
1/5	-	+	+	-	+	-	+	+	-	+	-	+
1/7	-	+	+	-	+	-	+	-	+	+	-	+
1/9	-	+	+	-	+	-	+	-	+	-	+	+
1/11	-	+	+	-	+	-	+	-	+	-	+	-
0	-	+	-	+	-	+	-	+	-	+	-	+
-1/11	-	+	-	+	-	+	-	+	-	+	-	-
-1/9	-	+	-	+	-	+	-	+	-	-	+	-
-1/7	-	+	-	+	-	+	-	-	+	-	+	-
-1/5	-	+	-	+	-	-	+	-	+	-	-	+
-1/4	-	+	-	+	-	-	+	-	-	+	-	+
-1/3	-	+	-	-	+	-	-	+	-	-	+	-
-2/5	-	+	-	-	+	-	-	+	-	-	-	+
-3/7	-	+	-	-	+	-	-	-	+	-	-	+
-5/11	-	+	-	-	+	-	-	-	+	-	-	-
-1/2	-	+	-	-	-	+	-	-	-	+	-	-
-5/9	-	+	-	-	-	+	-	-	-	-	+	-
-3/5	-	+	-	-	-	-	+	-	-	-	-	+
-7/11	-	+	-	-	-	-	+	-	-	-	-	-
-2/3	-	+	-	-	-	-	-	+	-	-	-	-
-5/7	-	+	-	-	-	-	-	-	+	-	-	-
-3/4	-	+	-	-	-	-	-	-	-	+	-	-
-7/9	-	+	-	-	-	-	-	-	-	-	+	-
-4/5	-	+	-	-	-	-	-	-	-	-	-	+
-9/11	-	+	-	-	-	-	-	-	-	-	-	-

2.3. Le bruit de quantification

L'expression générale de l'erreur introduite par le quantificateur dans le cas d'un CAN ΣΔ de 1er ordre est :

$$e_n = \frac{1}{2} - \left\langle \frac{n}{2} + \sum_{k=0}^{n-1} \frac{x_k}{\Delta} \right\rangle \qquad (2.15)$$

où $\left\langle \frac{n}{2} + \sum_{k=0}^{n-1} \frac{x_k}{\Delta} \right\rangle$ représente la partie rationnelle de $\left(\frac{n}{2} + \sum_{k=0}^{n-1} \frac{x_k}{\Delta} \right)$, qui peut être exprimée autrement (r mod 1), de plus nous avons :

$$\Delta = \frac{2b}{M-1} \tag{2.16}$$

et b correspond à l'intervalle dynamique du quantificateur [$-b$, b] et M est le nombre de sous-intervalles.

Dans le cas d'une entrée DC x telle que $-b \leq x < b$, le bruit introduit a les caractéristiques suivantes:

$$\bar{E}\{e_n\} = 0 \tag{2.17}$$

avec

$$\bar{E}\{x_n\} = \lim_{N \to \infty} \frac{1}{N} \sum_{n=1}^{N} E(x_n) \tag{2.18}$$

E(x_n) étant l'espérance de x_n et

$$\bar{E}\{e_n^2\} = \frac{1}{12} \tag{2.19}$$

Ce qui fait que le bruit est considéré comme uniforme et son spectre de puissance serait :

$$S_n = \begin{cases} 0; & si\ n = 0 \\ \dfrac{1}{(2\pi n)^2}; & si\ n \neq 0 \end{cases} \tag{2.20}$$

La figure 2.4 montre la variation de l'erreur de quantification. Cette figure est obtenue à partir de l'équation (2.15). Dans ce cas de figure l'entrée est égale à 0.374313 et une longueur de code de 1024, la valeur moyenne du signal est -0.002296 et la variance est de 0.083420.

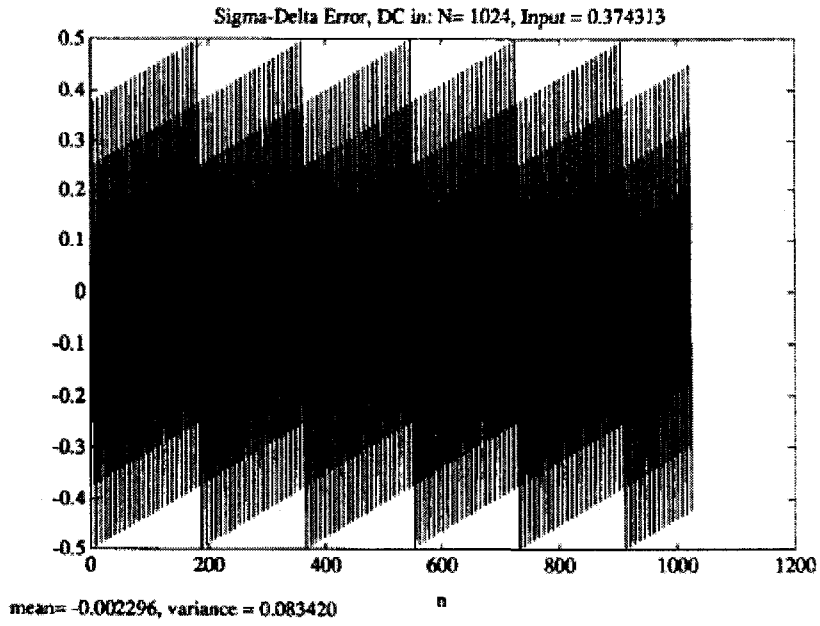


Figure 2.4. Erreur de quantification pour une entrée DC

Cette explication théorique est un résumé des points les plus importants cependant des détails plus approfondis figurent dans [31].

Nous remarquons que l'erreur est un paramètre intrinsèque au quantificateur, d'où l'idée de développer un système capable de compenser ou d'éliminer cette dernière. Plusieurs approches ont été adoptées dont la décimation qui fait l'objet de la section suivante.

2.4. La décimation dans les modulateurs ΣΔ

Étant donné que les modulateurs ΣΔ sont basés sur le principe de sur-échantillonnage, nous avons besoin d'une technique complémentaire au sur-échantillonnage pour retracer l'information adéquatement. Cette technique s'appelle la décimation, dont le but est de déplacer la fréquence d'échantillonnage vers la fréquence désirée (celle de Nyquist). Nous allons expliquer brièvement et mathématiquement les raisons de la nécessité de la décimation.

Pour cette fin nous avons repris la transformé en z de l'expression du ΣΔ de premier ordre selon [31]:

$$Y(z) = z^{-1}X(z) + (1 - z^{-1})E(z) \tag{2.21}$$

Sachant que

$$E(f) = \bar{e}\sqrt{2\tau} = \sigma\sqrt{\frac{\tau}{6}} \tag{2.22}$$

où σ est l'écart type du quantificateur et $\bar{e}$ est le « bruit rms » et τ le temps d'échantillonnage.

La densité spectrale de puissance du bruit est alors

$$N(f) = 2\bar{e}\sqrt{2\tau}\sin(\pi f \tau) \tag{2.23}$$

L'équation (2.23) montre que le bruit dépend de la fréquence. De plus on a remarqué que dans les hautes fréquences le bruit a effet désastreux, c'est pour cette raison que nous évitons d'échantillonner avec des fréquences élevées. Cependant dans le cas du ΣΔ nous ne pouvons faire autrement d'où la nécessité d'appliquer un processus capable de minimiser ce bruit de décimation.

Selon [31] suite à la décimation, tous les bruits provenant du circuit analogique et ceux qui résultent de la quantification s'additionnent, c'est pour cette raison qu'il est nécessaire d'utiliser une technique de filtrage qui permet d'éliminer les bruits avant de procéder à la décimation. Les filtres passe bas constituent une solution à ce problème elle peut être très coûteuse

Le résultat de la décimation donne un nouveau code qui est plus long que le code fourni par le quantificateur. Cependant la fréquence d'échantillonnage est plus faible.

Ces filtres ne doivent en aucun cas réduire la résolution de façon considérable. Pour cette fin nous définissons le taux de réjection du bruit par l'équation suivante :

$$\Phi = \left(\frac{\sin\left(\pi\left(\frac{1}{N} - \tau f_0\right)\right)}{\sin(\pi\tau f_0)} \right)^k \tag{2.24}$$

Cette équation est valide pour un filtre dont l'équation est

$$D(z) = \left(\frac{1}{N} \frac{(1-z^{-N})}{(1-z^{-1})} \right)^k \tag{2.25}$$

La décimation dépend de la nature du filtre utilisé qui peut être défini par l'équation (2.25). Cette dernière est caractérisée par les deux paramètres N et k, avec N qui représente le nombre d'échantillons du signal et k correspond à l'ordre du filtre. Ainsi la décimation est aussi caractérisée par ces 2 derniers. La figure 2.5 montre l'impact de ces paramètres sur la longueur du code [31].

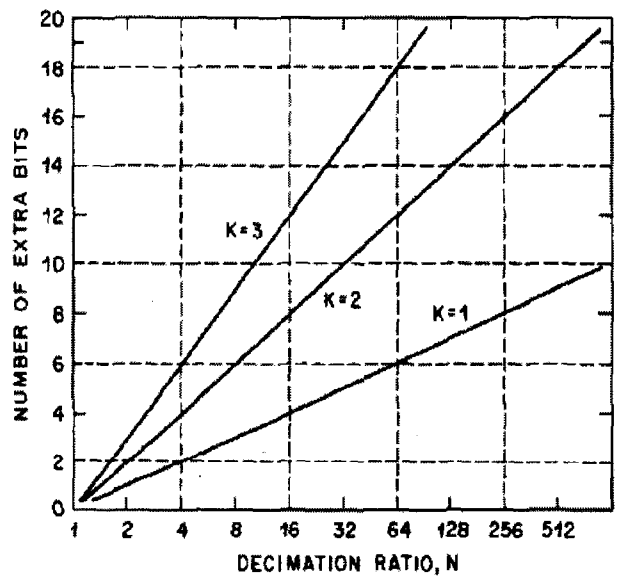


Figure 2.5. Impact de l'ordre du filtre de décimation sur la résolution du CAN

La technique de décimation par filtrage réduit la fréquence d'échantillonnage, mais son utilisation pour des basses fréquences résulte en un taux de réjection de bruit faible et introduit une distorsion, c'est pour cette raison que des recherches sont en cours pour améliorer des performances et peut être remplacer la décimation. Une des solutions envisageable est l'utilisation des algorithmes de décodage. En effet, des travaux récents donnent lieux à des résultats encourageant, surtout du point de vue du taux de réjection du bruit. Nous nous intéressons dans ce mémoire à une nouvelle méthode de décodage. Dans la section qui suit nous allons présenter les divers algorithmes de décodage répertoriés et nous introduisons une nouvelle technique de décodage appelée dynamique.

2.5. Algorithmes de décodage

Le décodage est le processus qui transforme un signal d'entrée $u(x)$ ayant subit la quantification en une estimation $\tilde{u}(x)$. Le filtrage linéaire est un décodage qui réduit le bruit introduit par la quantification avec une très haute résolution. Les décodeurs les plus répandus sont les filtres à réponse impulsionnelle finie (FIR) qui présentent des caractéristiques fortes intéressantes notamment en ce qui concerne le bruit et la vitesse.

Étant donné que le signal de sortie du quantificateur renferme beaucoup d'informations et que l'utilisation d'un filtre linéaire pourrait provoquer la perte d'une partie de ces informations, notre attention s'est portée sur de nouveaux algorithmes de décodage pour extraire le maximum d'information de ce dernier. Les recherches dans ce domaine ne sont pas abondantes car l'utilisation des algorithmes de décodage présente un défi pour aboutir à une implémentation matérielle peu complexe. Avec l'arrivée des FPGA, la tâche est devenue plus facile, puisque cet outil permet de passer d'une description logicielle à une implémentation matérielle facilement.

Parmi les approches qui ont été élaborées pour le décodage d'une séquence générée par un modulateur ΣΔ, nous retrouvons le « Zoomer algorithm » [32], le « Robust O(N log N) algorithm » [38] et le « Rational cycle decoding algorithm » [34].

2.5.1. L'algorithme « Zoomer »

L'idée de base de cet algorithme est d'utiliser les limites inférieures et supérieures pour pouvoir déterminer une approximation de la valeur d'entrée, ces limites sont définies par les lettres *U (Upper) et L (Lower).* En effet cet algorithme compare chaque fois la valeur quantifiée à la valeur moyenne $\overline{X}$ de la somme des valeurs quantifiées précédentes. La limite supérieure correspond au max de (U, $\overline{X}$) et la limite inférieure correspond au min de (L, $\overline{X}$). Ainsi plusieurs itérations sont effectuées jusqu'à un maximum de N itérations, N étant la longueur de la séquence en entrée. La valeur décodée est la moyenne de U et L. Au cours de l'initialisation, chaque limite est affectée à une borne de la gamme dynamique du quantificateur. En d'autres mots cet algorithme fonctionne comme une boule qui ce gonfle jusqu'à atteindre un volume donné comme montré dans la figure 2.6.

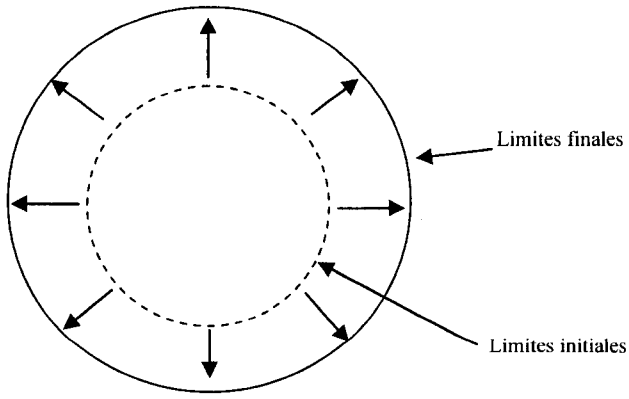


Figure 2.6. Principe de l'algorithme "Zoomer"

La figure 2.7 détaille le fonctionnement d'un tel algorithme.

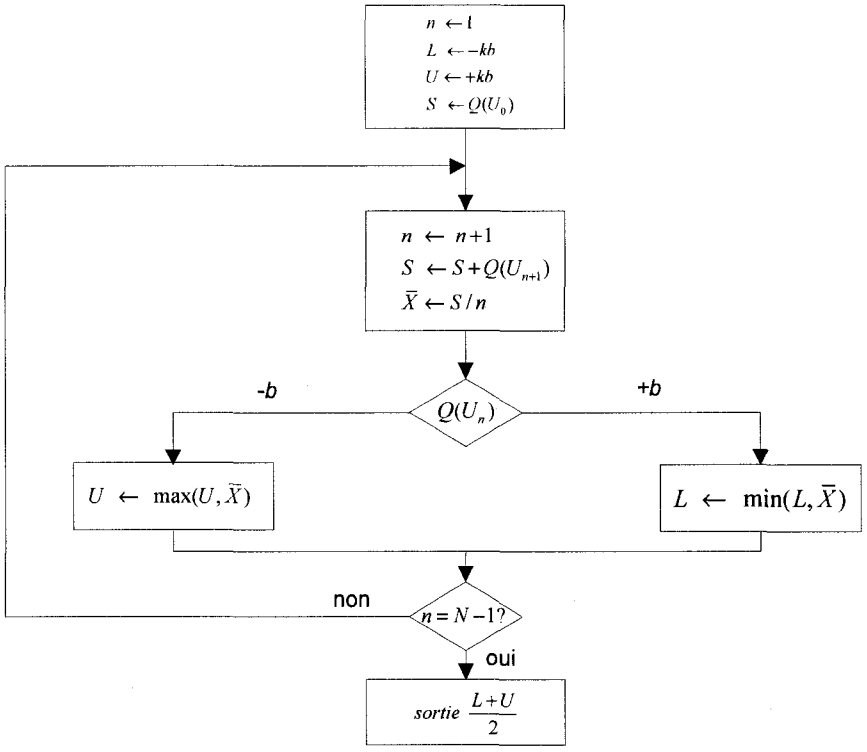


Figure 2.7. Organigramme de l'Algorithme "Zoomer"

où S est la sommes des valeurs quantifiées dans une séquences données, n un compteur qui permet de déterminer le nombre d'itérations effectuées, b est un paramètre du quantificateur défini par U dans l'équation (2.1). k est un coefficient déterminant la gamme dynamique du quantificateur, idéalement on lui attribue la valeur 0.9.

$q(u_n)$ est la valeur quantifiée du signal à un instant donné, $\bar{X}$ est la valeur moyenne du signal en tenant compte uniquement d'un certain nombre de bits significatifs dépendamment de l'itération et N est la longueur de la séquence.

Notons que pour cet algorithme :

- Tous les états initiaux sont à zéro.
- Le signal d'entrée est constant.

La figure 2.8 montre la performance du décodage en utilisant l'algorithme Zoomer par rapport à un FIR, le critère de performance est le SNR.

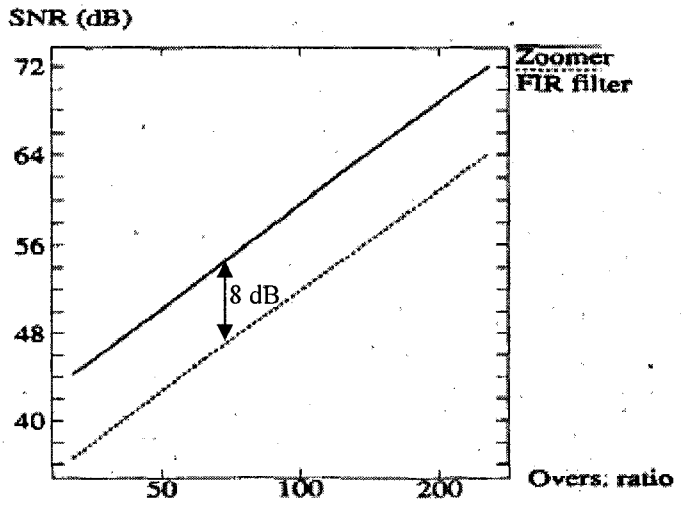


Figure 2.8. SNR vs. ratio de sur échantillonnage (tiré de [32])

Nous remarquons que le décodage utilisant l'algorithme Zoomer est meilleur que celui utilisant le FIR dans le cas d'un modulateur ΣΔ de premier ordre. Il est à noter que cet algorithme a été validé aussi pour les modulateurs ΣΔ de $2^{ème}$ ordre.

2.5.2. L'algorithme Robust O(N log N)

Cet algorithme se sert d'un signal d'entrée $x \in [0,1/2]$. Soit $y(m)$ une séquence générée par un modulateur ΣΔ de premier ordre pour ce signal constant x compris entre 0 et ½. Posons $w(i)$ la séquence dérivée de $y(m)$. $w(i)$=1 si la longueur des zéros successifs est égale à $\lfloor 1/x \rfloor$ et $w(i) = 0$ dans le cas où cette longueur est égale à $\lfloor 1/x \rfloor$ - 1. Notons que $\lfloor 1/x \rfloor$ représente le plus grand entier inférieur ou égal à $1/x$. Notons que m est la longueur de la séquence originale et i celle de la séquence dérivée.

L'exemple de la Figure 2.9 illustre ce principe :

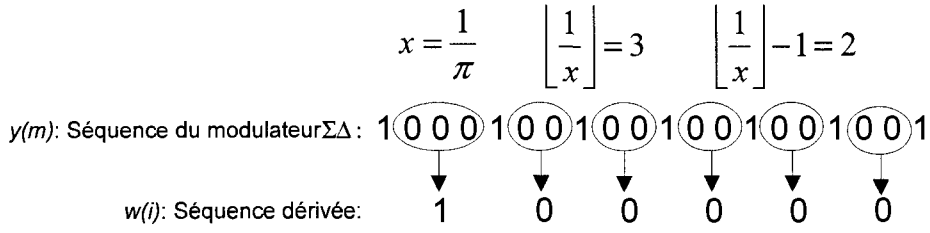


Figure 2.9. Exemple de génération de la séquence dérivée *w(i)*

En fait cet algorithme reconstitue le signal d'entrée à partir de la séquence dérivée *w(i)*. Si la longueur de la séquence dérivée est inférieure ou égale à la moitié de la longueur de la séquence originale *y(m),* alors l'algorithme calcule le signal d'entrée à partir de cette dernière, dans le cas contraire, il calcule la dérivée de celle-ci et ainsi de suite jusqu'à satisfaire la condition précédente.

Dépendamment de la séquence *y(m)* ou *w(i)*, les cas suivants se présentent :

- Il n'y a que des 0 dans la séquence

$$E[x|\ x<(1/N)\ et\ uniquement\ des\ 0s]=2N\int_0^{1/N}x(1-Nx)dx=\frac{1}{3N} \quad (2.26)$$

$E[x|x<\frac{1}{N}etuniquementdes0s]=2N$

où N est la longueur de la séquence. Et E une estimation de x.

Si j était le nombre de zéros précédant le 1 et k le nombre des zéros succédant à ce dernier, le calcul d'une estimation E de x se fait en se basant sur les formules suivantes :

- $j \neq k$

$$E[x|un\ 1\ et\ L\ 0s]=L(L+1)\left[\int_0^{1/(L+1)}2x^2dx+\int_{1/(L+1)}^{1/L}2x^2\left(\frac{1}{x}-L\right)dx\right]=\frac{1}{3(L+1)}+\frac{1}{3L} \quad (2.27)$$

où $L=max\ (j,\ k)$

- $j = k$

$$E[x|un\ 1, N\ pair\ (N-1)/2\ 0s] = 2N(N+1)\left[\int_0^{2/(N+1)} 2x^2 dx + \int_{2/(N+1)}^{2/N} x^2\left(\frac{1}{x} - \frac{N}{2}\right)dx\right] = \frac{2}{3(N+1)} + \frac{2}{3N} \quad (2.28)$$

où N est la longueur de la séquence

- Lorsque séquence totale ne peut être décomposée qu'en un seul échantillon d'une longueur unique, alors

$$E\left[x|uniquement\ des\ 0\ de\ longueur\ L\right] = \frac{1}{L+1-\frac{1}{9n^2}(L+1)} \approx \frac{1}{L+1} \quad (2.29)$$

où n est un paramètre interne représentant le nombre de d'élément courant dans une séquence. L'élément courant peut être 1 ou 0 selon le signal échantillonné [33]. Ainsi la complexité d'un tel algorithme est de *O (N log (N))* [33]. La figure 2.10 montre les performances de cet algorithme par rapport à des FIR de différentes formes. La forme d'un FIR correspond à la fenêtre de filtrage appliquée. Dans l'exemple présenté à la Figure 2.10 des fenêtres carrées et triangulaires, définies dans [33], ont été appliquées.

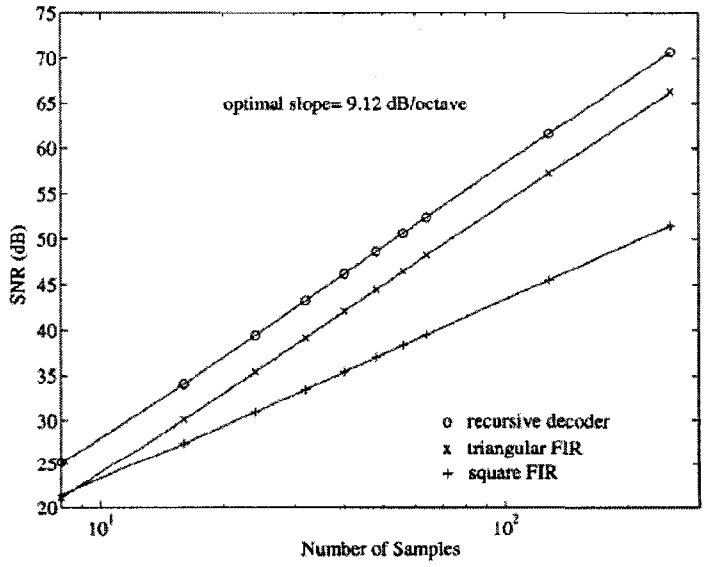


Figure 2.10. Performances de l'algorithme *O* (N *log* N) (tiré de [33])

Nous constatons qu'avec cet algorithme, nous avons un meilleur SNR par rapport aux autres techniques mentionnées dans la figure 2.10.

2.5.3. Modèle de génération hiérarchique et décodage optimal

Cet algorithme (rational Cycle decoding) est l'un des plus récents algorithmes de décodage. Il présente des performances meilleures qu'un FIR, que l'algorithme « Zoomer » et l'algorithme « Robust O(Nlog(N)) », au dépens d'une complexité plus élevée [34].

Ce décodage est basé sur la prédiction du code suivant, tout en estimant le prédécesseur d'un code donné, le résultat final serait la moyenne de ces trois valeurs.

Sachant que cet algorithme est à l'origine réalisé dans ce mémoire, nous allons décrire plus en détails son fonctionnement. Mais pour cette fin il est important de connaître le fonctionnement de l'encodeur.

2.5.3.1. Aperçu sur l'encodeur

Reprenons le schéma du CAN ΣΔ de la Figure 2.1 mais vu autrement comme montré dans la Figure 2.11. Ce schéma, dans lequel le délai a été représenté dans la boucle de rétroaction, a été repris uniquement pour simplifier les notations.

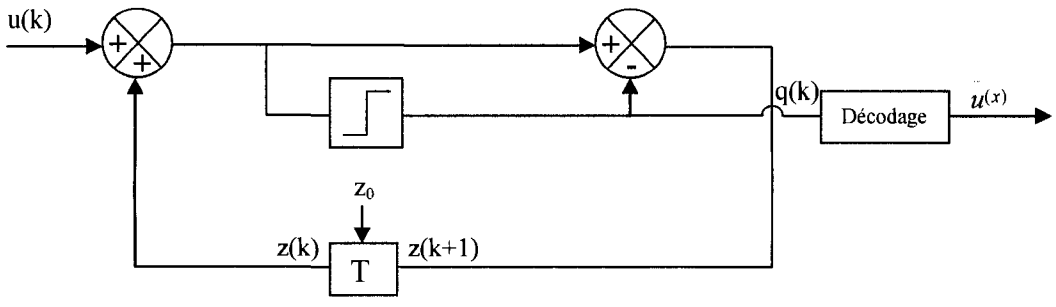


Figure 2.11. Structure du convertisseur ΣΔ de premier ordre (Tirée de [34])

La problématique de cet algorithme est la suivante:

Tenant compte du fait que l'entrée est représentée par une séquence finie comme montré dans l'équation (2.30), et que l'état initial z_0 est inconnu, une estimation $(\tilde{u}(q))$ de la séquence q peut être trouvée. Notons que q est une séquence de longueur N,

$$q = q(0), \ldots, q(N-1) \tag{2.30}$$

De plus, le ratio signal sur bruit quantifié (SQNR, Signal-to-Quantization Noise Ratio) doit être maximisé :

$$SQNR = \frac{E(u^2(k))}{E((u(k)-\tilde{u}(k))^2)} \Rightarrow Max \tag{2.31}$$

E(.) étant l'espérance.

Chaque séquence q d'un ensemble de bits correspond à une région bien définie du diagramme des vecteurs de quantification, tel qu'illustrer aux Figure 2.12 et figure 2.13. Vu le caractère aléatoire du signal d'entrée, chaque valeur quantifiée dépend non seulement du signal lui-même u_0 mais aussi de la valeur d'initialisation z_0. De ce fait à chaque paire $(u_0, z_0) \in [0,1[^2$ correspond une région unique du diagramme, chaque région B possède les caractéristiques suivantes :

- B peut avoir la forme triangulaire ou celle d'un quadripôle.
- Si B était un quadripôle, alors une de ses diagonales sera parallèle à l'axe des z_0 horizontale (2 coins du quadripôle ont l'ordonnée u_0). Les coordonnées de cette région sont définies par (u_{max}, u_c, u_{min}).

Les figure 2.12 et figure 2.13 résument ces conditions. De plus, la figure 2.13 montre le diagramme des vecteurs de quantification pour une séquence de longueur 6 bits.

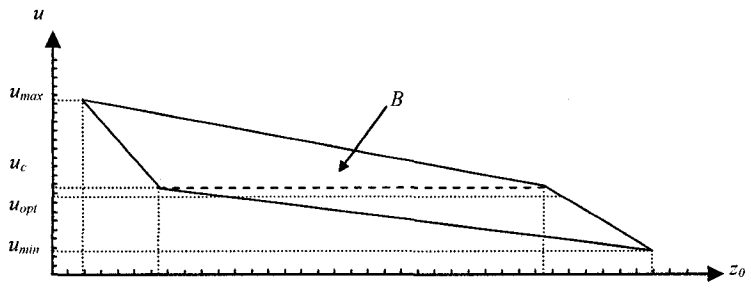


Figure 2.12. Région B sous forme d'un quadripôle

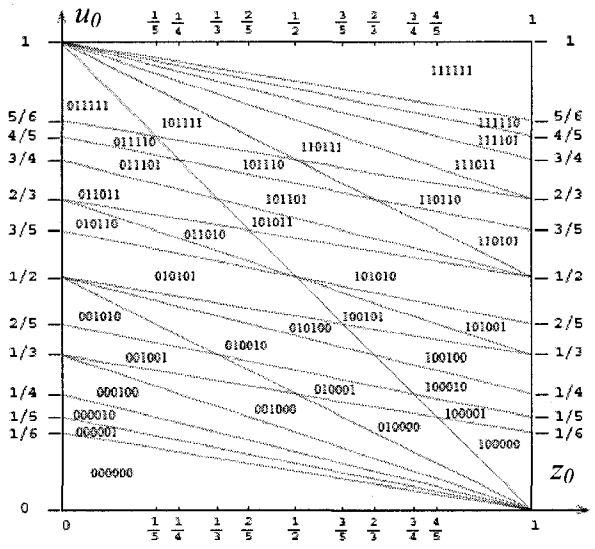


Figure 2.13. Diagramme des vecteurs de quantification normalisé

Chaque région de la Figure 2.13 est définie par un couple (u_0, z_0). À chaque couple (u_0, z_0) est associée une seule et unique séquence q de longueur N bits. Ce principe est utilisé pour retracer la valeur du signal encodé par le modulateur. Les paragraphes suivants détaillent le décodage en question.

2.5.3.2. Principe de fonctionnement du décodeur basé sur le modèle de génération hiérarchique

Le point de départ de cet algorithme connu aussi sous le nom de « Rational cycle decoding » est l'équation suivante :

$$MSE\ (Mean\ Square\ Error) = E(u(k) - \tilde{u}(k))^2 \tag{2.32}$$

où $\tilde{u}(k)$ est la valeur décodée de l'entrée

En intégrant cette équation sur la région B, on aura :

$$\forall q : MSE(q) = \iint_{B(q)} \left(u_0 - \tilde{u}(q)\right)^2 dz_0 du_0 \tag{2.33}$$

Les variables z_0 et u_0 sont supposées être indépendantes et uniformément distribuées sur toute la région [36] [37]. Ainsi la résolution de l'équation (2.33) nous conduit au résultat suivant :

$$\tilde{u}_{opt}(q) = \frac{1}{3}(u_{\max}(q) + u_{mc}(q) + u_{\min}(q)) \tag{2.34}$$

Ainsi la valeur optimale serait la valeur moyenne de la valeur courante, de son prédécesseur et de son successeur. En effet, chaque séquence q possède un prédécesseur et un successeur définis comme suit :

$$u_c(Succ(q)) = u_{\max}(q) \tag{2.35}$$

$$u_c(Pred(q)) = u_{min}(q) \tag{2.36}$$

où u_{mc} est la valeur de la séquence courante qui est égale à $u_c(q)$, $u_{max}(q)$ et $u_{min}(q)$ sont respectivement les valeurs du successeur et du prédécesseur. En conséquence l'équation (2.34) peut s'écrire autrement :

$$u_{opt}(q) = \frac{1}{3}(u_c(Succ(q)) + u_c(q) + u_c(Pred(q))) \tag{2.37}$$

Cet algorithme de décodage se résume donc aux opérations suivantes :

- trouver les successeurs et prédécesseurs d'une séquence donnée : q, *Succ(q)* et *Pred(q)*;
- déterminer les paramètres relatifs à: q, *Succ(q)* et *Pred(q)* : poids et longueur, ces derniers sont des fonctions de probabilité permettant de calculer q et son voisinage;
- calculer u_{opt} à partir de l'équation (2.34) en se servant des paramètres établis précédemment;

La méthode de détermination des successeurs et prédécesseurs a été établie par [34] à partir des Figure 2.12 et figure 2.13.

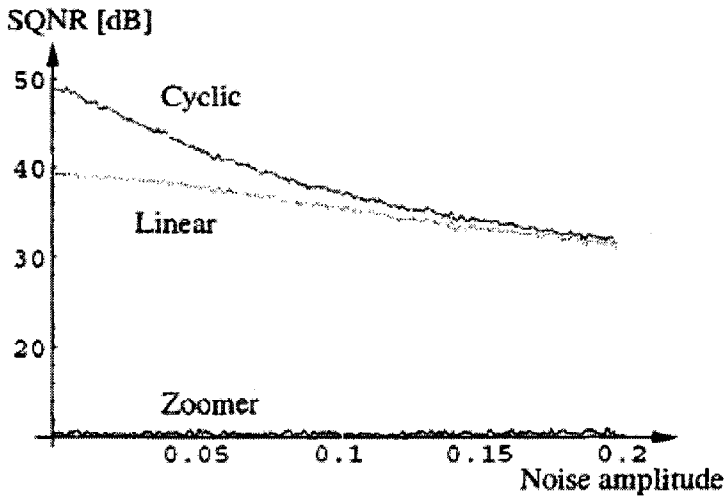


Figure 2.14. SQNR vs amplitude du bruit normalisée pour les algorithmes de décodage cycliques, linéaires et l'algorithme Zoomer.

Notons que le décodage linéaire consiste dans l'utilisation d'un filtre passe bas. Ce dernier algorithme présente de meilleures performances lorsque le bruit est de faible amplitude d'un point de vue SQNR que les autres, par contre son point faible est sa complexité qui est élevée.

De ce fait nous avons entrepris la mise en œuvre d'une architecture de décodage qui assure de meilleures performances que cette dernière et une vitesse de décodage plus élevée. Ceci est important pour des systèmes à haute précision ce qui est le cas de notre application où la détection des substances fluidiques dans la puce doit être très précise.

La compression des données et l'accélération du traitement de l'information, sont des techniques largement utilisées en imagerie et notamment dans le standard H.264 [38]. En effet, seulement l'information significative est utilisée pour restituer l'image. Certains algorithmes utilisant cette norme se basent sur un critère de comparaison qui leur permet de décider si l'information qu'ils sont en train de traiter est nécessaire ou non pour restaurer l'image source, et ceci, en la comparant avec des données précédemment sauvegardées dans une mémoire. Un tel système est principalement utilisé dans les vidéos conférences en temps réel. En essayant d'appliquer ce même principe pour les modulateurs ΣΔ, nous pouvons ainsi doter le CAN d'un certain degré de liberté. Ce dernier peut subséquemment arrêter la conversion au moment opportun, puisqu'il dispose de suffisamment d'information pour restituer le signal original.

2.6. Conclusion

Après avoir présenté les points culminants des convertisseurs ΣΔ, tout en examinant les principaux algorithmes de décodage, nous avons porté une attention particulière à l'algorithme « Rational cycle decoding ». Nous allons, dans les chapitres qui suivent, détailler l'architecture proposée tout en précisant les liens avec l'algorithme « Rational cycle decoding ».

Chapitre 3
Formulation mathématique du processus de décodage basé sur l'algorithme « Rational Cycle decoding »

3.1. Introduction

Après avoir présenté les principales techniques de décodage, leurs avantages et inconvénients, nous présentons dans ce chapitre l'algorithme de décodage que nous avons implémenté ainsi que sa formulation mathématique.

L'inconvénient principal du processus du décodage est le fait qu'il nécessite un grand temps d'exécution, ce qui réduit la fréquence d'échantillonnage du CAN ΣΔ. En plus, la plupart des CAN ne distinguent pas entre un signal DC et AC, ce qui représente un inconvénient important pour les applications des signaux DC ou de très faible fréquence. Ainsi au cours de ce chapitre, nous décrivons la théorie de décodage dynamique, développée tout en expliquant ses liens avec l'algorithme de génération hiérarchique et le décodage optimal.

Comme chaque circuit numérique, un décodeur numérique ne peut traiter que des séquences de bits : « 0 » et « 1 » uniquement. Les éléments de base de la technique de décodage utilisée sont l'élément correctif (Correction element) que nous noterons « $E_{correctif}$ » et l'élément courant (Run element) que nous noterons « $E_{courant}$ ». Ces deux éléments permettent de définir la nouvelle séquence. Une séquence est définie par une suite d'éléments que nous appelons symboles. Ainsi, une séquence de longueur n contiendra n symboles. Un symbole peut être l'élément correctif ou l'élément courant. Ces éléments constituent la base de la théorie qui sera élaborée dans ce chapitre.

3.2. Algorithme de décodage statique

Sachant que le décodeur reçoit une suite de bits (0 ou 1), la séquence originale, que nous représentons par le symbole $S_{originale}$ est une suite de 0 et de 1 générée par le modulateur ΣΔ tel que représentée par l'équation 3.1, N étant la longueur de la séquence.

$$S_{originale} = \left\{ S_{originale}(k),\ 0 \le k \le N-1,\ tel\ que\ S_{originale}(k) = 1\ ou\ 0 \right\} \quad (3.1)$$

3.2.1. Détermination de l'élément courant et de l'élément correctif

L'élément courant dans une séquence est une approximation du signal analogique, alors que l'élément correctif est une correction due à un dépassement de seuil dans le quantificateur du modulateur ΣΔ. Le nombre des $E_{courant}$ successifs dans une séquence est l'élément déterminant dans l'approximation du signal analogique. Il est à noter que dans une séquence donnée, seuls les $E_{courant}$ peuvent se succéder. On ne peut jamais avoir deux éléments correctifs successifs. D'un point de vue mathématique ces deux éléments sont calculés en utilisant le poids moyen de ces derniers, qui est la valeur moyenne des poids respectifs des deux éléments.

En effet chaque élément de la séquence possède son propre poids w_r si c'est un $E_{courant}$ ou, w_c si c'est un $E_{correctif}$ et sa longueur l_r si c'est un $E_{courant}$ ou, l_c si c'est un $E_{correctif}$. Ces derniers sont des paramètres de probabilité dont l'explication ne représente pas une grande importance dans l'algorithme développé. Ainsi si nous représentons $E_{courant}$ par le symbole S_r et l'élément correctif par le symbole S_c, nous pouvons attribuer à chacun d'eux les couples suivants.

$$S_r \rightarrow (w_r, l_r) \quad (3.2)$$

$$S_c \rightarrow (w_c, l_c) \quad (3.3)$$

Par la suite nous pouvons dériver la valeur moyenne de la séquence :

$$W = \frac{w_r + w_c}{l_r + l_c} \tag{3.4}$$

D'un point de vu mathématique, comment pouvons nous distinguer les 2 éléments l'un de l'autre ?

En fait cette distinction se base principalement sur le paramètre *W*. Nous représentons la valeur échantillonnée du signal analogique à un instant *t* par u_0, et étant donné qu'au début nous ne pouvons savoir lequel des deux éléments est l'$E_{courant}$ ou l'$E_{correctif}$, nous utilisons cette représentation : S_0 pour représenter le bit de valeur « 0 » et S_1 pour le bit de valeur « 1 ». Ainsi :

- Si u_0>W alors S_1 est l'$E_{courant}$ et S_0 est l'$E_{correctif}$.
- Si u_0<W alors S_0 est l'$E_{courant}$ et S_1 est l'$E_{correctif}$.
- Si u_0=W alors S_0 peut être considéré comme l'$E_{courant}$ et S_1 comme l'$E_{correctif}$ et vise versa.

Maintenant que nous connaissons les deux éléments de base de la séquence courante, nous pouvons générer une nouvelle séquence qui fera l'objet de la prochaine section.

3.2.2. Génération des séquences intermédiaires

L'algorithme de décodage utilisé se base sur le modèle de génération hiérarchique des séquences [34]. Ainsi pour atteindre la valeur optimale, plusieurs itérations sont nécessaires. Chaque itération correspond à un niveau de décodage dans lequel plusieurs opérations sont effectuées pour approximer la valeur échantillonnée du signal analogique. Pour cette fin nous allons désormais attribuer un autre indice

aux différents paramètres expliqués précédemment correspondant au niveau courant. Par exemple, si nous sommes au premier niveau de décodage correspondant au niveau « 0 », c'est-à-dire que la séquence analysée est celle fournie par le modulateur ΣΔ, nous aurons selon [34]:

$$W_0 = \frac{w_{r,0} + w_{c,0}}{l_{r,0} + l_{c,0}} \tag{3.5}$$

l'équation (3.5) correspond à la valeur moyenne des poids relatifs calculé à l'itération de décodage courante. et ainsi de suite jusqu'à satisfaire des conditions d'arrêts qui sont expliquées dans 3.2.3. Pour plus de clarté, au paragraphe 3.2.5 un exemple détaillé d'une séquence générée par le modulateur ΣΔ de premier ordre.

La génération des nouvelles séquences se base sur le nombre d'$E_{courant}$ successifs dans la séquence du niveau précédent. Notons qu'un niveau correspond à une itération effectuée par l'algorithme de décodage. La figure 3.1 extraite de [34] représente la méthode de génération des séquences pour différents niveaux de décodage. Seq_i correspond à la séquence générée à la fin du décodage du niveau i. $w_{r,i}$ et $l_{r,i}$ représentent le poids et la longueur du $E_{courant}$; $w_{c,i}$ et $l_{c,i}$ ceux de l'élément correctif dans la séquence du $i^{ème}$ niveau. Supposons que la séquence du niveau i est formée par les symboles suivants :

$$S_{c,i}S_{r,i}S_{r,i}S_{c,i}S_{r,i}S_{r,i}S_{r,i}S_{c,i}S_{r,i}S_{r,i}S_{c,i}S_{r,i}S_{r,i}S_{r,i}S_{c,i}S_{r,i}S_{r,i}S_{r,i}S_{c,i}S_{r,i}S_{r,i}S_{c,i} \tag{3.6}$$

La séquence qui est généré au niveau i+1 est :

$$S_{c,i+1}S_{r,i+1}S_{c,i+1}S_{r,i+1}S_{r,i+1}S_{c,i+1} \tag{3.7}$$

avec $S_{c,i+1}=2$ et $S_{r,i+1}=3$ puisque il y a trois $S_{r,i}$ successifs et deux $S_{r,i}$ successifs séparés par un $S_{c,i}$.

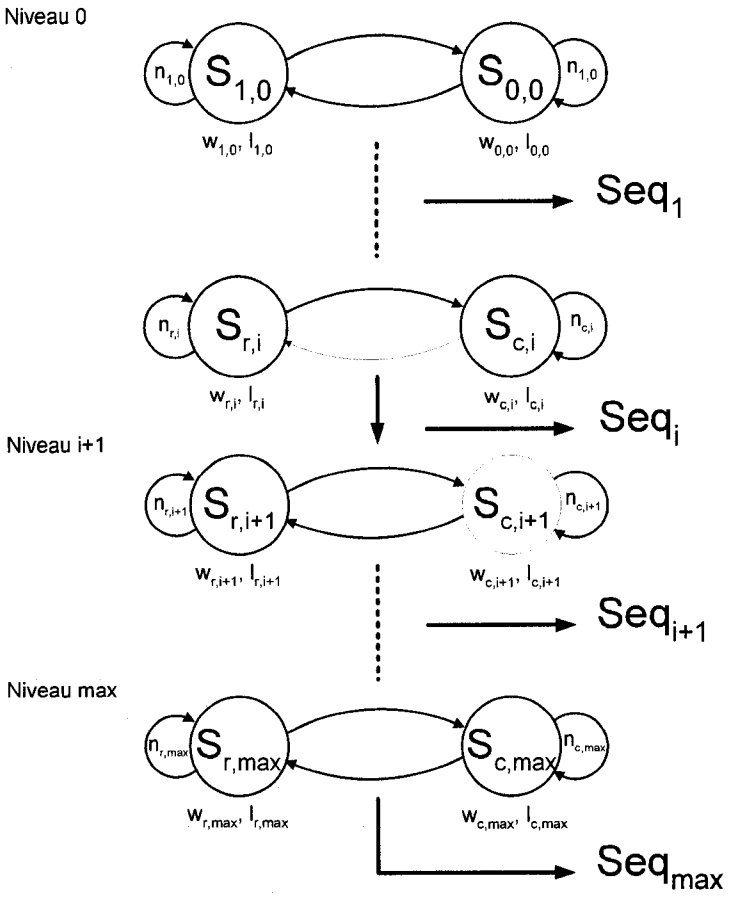


Figure 3.1. Modèle de génération des séquences intermédiaires

Il est à noter que dans une séquence donnée l'$E_{courant}$ correspond au symbole dont la longueur est supérieure ou égale à 1. La longueur de l'élément correctif ne peut être que « un ». Ceci est du encore une fois au principe de fonctionnement du convertisseur ΣΔ. L'index *max* correspond au dernier niveau de décodage qui répond aux critères détaillés dans le paragraphe 3.2.3. $n_{r,i}$ et $n_{c,i}$ correspondent respectivement au nombre d'$E_{courant}$ et d'$E_{correctif}$ dans une séquence donnée. À chaque niveau de décodage i, W_i est calculée selon la formule suivante :

$$W_i = \frac{w_{r,i} + w_{c,i}}{l_{r,i} + l_{c,i}} \tag{3.8}$$

la valeur échantillonnée serait W_{max}

$$W_{max} = \frac{w_{r,max} + w_{c,max}}{l_{r,max} + l_{c,max}} \tag{3.9}$$

Les formules suivantes permettent de retrouver les différents paramètres:

$$si\ n_{r,i} \geq 1 \begin{cases} S_{r,i+1} = n_{r,i} + 1 \\ S_{c,i+1} = n_{c,i} \\ l_{r,i+1} = (n_{r,i} + 1) l_{r,i} + l_{c,i} \\ l_{c,i+1} = n_{r,i} l_{r,i} + l_{c,i} \\ w_{r,i+1} = (n_{r,i} + 1) l_{r,i} + l_{c,i} \\ w_{c,i+1} = n_{r,i} w_{r,i} + w_{c,i} \end{cases} \tag{3.10}$$

L'équation 3.10 représente les paramètres caractéristiques dans le cas où de l'$E_{courant}$. De la même façon nous déterminons les paramètres correspondants à l'$E_{correctif}$ en se référant à [34]. Comme cela a été déjà mentionné dans le 1er chapitre, la valeur obtenue à la fin du décodage est la valeur moyenne de la séquence obtenue du modulateur ΣΔ. En fin du compte, cette technique de décodage est comme si nous avions un modulateur ΣΔ au niveau de chaque étage *i* qui génère la séquence Seq_i. À chaque étage, nous essayons de retrouver la valeur moyenne W_i de la séquence générée et à chaque étage la valeur estimée de l'entrée analogique échantillonnée est soit supérieure ou inférieure à cette dernière jusqu'au niveau max ou elle est égale à W_{max}. À titre d'exemple, la figure 3.2 montre l'évolution du décodage niveau par niveau, sachant que u_{est} est la valeur estimée de l'entrée analogique.

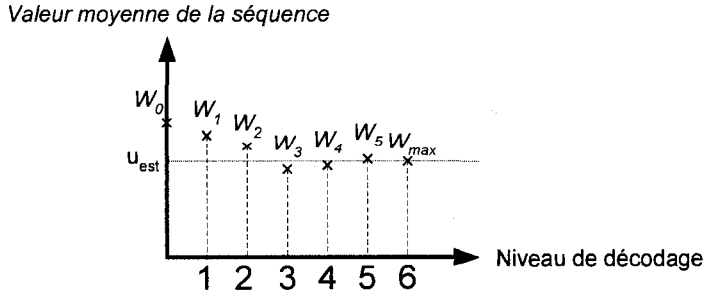


Figure 3.2. Principe du décodage

3.2.3. Critères d'arrêt

Le processus de décodage s'arrête dès que l'une de ces conditions est satisfaite :

- la longueur de la séquence générée à la dernière itération est inférieure ou égale à 2.

$$Seq_i = \left(S_{r,i}S_{c,i}\right) \text{ ou } Seq_i = \left(S_{r,i}\right) \tag{3.11}$$

- La période de la séquence Seq_i est 1 et peu importe le symbole qu'elle contient tel que montré dans l'équation (3.12) qui est un cas particulier de l'équation (3.6). ce qui revient à vérifier la périodicité de la séquence.

$$Seq_i = \left(S_{r,i}S_{r,i}S_{r,i}S_{r,i}S_{r,i}S_{r,i}S_{r,i}S_{r,i}\right) \tag{3.12}$$

3.2.4. Technique de décodage

Dans cette section nous présentons la technique de décodage et comment aboutir à u_{est} à partir de la séquence générée par le modulateur ΣΔ.

Le décodage se base principalement sur la quantification vectorielle (vector quantification approach) présentée dans 2.5.3 et dans [31] pour déterminer les séquences précédentes et suivantes d'une séquence donnée.

Avant qu'une séquence ne soit envoyée au niveau de décodage suivant, les éléments superflus sont enlevés. Ces éléments s'appellent les $E_{courant}$ non encadrés (Non Closed Run : NCR). Ces derniers ne renferment pas une information utile parce qu'ils correspondent à des $E_{courant}$ situés aux extrémités de la séquence et ne sont pas encadrés par des éléments correctifs de part et d'autre. Dans le cas où il y a deux NCR dans une séquence, seule le plus petit des NCR est supprimé. La nouvelle séquence est utilisée pour déterminer la séquence précédente et suivante selon la méthode de la quantification vectorielle.

Supposons que nous disposons d'une séquence courant Seq_i dans le niveau de décodage i tel que :

$$Seq_i = \{Seq_i(k),\ tels\ que\ Seq_i(k)\ sont\ les\ E_{courant}\ et\ E_{correctif}\ de\ la\ séquence\} \quad (3.13)$$

avec $0 \leq k \leq N_{max}$, N_{max} correspond à la longueur maximale de la séquence déterminée par le nombre d'$E_{courant}$ et d'$E_{correctif}$ dans cette dernière.

Connaissant Seq_i nous déterminons son prédécesseur $Pred_i$ et son successeur $Succ_i$ comme suit :

$$Pred_i = \begin{cases} Pred_i(k), tel\ que\ 0 \leq k \leq a\ et\ a \leq k \leq N_{max} \\ Pred_i(a) = Seq_i(a) + 1 \\ Seq_i(a) : le\ dernier\ symbole\ le\ moins\ élevé\ possible \end{cases} \quad (3.14)$$

$$Succ_i = \begin{cases} Succ_i(k), tel\ que\ 0 \leq k \leq b\ et\ b \leq k \leq N_{max} \\ Succ_i(b) = Seq_i(b) - 1 \\ Seq_i(b) : le\ dernier\ symbole\ le\ plus\ élevé\ possible \end{cases} \quad (3.15)$$

Sachant que :

$$w_{c,i+1} = E_{correctif}.w_{r,i} + w_{c,i}, l_{c,i+1} = E_{correctif}.l_{r,i} + l_{c,i} \quad (3.16)$$

$$w_{r,i+1} = E_{courant}.w_{r,i} + w_{c,i}, l_{r,i+1} = E_{courant}.l_{r,i} + l_{c,i} \quad (3.17)$$

Au premier niveau de décodage nous avons :

$$w_{r,0} = l_{c,0} = 1\ et\ w_{c,0} = l_{r,0} = 0 \quad (3.18)$$

Si les conditions d'arrêts citées dans 3.2.3 sont satisfaites alors une estimation de la valeur échantillonnée est effectuée en utilisant les équations suivantes :

$$u_{Pred_{max}} = \frac{\left(\sum_k Pred_{\max}(k)\right) w_{r,\max} + w_{c,\max}}{\left(\sum_k Pred_{\max}(k)\right) l_{r,\max} + l_{c,\max}} \tag{3.19}$$

$$u_{Seq_{max}} = \frac{\left(\sum_k Seq_{max}(k)\right) w_{r,max} + w_{c,max}}{\left(\sum_k Seq_{max}(k)\right) l_{r,max} + l_{c,max}} \tag{3.20}$$

$$u_{Succ_{max}} = \frac{\left(\sum_k Succ_{max}(k)\right) w_{r,max} + w_{c,max}}{\left(\sum_k Succ_{max}(k)\right) l_{r,max} + l_{c,max}} \tag{3.21}$$

Si la séquence courante ne renferme que des NCR alors

$$u_{Seq_{max}} = \frac{w_{r,max}}{l_{r,max}} \tag{3.22}$$

L'indice *max* correspond au dernier niveau de décodage.

Enfin en appliquant l'équation (3.23), nous déterminons une estimation de la valeur échantillonnée tel que :

$$u_{est} = \frac{u_{Pred_{max}} + u_{Seq_{max}} + u_{Succ_{max}}}{3} \tag{3.23}$$

Ainsi, nous avons présenté l'approche de décodage statique utilisée par l'algorithme et détaillée encore plus en [34]. Cette technique étant extrêmement fiable pour un signal DC, nous nous sommes posés la question suivante : Pourquoi avoir à procéder à un décodage en entier si la valeur échantillonnée a été déjà décodée précédemment?

Ceci nous a poussé à développer une méthode d'optimisation du temps de décodage et ceci en minimisant ce dernier. Nous appelons cette méthode de décodage « dynamique ». cette méthode permet de réduire le nombre d'itérations ou de niveaux de

décodage nécessaires pour aboutir au résultat convenable. Ceci dit, une telle approche nécessite une modification de l'algorithme précédent. En effet avec ce décodage dynamique proposé il est primordial d'avoir une valeur de référence à chaque itération. Pour cette fin, nous avons utilisé l'équation (3.23) à chaque niveau de décodage étant donné que les séquences *Pred* et *Succ* sont déjà définies à chaque niveau. D'où l'équation suivante :

$$u_{est,i} = \frac{u_{pred,i} + u_{Seq,i} + u_{Succ,i}}{3} \tag{3.24}$$

où *i* représente le niveau de décodage courant.

En conséquence, à la fin du décodage statique d'une valeur échantillonnée, nous aurons une suite de valeurs estimées à chaque niveau et que nous noterons ensuite S_{est}, chaque suite est unique pour une valeur échantillonnée donnée.

$$S_{est} = \left(U_{est,0}, \ldots, U_{est,j}, \ldots, U_{est,max}\right) \tag{3.25}$$

Il est à noter que la valeur *max* correspond au dernier niveau de décodage et que cette valeur indique le nombre d'itérations qu'il faut faire pour aboutir à la bonne estimation.

La Figure 3.3 résume le résultat obtenu suite à l'opération de décodage statique

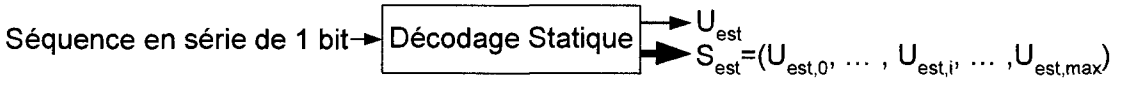


Figure 3.3. Vue globale du décodage statique (I/O)

Ensuite, le résultat obtenu du module statique est envoyé au module dynamique qui procède à la seconde phase du décodage comme expliqué dans la section 3.3.

3.2.5. Exemple de décodage

Dans cette section, un exemple détaillé de décodage a été repris pour expliquer les différentes étapes. Cet exemple est extrait de [34].

Supposons que nous disposons de la séquence Seq_0 générée par le modulateur ΣΔ pour un signal échantillonné de valeur $u_0 = \frac{14}{25}$ et état initial $z_0 = 0.5$ tel que :

$$Seq_0 = \{1,0,1,0,1,0,1,0,1,1,0,1,0,1,0,1,1,0,1,0,1,0,1,1,0,1,0,1,0,1,1,0,1,0,1,0,1,1,0,1,0,1,0\}$$

durant la première itération la séquence Seq_1 est générée par le décodeur tel qu'expliqué dans 3.2.2 et ceci en comptant le nombre de $E_{courant}$ dans la séquence du 1er niveau de décodage. Ainsi nous aurons :

$Seq_0 = \{1,0,1,0,1,0,1,0,1,1,0,1,0,1,0,1,1,0,1,0,1,0,1,1,0,1,0,1,0,1,1,0,1,0,1,0,1,1,0,1,0,1,0\}$

$Seq_1 = \{(1),1,1,1,2,1,1,2,1,1,1,2,1,1,1,2,1,1\}$

L'élément courant de Seq_0 est « 1 » puisque c'est le seul élément qui a une longueur supérieure à 1 à certains niveaux de la séquence Seq_0 tel que montré par les éléments encerclés dans l'exemple précédent. La longueur des $E_{courant}$ correspond au nombre de ces éléments qui sont encadrés par des $E_{correctif}$. Ainsi l'élément correctif ne peut être que « 0 ». À ce stade nous calculons les cœfficients relatifs à ce niveau en utilisant les équations (3.16) et (3.17), ainsi :

$$E_{courant} = 1 \qquad E_{correctif} = 0$$

$$w_{r,1} = 1 \qquad w_{c,1} = 0$$

$$l_{r,1} = 1 \qquad l_{c,1} = 0$$

$w_{r,1}$, $w_{c,1}$, $l_{r,1}$, $l_{c,1}$ correspondent aux poids (w) et longueur (l) des éléments courants (r) et éléments correctifs (c). L'objectif de notre décodage étant d'accélérer le temps de

traitement nous nous contentons de la définition mathématique des paramètres précédents. Ceci dit la définition exacte de ces derniers figure dans [34].

l'élément entre parenthèses (1) doit être supprimé de la séquence Seq_1 parce qu'il correspond à un élément superflue NCR (Non Closed Run). La longueur de la séquence Seq_1 est égale à 18 et sa période n'est pas égale à 1 et donc les conditions d'arrêt figurant dans 3.2.3 ne sont pas satisfaites. Ce qui implique une 2ème itération. Le même principe appliqué pour générer la séquence Seq_1 est utilisé pour générer la séquence Seq_2. Ainsi nous remarquons que dans la Seq_1 seul l'élément 1 a une longueur supérieur à 1 (éléments encerclés). Ce qui implique que 1 est l'$E_{courant}$. En conséquence :

$$Seq_1 = \{(1,1,1),2,(1,1),2,(1,1,1),2,(1,1,1),2,(1,1)\}$$

$$Seq_2 = \{(3),2,3,3,(2)\}$$

et

$$E_{courant} = 1 \qquad E_{correctif} = 2$$

$$w_{r,2} = 1 \qquad w_{c,2} = 2$$

$$l_{r,2} = 2 \qquad l_{c,2} = 3$$

L'élément (2) est supprimé de la séquence Seq_2, parce qu'il correspond au plus petit NCR de la séquence. La longueur de la séquence Seq_2 est égale à 4 et sa période n'est pas égale à 1 et donc les conditions d'arrêt figurant dans 3.2.3 ne sont pas encore satisfaites. Une troisième itération est lancée. Subséquemment :

$$Seq_2 = \{3,2,(3,3)\}$$

$$Seq_3 = \{(1),(2)\}$$

et

$$E_{courant} = 3 \qquad E_{correctif} = 2$$

$$w_{r,3} = 5 \qquad w_{c,3} = 4$$

$$l_{r,3} = 9 \qquad l_{c,3} = 7$$

L'élément (1) est supprimé de la séquence Seq_3 parce qu'il correspond au plus petit NCR de la séquence. La longueur de la séquence Seq_3 est égale à 1 ce qui correspond à la condition d'arrêt figurant dans 3.2.3. À ce stade nous générons le prédécesseur et le successeur en utilisant respectivement les définitions (3.14) et (3.15).

$$Pred_3 = \{1\}$$

Seq_3 ne contient que des NCR donc en se basant sur l'équation (3.22) nous n'avons pas à déterminer explicitement le successeur de la séquence Seq_3. par la suite nous calculons les estimations des différentes séquences : le prédécesseur, la séquence courante et le successeur en utilisant respectivement les équations (3.19), (3.20) et (3.22).

$$u_{Pred_3} = \frac{w_{r,3} + w_{c,3}}{l_{r,3} + l_{c,3}} = \frac{9}{16}$$

$$u_{Succ_3} = \frac{w_{r,3}}{l_{r,3}} = \frac{5}{9}$$

$$u_{Seq_3} = \frac{2.w_{r,3} + w_{c,3}}{2.l_{r,3} + l_{c,3}} = \frac{14}{25}$$

et enfin une estimation de séquence générée par le modulateur ΣΔ est calculée en utilisant l'équation (3.23) :

$$u_{est} = \frac{u_{Pred_3} + u_{Seq_3} + u_{Succ_3}}{3} = \frac{6041}{10800} \approx 0.55935$$

3.3. Algorithme de décodage dynamique

L'idée de base d'un algorithme de décodage dynamique est d'utiliser les résultats précédents en vu d'améliorer la vitesse d'échantillonnage. Une telle approche est déjà utilisée dans la gestion du flux des données sur Internet [31]. Dans cette section une description mathématique de cette approche est présentée.

3.3.1. Définition du décodage dynamique

Comme cela a été déjà mentionné dans le 2ème chapitre les modulateurs des convertisseurs ΣΔ fonctionnent en mode de sur échantillonnage et donc pour ramener la fréquence au domaine de Nyquist, il faut utiliser les filtres et les décodeurs. Cependant l'objectif principal était l'amélioration du SQNR. Cet objectif a été atteint par les décodeurs qui restent cependant lents. Pour accélérer le temps de conversion, nous avons eu recours à une approche dynamique basée sur une gestion du flux des données à travers des mémoires. Une telle approche requiert de la mémoire, ce qui est le cas de l'algorithme proposé qui, par l'intermédiaire d'une mémoire, gère le flux de données. Une vue globale de l'approche dynamique proposée est présentée dans la Figure 3.4.

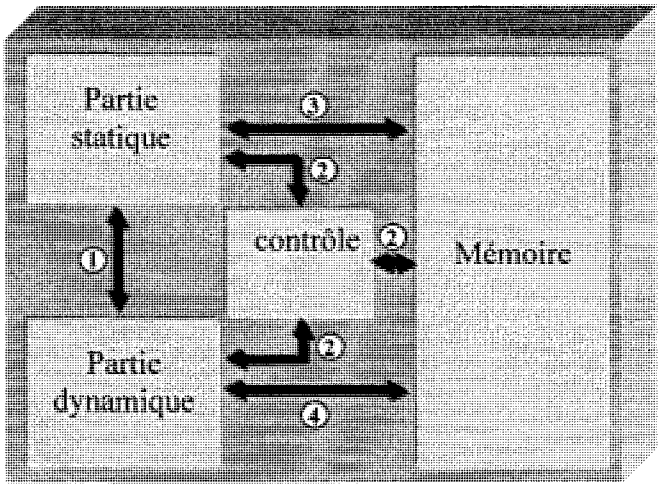


Figure 3.4. Principe de décodage dynamique

① Une communication entre le module dynamique et statique permet de stopper le décodage à n'importe quel niveau ou itération.

② La partie contrôle est l'équivalent du garde fou qui contrôle toutes les parties incluant la mémoire et donne les autorisations de fonctionner à chaque partie.

③ La partie statique est en communication directe avec la mémoire vu que le décodage statique se base sur un décodage sur plusieurs niveaux ou itérations et chacune des itérations fait appel aux résultats précédents.

④ La partie dynamique se base principalement sur la mémoire en comparant constamment le résultat de chaque niveau de décodage à tous les résultats précédents.

Les opérations 1, 2, 3 et 4 ne sont nullement indépendantes, d'où la particularité de la technique de décodage développée. Si nous comparons cette approche aux autres convertisseurs, nous remarquerons que la majorité des CAN ne donnent pas une grande importance au décodage, mais qu'elles portent plutôt une grande attention au temps d'échantillonnage en essayant d'optimiser les modulateurs.

3.3.2. Définition mathématique

Le décodage dynamique tel que montré aux Figure 3.4 et 3.**Erreur ! Source du renvoi introuvable.**7 dépend considérablement de l'algorithme de décodage statique. De ce fait nous utiliserons l'algorithme de décodage cyclique, étant donné que ce dernier présente de meilleures performances, tel qu'expliqué dans le 2ème chapitre. Sachant que S_{est_m} figurant dans l'équation (3.26) est le résultat du module statique pour une seule séquence de N bits générée par un modulateur ΣΔ et supposons que nous disposons de m séquences générées par ce modulateur, on attribue à la séquence S_{est} un autre indice est_m et max_m.

$$S_{est_m} = \left(u_{est_m,0},\ \ldots,\ u_{est_m,j},\ \ldots,\ u_{est_m,max}\right) \tag{3.26}$$

sachant que

- est_m correspond à l'ordre de génération de la séquence de bits du modulateur. En effet le modulateur fonctionne en continu, de sorte que les séquences se succèdent les unes après les autres en flux continu.
- max_m correspond au nombre d'itérations maximal pour décoder la séquence du modulateur qui est un élément caractéristique de chaque séquence.

Supposons maintenant que la mémoire dont nous disposons soit représentée par une matrice de l lignes et p colonnes ($mem\ [l \times p]$) telle que représentée dans l'équation (3.27).

$$mem = \begin{pmatrix} m_{00} & \cdots & m_{0(p-1)} \\ \vdots & \ddots & \vdots \\ m_{(l-1)1} & \cdots & m_{(l-1)(p-1)} \end{pmatrix} \tag{3.27}$$

De l'équation 3.26 nous remarquons qu'une séquence S_{est_m} est définie par l'indice max_m parmi d'autres paramètres. Une condition s'applique sur cet indice qui est :

$$max_m \leq l \tag{3.28}$$

Chaque séquence donnée occupe une seule et unique colonne de la matrice de *mem*. Comme la longueur d'une colonne est l, qui n'est pas forcément égale à max_m alors il faut à chaque fois concaténer S_{est_m} avec une matrice [l-max_m-1 × 1] de zéro que nous nommerons matrice « Zéro_m» de sorte que la nouvelle matrice S'_{est_m} a la même longueur que le nombre de lignes de la matrice *mem*.

$$S_{est_m} = \begin{pmatrix} u_{est_m\ 0} \\ \vdots \\ u_{est_m\ max_m} \end{pmatrix} \text{ et } Zéro_m = \begin{pmatrix} a_0 \\ \vdots \\ a_{l\text{-}max_m-1} \end{pmatrix} \text{ avec } a_{0 \leq i \leq l-\max_m-1} = 0 \tag{3.29}$$

d'où

$$S'_{est_m} = \begin{pmatrix} S_{est_m} \\ Zéro_m \end{pmatrix} \tag{3.30}$$

Supposons maintenant que le modulateur ΣΔ génère k séquences différentes en continu et l'une après l'autre tel que :

$$k < p \tag{3.31}$$

p étant le nombre de colonnes de la matrice *mem* ce qui correspond au nombre de séquences qui peuvent être enregistrées dans la mémoire du décodeur dynamique. À chaque séquence est associée une autre séquence S'_{est_i} où *i* représente le numéro de la séquence générée par le modulateur. Ainsi lorsque la dernière séquence *k*-1 est atteinte, la matrice *mem* sera comme suit :

$$mem = \left(S'_{est_0} \cdots S'_{est_{k-1}} \; Z\acute{e}ro_{l,p-k} \right) \tag{3.32}$$

où

$$Z\acute{e}ro_{l,p-k} = \begin{pmatrix} a_{00} & \cdots & a_{0(p-k-1)} \\ \vdots & \ddots & \vdots \\ a_{(l-1)0} & \cdots & a_{(l-1)(p-k-1)} \end{pmatrix}, \; avec \; a_{(0 \le i \le l-1)(0 \le j \le p-k-1)} = 0 \tag{3.33}$$

Autrement, la matrice *mem* renferme les valeurs décodées à chaque niveau d'itération et de chaque séquence générée par le modulateur ΣΔ. Le reste de la matrice est complété par des « 0 ». En plus de la matrice *mem*, une autre matrice ligne appelée *index* [*0* × *p*] est utilisée contenant *p* éléments. Chaque élément est l'indice max_i de la séquence *i*. Si nous disposons comme précédemment de *k* séquences, *index* sera :

$$index = \left(max_0 \cdots max_{k-1} \; ind_k \cdots ind_{p-1} \right) avec \; ind_{k \le i \le p-1} = 0 \tag{3.34}$$

Le contrôle de la mémoire se fait par le biais d'un masque. Ce dernier est représenté par une matrice que nous appelons *mas.* Cette matrice a les mêmes dimensions que la matrice *mem.*

$$mas = \begin{pmatrix} M_{00} & \cdots & M_{0(p-1)} \\ \vdots & \ddots & \vdots \\ M_{(l-1)0} & \cdots & M_{(l-1)(p-1)} \end{pmatrix} \tag{3.35}$$

Ce masque est utilisé durant la phase de la gestion de la mémoire (matrice) *mem* qui sera présentée ultérieurement dans la section 3.3.4.

Dans les paragraphes suivants nous détaillons le principe de fonctionnement de la partie dynamique de l'algorithme de décodage. Cette dernière se divise en deux grandes parties : la phase de latence et la phase d'apprentissage.

3.3.3. Phase de latence

Tout au long de cette phase, dont le principe de fonctionnement est montré à la Figure 3.5, la matrice *mem* reçoit de nouvelles valeurs. Pour cette fin nous représentons la matrice *mem* par des colonnes, comme suit :

$$mem = \left(C_0 \cdots C_{p-1}\right) avec\ C_{0\le i\le p-1} = \begin{pmatrix} m_{0i} \\ \vdots \\ m_{(l-1)i} \end{pmatrix} \tag{3.36}$$

Supposons que suite à une $j^{ème}$ séquence du modulateur nous obtenons S'_{est_j} en se basant sur l'équation (3.30). Le remplissage de la matrice *mem* se fait comme suit:

- Initialisation :

$$\begin{cases} C_0 = C_1 = \cdots = C_{p-1} = Zéro_{[1\times l]} \\ k_0 = 0 \end{cases} \tag{3.37}$$

- Remplissage de la 1ère colonne :

$$\begin{cases} C_0 = S'_{est_j}; \\ k_1 = 1; \end{cases} \tag{3.38}$$

- Remplissage de la 2ème colonne :

$$\begin{cases} Si\ S'_{est_j} \neq C_0\ alors\ C_1 = S'_{est_{j+1}}; \\ k_2 = 2; \end{cases} \tag{3.39}$$

- Remplissage de la 3ème colonne :

$$\begin{cases} Si\ S'_{est_j} \neq \{C_0, C_1\}\ alors\ C_2 = S'_{est_{j+2}}; \\ k_3 = 3; \end{cases} \tag{3.40}$$

- Remplissage de la $t^{ème}$ colonne :

$$\begin{cases} Si\ S'_{est_j} \neq \{C_0, C_1, \cdots, C_{k_t-1}\}\ alors\ C_{k_t} = S'_{est_{j+t}}; \\ k_{t+1} = k_t + 1; \end{cases} \tag{3.41}$$

- Remplissage de la colonne t_{max}:

$$\left\{ Si\ S'_{est_j} \neq \{C_0, C_1, L, C_{k_{t_{max}}-1}\}\ alors\ C_{k_{t_{max}}} = S'_{est_{j+t}}; \right. \tag{3.42}$$

Une valeur maximale t_{max} est attribuée au paramètre t dépendamment de la précision voulue. Si nous voulons à titre d'exemple que le système acquiert 5 valeurs différentes durant la phase de la latence, à ce moment t_{max} doit être égale à 5. À la fin de la phase de latence la matrice *mem* est comme suit :

$$\begin{cases} mem = \left(S_{est_0}, \cdots, S_{est_{t_{max}}}, C_{t_{max}-1}, \cdots, C_{p-1}\right) \\ avec\ S_{est_0} \neq S_{est_1} \neq \cdots \neq S_{est_{t_{max}}} \end{cases} \tag{3.43}$$

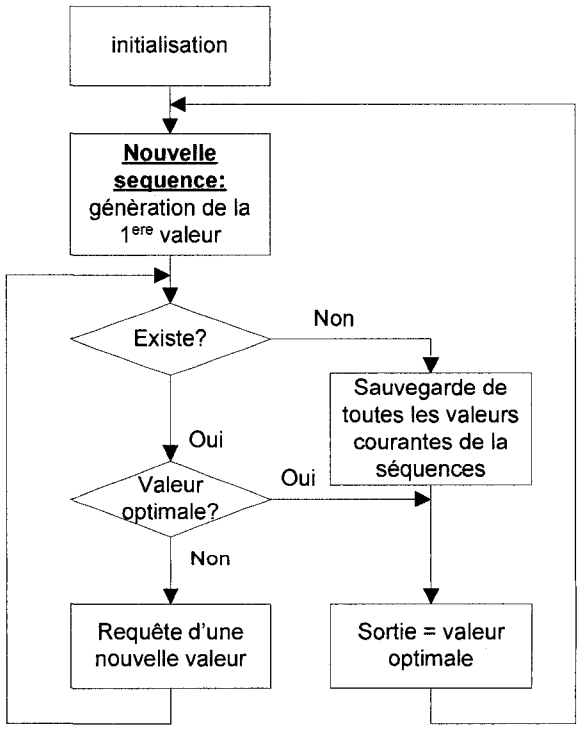


Figure 3.5. Algorithme d'apprentissage durant la phase de latence

3.3.4. Phase de décodage dynamique

Contrairement à la phase de latence, durant laquelle nous comparons des séquences en entier, durant cette phase la comparaison se fait au niveau des éléments de chaque séquence ce qui correspond à l'étape de vérification du nombre de valeurs suspectes dans l'algorithme de la Figure 3.4. Ainsi le décodage peut être arrêté à n'importe quelle itération si l'algorithme de décodage dynamique arrive à retrouver la séquence en question dans la matrice *mem*. En conséquence le temps de décodage se trouve réduit. Il est à noter que cette phase ne peut en aucun cas être entamée sans que la phase de latence ne soit achevée. Pour simplifier le calcul nous allons utiliser l'équation (3.36) au lieu de l'équation (3.43).

Maintenant supposons que le premier élément d'une séquence S'_{est_i} est le suivant $u_{est_i 0}$. Ci-dessous une description de l'algorithme proposé durant la phase de décodage dynamique.

- *Initialisation :*

  La matrice *mas* est initialisée à 0.

- *1ère étape (A) :*

$u_{est_i 0}$ est comparé à chaque élément de la ligne correspondante de la matrice *mem* c'est-à-dire la 1ère ligne (ligne 0). Si un élément ou plusieurs sont égaux à $u_{est_i 0}$, l'élément correspondant dans la matrice *mas* est mis à 1. Soit S le nombre d'éléments de « 1 » dans la matrice mas et dans la ligne 0

$$S = \sum_{i=0}^{i=p-1} M_{0i} \tag{3.44}$$

Si S=1, le décodage est arrêté sinon le système passe à la deuxième étape. L'exemple suivant illustre cette opération dans le cas où $u_{est_i 0} = 3$:

$$\begin{matrix} mem & & \end{matrix} \qquad \begin{matrix} mas & & \end{matrix}$$
$$\begin{pmatrix} 3 & 2 & 3 \\ 2 & 3 & 1 \\ 4 & 0 & 0 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{3.45}$$

S=2 et donc la deuxième étape est activée.

La matrice indexe correspondant à la matrice *mem* est :

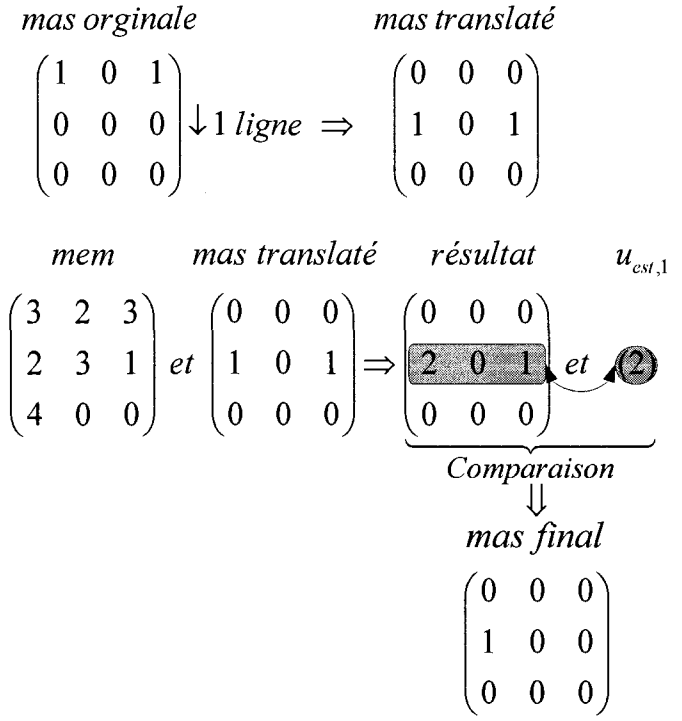
$$index = (3,2,2) \tag{3.46}$$

- *2ème étape (B) :*

La comparaison se fait entre $u_{cst_i\,1} = 2$ et la ligne suivante de la matrice *mem* c'est-à-dire la 2ème ligne. La matrice *mas* est translatée d'une ligne vers le bas. La comparaison s'effectue entre chaque élément de la 2ème ligne de *mem* et dont l'élément correspondant dans la matrice *mas* translatée est « 1 » avec $u_{cst_i\,1}$. Si les éléments en question sont égaux à $u_{cst_i\,1}$ alors l'élément correspondant de la matrice *mas* est maintenu à 1 sinon il est remis à zéro. *S* est recalculé à partir de la 2ème ligne de la matrice *mas*. L'exemple précédent sera poursuivi dans l'équation (3.47) pour illustrer ce fonctionnement.

Dans ce cas, *S*=1 et donc nous arrêtons le décodage. Et nous passons à l'étape finale : étape d'extraction des données **(C)**, sinon l'étape 2 est répétée jusqu'à ce que:

- $S = 0$: on passe à l'étape de remplissage de la matrice *mem*.
- $S = 1$ et à ce moment la nous passons à l'étape extraction.
- $S > 1$ et la dernière ligne de la matrice n'est pas atteinte, nous refaisons l'étape 2.
- $S > 1$ et la dernière ligne de la matrice est atteinte, à ce moment nous reprenons l'étape de remplissage de la matrice *mem*.



(3.47)

- *Étape d'extraction des données (C) :*

Cette étape n'est réalisée que si S=1. La valeur du signal analogique se trouve dans la colonne de la matrice *mem* correspondant à la colonne dans laquelle se trouve le seul élément égal à « 1 » dans la matrice *mas*. L'information utile qui correspond à la valeur du signal analogique correspond à la dernière valeur non nulle d'une séquence décodée S'_{est_i}. Cette dernière est indiquée par la matrice *index* dont le modèle est présenté dans l'équation (3.34). L'élément en question serait $u_{est_j max_j}$, j étant l'indice de la colonne contenant la valeur du signal analogique, max_j est l'élément de la matrice *indexe* dont l'emplacement est le même que celui de la colonne contenant l'élément égale à 1 dans la matrice *mas*.

Pour plus de clarté l'exemple précédent est encore repris :

$$mas\ finale = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad et\ index = (3,1,2) \tag{3.48}$$

L'unique élément qui est égal à 1 dans la matrice *mas* se trouve dans la 1ère colonne ce qui signifie que la valeur du signal analogique se trouve aussi dans la 1ère colonne de la matrice *mem* comme indiqué dans l'équation (3.49)

$$\begin{matrix} 1^{ère}\ colonne\ de\ mem \\ \begin{pmatrix} 3 \\ 2 \\ 4 \end{pmatrix} \end{matrix} \tag{3.49}$$

L'indice de la valeur du signal analogique dans la colonne précédente est le premier élément de la matrice indexe, parce que le seul élément égal à « 1 » dans la matrice mas se trouve dans la 1ère colonne, si ce dernier se trouvait dans la 2ème colonne on aurait pris le 2ème élément de la matrice indexe et ainsi de suite. Le 1er élément de la matrice indexe est égale à 3. Ainsi l'emplacement de la valeur du signal analogique se trouve dans la 1ère colonne et la 3ème ligne c'est-à-dire 4.

- *Étape de remplissage de la matrice mem (D) :*

Cette étape n'est activée que si la matrice *mem* ne renferme pas la valeur recherchée. À ce moment, on revient à l'étape du remplissage de la $t^{ème}$ colonne de la phase de latence, la phase d'initialisation et la condition de nombre maximal de séquences de la phase de latence sont ignorées. Il est à noter que durant la phase d'initialisation on ne remet pas les indices à zéro à la fin, ce qui permet un repositionnement correct dans la matrice *mem* une fois l'étape de remplissage de cette dernière est lancée à partir du processus de décodage dynamique. L'exemple de la figure 3.6 résume le cas précédent.

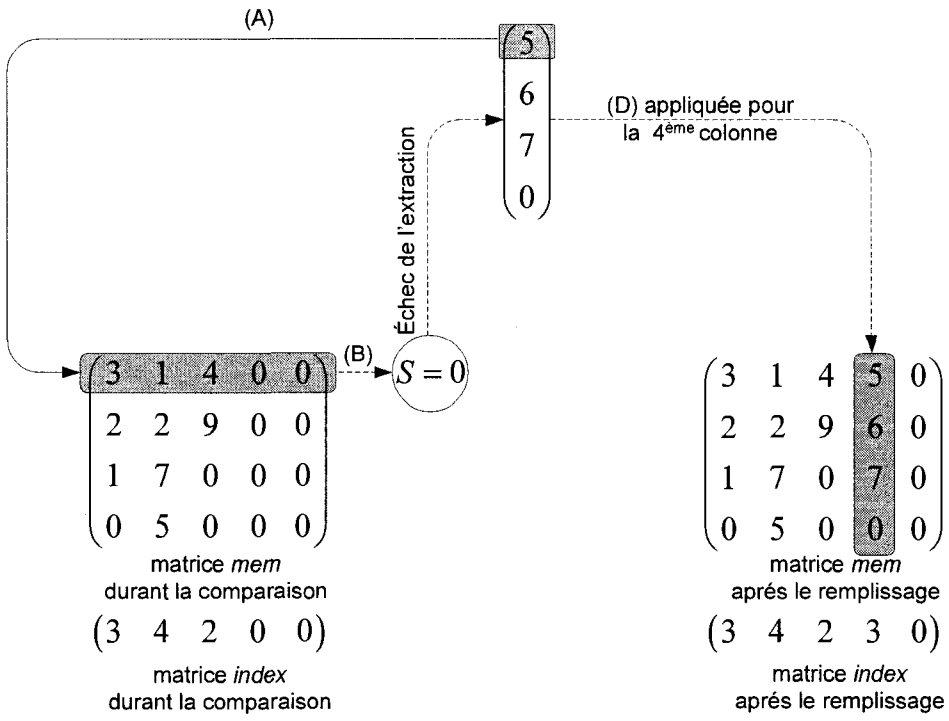


Figure 3.6. Exemple de remplissage dynamique de la matrice *mem*

Rappelons que si S est supérieur ou égal à 1 on passe soit à l'étape 2 ou à l'étape de l'extraction selon les explications fournies précédemment. Ainsi l'étape (C) ne figure pas dans l'exemple précédent étant donné que cette dernière n'est appliquée que si la séquence générée par le modulateur se trouve déjà dans la matrice *mem*, ce qui n'est pas le cas dans l'exemple présenté. Nous nous sommes abstenus de représenter l'étape d'initialisation vu que cette dernière est appliquée tout au début et elle consiste uniquement dans l'initialiser des indices.

Enfin pour résumer toutes les étapes précédentes la **Erreur ! Source du renvoi introuvable.** montre l'algorithme utilisé durant la phase de latence et la phase de décodage dynamique avec toutes les étapes intermédiaires et les interactions avec la partie de décodage statique.

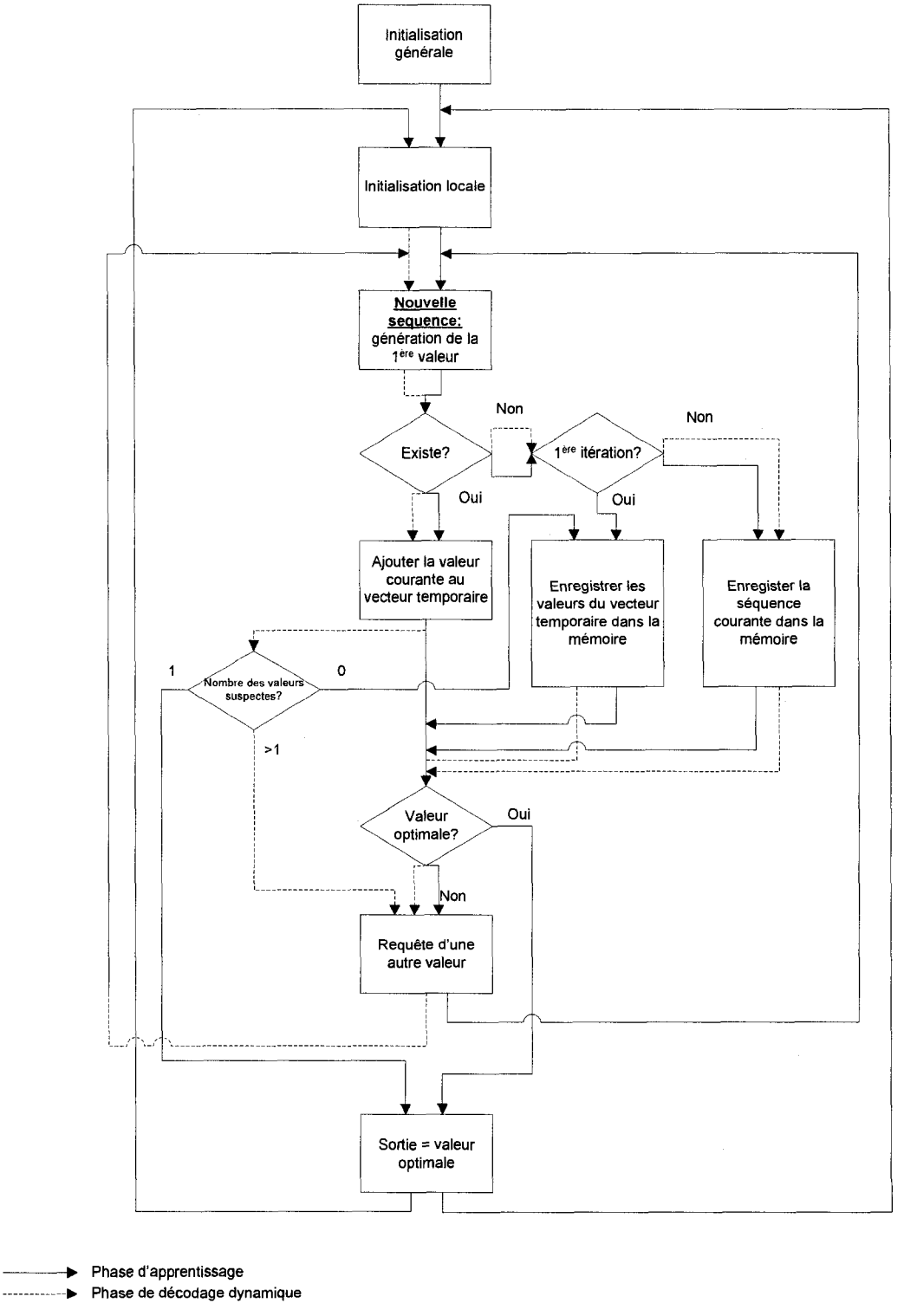


Figure 3.7. Algorithme de décodage dynamique

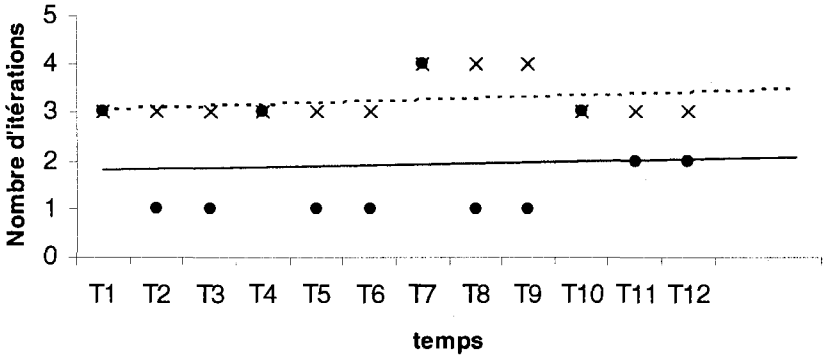
3.3.5. Exemple de décodage dynamique

Supposons que la matrice mem est remplie comme suit :

$$\begin{array}{ccccccc} S_{est_0} & S_{est_1} & S_{est_2} & S_{est_3} & S_{est_4} & S_{est_5} & S_{est_6} \\ \Downarrow & \Downarrow & \Downarrow & \Downarrow & \Downarrow & \Downarrow & \Downarrow \end{array}$$

$$\begin{pmatrix} 0.9545 & 0.9283 & 0.9714 & 0.9714 & 0.0698 & 0.0772 & 0.9283 \\ 0.5201 & 0.7325 & 0.5201 & 0.5218 & 0.3333 & 0.3514 & 0.7325 \\ 0.5594 & 0.7143 & 0.5641 & 0.5655 & 0 & 0.3750 & 0.7143 \\ 0 & 0 & 0.5714 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} = mem$$

Maintenant supposons que le modulateur ΣΔ envoient les séquences suivantes selon cet enchaînement : (S_{est_0}, t_1), (S_{est_0}, t_2), (S_{est_0}, t_3), (S_{est_1}, t_4), (S_{est_1}, t_5), (S_{est_1}, t_6), (S_{est_2}, t_7), (S_{est_2}, t_8), (S_{est_2}, t_9), (S_{est_3}, t_{10}), (S_{est_3}, t_{11}), et (S_{est_3}, t_{12}). Ou t_i représente les différents instants d'envoie tel que $t_i < t_{i+1}$.



Légende
- • Algorithme de décodage dynamique
- × Algorithme de décodage statique : Décodage cyclique
- — Approximation linéaire du nombre d'itération de l'algorithme dynamique
- --- Approximation linéaire du nombre d'itérations de l'algorithme de décodage statique : décodage cyclique

Figure 3.8. Décodage statique vs. Décodage dynamique pour des séquences de 40 bits

Comme nous pouvons le noter dans la Figure 3.8 le nombre d'itérations nécessaires pour le décodage dynamique est plus faible que dans le cas statique. Cependant nous notons des pics durant lesquels le nombre d'itérations est le même que dans le deux cas. Ceci s'explique par le fait que la séquence est nouvellement introduite dans la matrice *mem*. Subséquemment l'algorithme dynamique a besoin de procéder à un décodage en entier pour la première fois. Par la suite nous notons une accélération du processus de décodage de 3 à 4 fois plus vite que l'algorithme statique.

3.4. Conclusion

Dans ce chapitre, nous avons décrit la théorie du décodage dynamique proposé. Cependant lors de l'implémentation de cette dernière, certaines concessions doivent être faites à cause des limitations matérielles comme à titre d'exemple la taille de la mémoire qui n'est en aucun cas illimitée. Ainsi au cours du chapitre suivant nous aborderons ces limitations tout en présentant l'implémentation matérielle sur FPGA de l'algorithme précédemment décrit. L'approche de décodage dynamique est une approche heuristique permettant une réduction du temps de décodage. Cette réduction sera présentée dans le chapitre 5.

Chapitre 4
Architecture du module de décodage dynamique

4.1. Introduction

Après une description algorithmique de la technique de décodage dynamique, ce chapitre est consacré à l'aspect architectural de l'algorithme proposé. Pour faciliter l'introduction du module de décodage nous décrivons dans la première partie du chapitre le module de décodage statique avant de procéder à la mise ne œuvre du module de décodage dynamique. Ainsi à travers ce chapitre nous décrivons l'architecture de chaque module ainsi que sa spécificité.

4.2. Architecture de décodage statique

4.2.1. Rappel des étapes de l'algorithme de décodage statique

L'algorithme de décodage statique tel qu'expliqué dans le 3ème chapitre se résume dans les étapes suivantes:

1) détermination de l'$E_{courant}$ et de l'$E_{correctif}$;
2) génération d'une nouvelle séquence en se basant sur ces 2 éléments;
3) élimination des NCR;
4) calcul des cœfficients correspondant au niveau de décodage courant;
5) calcul des valeurs estimées de la séquence précédente et successive;
6) estimation de la valeur du signal analogique du niveau de décodage courant;
7) arrêt du décodage et déclenchement du cycle suivant.

Les 6 premières étapes se répètent jusqu'à atteindre le critère d'arrêt.

4.2.2. **Description générale**

Pour implémenter une telle architecture ,il nous a fallu 3 mémoires de même taille, dans lesquelles la séquence tout au long des 6 premières étapes est sauvegardée. L'utilisation de 3 mémoires différentes assure la disponibilité de la séquence en cours de décodage dans ses différentes formes dans une même itération. Les étapes 2, 3 et 6 dans la figure 4.1 sont les mêmes que celles qui ont été décrites précédemment. Ce module se base en partie sur des machines à états finis (MEF) qui manipulent des mémoires. Ces mémoires renferment la séquence en entier. Mais comme la taille de la mémoire se trouve limitée, nous avons implémenté des mémoires de : 6x40 bits = 240 bits. Autrement dit, chaque mémoire peut contenir au maximum une séquence de 40 éléments de 8 bits (ces mémoires sont disponibles dans le FPGA utilisé). Ceci est largement suffisant dans le cas du décodeur proposé étant donnée que dans le prototype développé nous considérons une séquence de longueur maximale de 40 éléments. Ces mémoires peuvent être étendues dépendamment des ressources matérielles disponibles. Dans ce cas précis elles dépendent du paramètre *n* (longueur de la séquence) telle que montré dans la figure 4.1. Les différents modules du décodage statique sont décrits dans les sections suivantes.

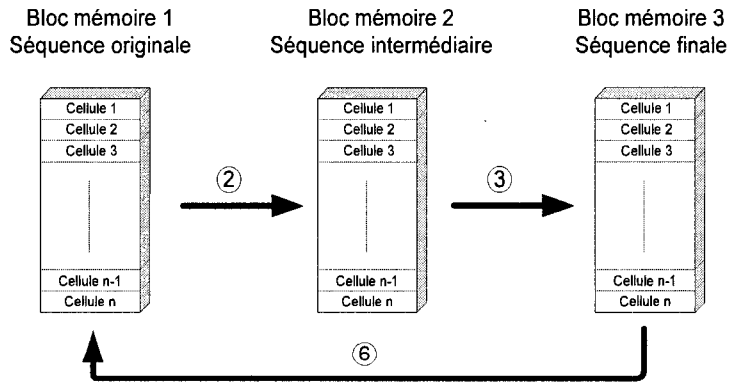


Figure 4.1. Architecture de base de la partie statique du décodeur

4.2.3. Module de détermination de l'$E_{courant}$ et de l'$E_{correcrif}$

C'est le cœur du décodage. S'il y a une erreur dans cette étape, tout le calcul qui s'en suit devient erroné. Étant donné qu'une séquence ne peut avoir deux $E_{correcrif}$ successifs, la technique de détermination de l'$E_{courant}$ et de l'$E_{correcrif}$ se base sur la redondance de l'$E_{courant}$. Ainsi les cas suivants peuvent se présenter :

- La séquence ne contient que des 1 ou des 0 : ceci est un cas particulier et est ignoré par cette étape. Un traitement spécial doit être effectué. Nous y reviendrons à la fin de ce chapitre.

- La séquence est de période 2 : $E_{courant}$ $E_{correcrif}$ $E_{courant}$ $E_{correcrif}$ $E_{courant}$ $E_{correcrif}$ $E_{courant}$ ...
  L'$E_{courant}$ correspond au 1^{er} élément et l'$E_{correcrif}$ correspond au $2^{ème}$ élément.

- Dans les autres cas, le traitement suivant est appliqué : deux registres appelés $E_{courant}$ et $E_{correcrif}$ sont initialisés respectivement au 1^{er} élément et au second élément de la séquence enregistrée dans la mémoire. La mémoire renfermant la séquence originale est balayée cellule par cellule. Chaque cellule est comparée à la précédente. Dès qu'il y a une redondance, le registre $E_{courant}$ prend la valeur de l'une de ces deux dernières cellules, alors que le registre $E_{correcrif}$ prend la première valeur différente de l'$E_{courant}$. Si l'$E_{courant}$ trouvé à la suite du balayage n'est autre que la valeur de l'$E_{correcrif}$ sauvegardé durant l'initialisation, alors la MEF inverse la valeur des registres $E_{courant}$ et $E_{correcrif}$. Cette dernière opération est assurée par le module présenté à la figure 4.2. Ce module sera noté RCS (Run and Correction Scan).

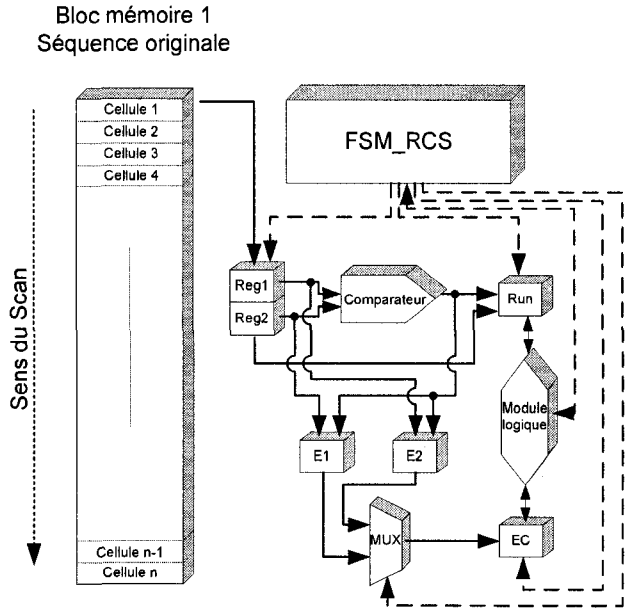


Figure 4.2. Module RCS

E_1 et E_2 sont deux registres intermédiaires qui permettent de sauvegarder les deux valeurs possibles de l'$E_{correctif}$. EC est l'$E_{correctif}$ de la séquence courante et le Run est l'$E_{courant}$. Nous notons la présence d'une entité appelée FSM_RCS qui assure le bon fonctionnement de chaque élément du module RCS. La FSM_RCS est en relation avec presque chaque élément pour synchroniser le tout avec le reste des modules de décodage. À ce niveau, le bloc mémoire 2 est encore vide. C'est au cours de l'étape suivante que ce dernier est activé.

4.2.4. Module de génération d'une nouvelle séquence

Connaissant à ce stade l'$E_{courant}$ et l'$E_{correcrif}$, il nous est possible de générer une séquence de deuxième niveau. Nous avons qualifié cette séquence d'intermédiaire. étant donné que celle-ci renferme des informations superflues, de sorte qu'elle doit subir un autre traitement pour générer la séquence finale.

Cette étape est complètement gouvernée par une machine à états finis que nous appellerons FSM_CNIS_NCR. Ci-dessous les opérations effectuées par cette MEF.

- Détection des $E_{correcrif}$, dans la séquence originale.
- Calcul du nombre des $E_{courant}$ entre chaque 2 $E_{correcrif}$.
- Placer chaque nombre dans une cellule du bloc mémoire 2 en respectant l'ordre de détermination de ce dernier.

La génération de la nouvelle séquence est assurée par la MEF qui :

- Génère l'adresse de lecture de chaque symbole dans le bloc mémoire 1.
- Calcule chaque symbole de la nouvelle séquence en fonction de l'originale.
- Détermine l'adresse de stockage pour chaque nouveau symbole dans la nouvelle mémoire.

À chaque fois que la MEF rencontre un $E_{correcrif}$ dans le bloc mémoire 1, elle incrémente l'adresse d'écriture du bloc mémoire 2. Par contre l'incrémentation de l'adresse de la cellule du bloc mémoire 1 se fait à chaque coup d'horloge de sorte que la nouvelle séquence est prête au bout de n cycles d'horloge, n étant la profondeur du bloc mémoire. La figure 4.3 montre l'interconnexion entre les deux blocs mémoires.

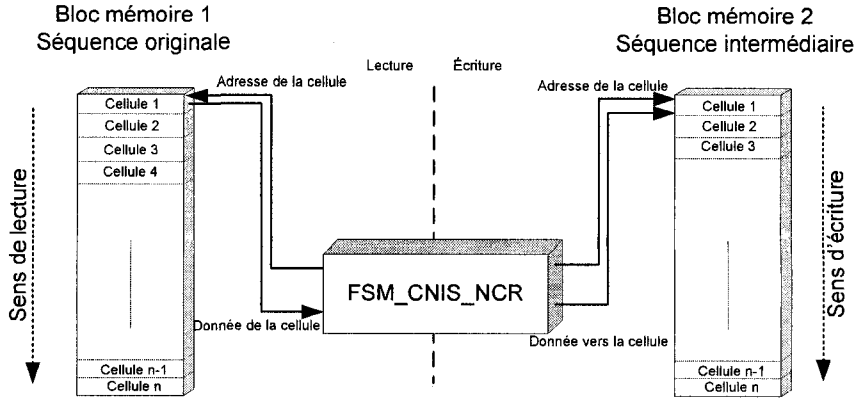


Figure 4.3. Module CNIS_NCR

4.2.5. Module de génération de la séquence finale

4.2.5.1. Détermination des NCR

Comme nous l'avons déjà précisé au cours du deuxième chapitre un NCR correspond à un $E_{courant}$ qui n'est pas encadré par deux $E_{correcrif}$. Un « Flag » est utilisé pour indiquer si un NCR à gauche de la séquence (en haut du bloc mémoire 2) a été trouvé. De même pour celui de droite. Il est à noter que la détermination du NCR se fait en se basant sur la séquence originale et non la séquence intermédiaire. C'est pour cette raison que cette étape est déclenchée en même temps que l'étape 2.

Comme montré dans la figure 4.4, cette étape se concentre uniquement sur les 3 premiers éléments de la séquence et par la suite les 3 derniers.

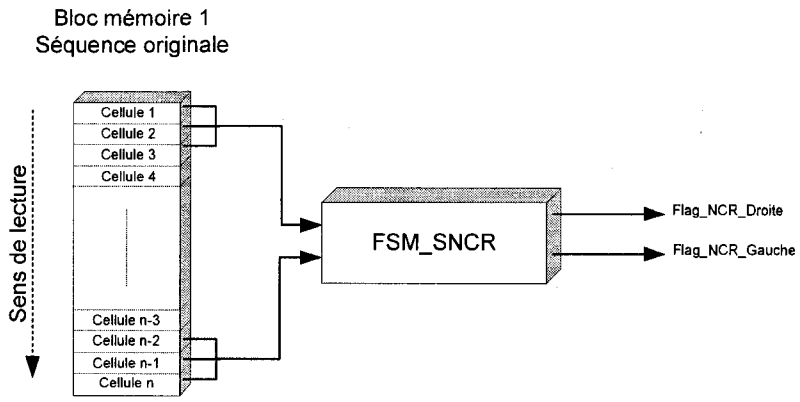


Figure 4.4. Détermination des NCR

4.2.5.2. Élimination des NCR

Les deux « Flags » de la figure 4.4 conditionnent le traitement effectué durant cette étape tel que montré dans la figure 4.5. Ainsi :

- si le *Flag_NCR_droite* est à 1 alors seule la première cellule du bloc mémoire 2 est supprimée,

- si le *Flag_NCR_gauche* est à 1 alors seule la dernière cellule du bloc mémoire 2 est supprimée,
- si les deux sont à 1 alors la première et la dernière cellule sont éliminées.

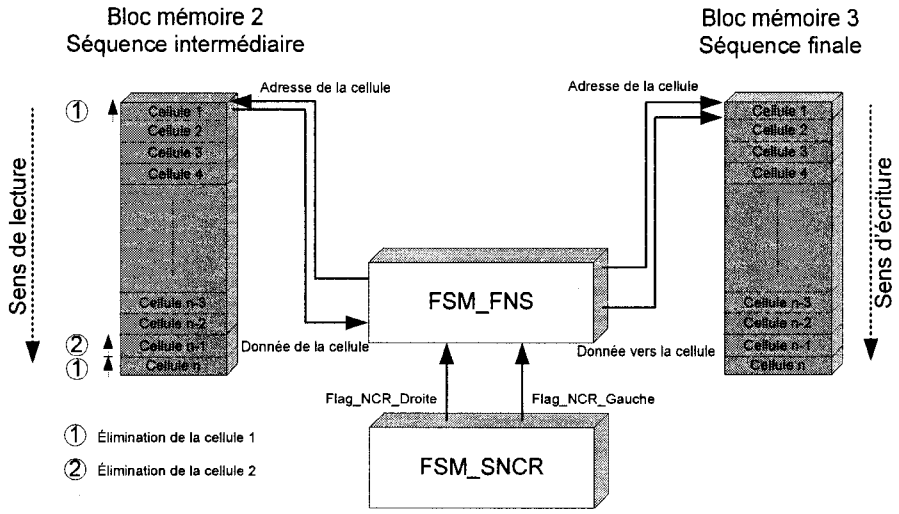


Figure 4.5. Génération de la séquence finale

① correspond au cas où la 1ère cellule de la séquence est un NCR.

② correspond au cas où la dernière cellule de la séquence est un NCR.

Chacune de ces étapes conduit à une élimination de la case mémoire correspondant à la cellule en question. Ces deux dernières peuvent s'additionner et aboutir à une élimination de deux cases mémoires : correspondant respectivement à la première cellule de la séquence ainsi qu'à la dernière de celle-ci. C'est pour cette raison que la longueur utile de la mémoire peut être soit n, n-1 ou bien n-2. La longueur utile de la mémoire est défini comme suit : *Étant donné que la taille de la mémoire est fixe et que celle de la séquence qui est enregistrée dans cette dernière est variable alors la longueur utile de la mémoire correspond à celle de la séquence qui est égale à la taille de la mémoire à la première itération du décodage, par la suite elle peut être inférieure ou égale à cette dernière.*

Maintenant que la séquence finale est enfin prête, les calculs peuvent être entamés pour générer les cœfficients $w_{r,i}$, $w_{c,i}$, $l_{r,i}$ et $l_{c,i}$.

4.2.6. Module de calcul des coefficients

Cette étape consiste en une simple multiplication et addition tel que montré dans les équations (3.16) et (3.17).

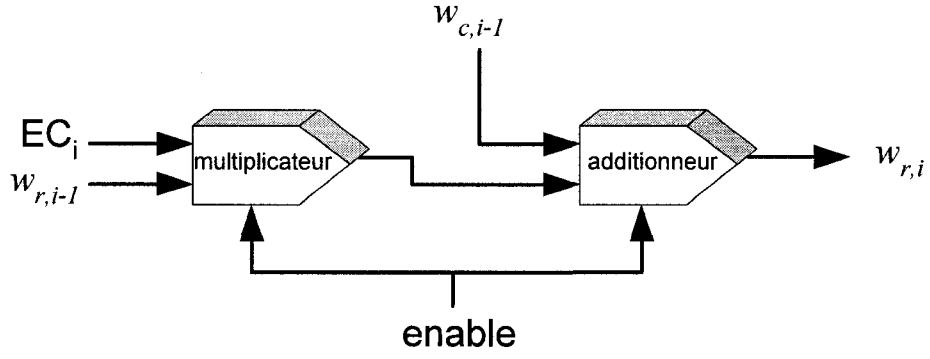


Figure 4.6. Élément du module de génération des nouveaux coefficients

La figure 4.6 est représente un des quatre éléments du module qui calcule les 4 cœfficients : $w_{r,i}$, $w_{c,i}$, $l_{r,i}$ et $l_{c,i}$. De la même façon nous implémentons trois autres éléments en utilisant le même modèle pour les autres coefficients.

4.2.7. Module de calcul de la somme des éléments de: la séquence précédentes, courante et suivante

En premier lieu il faut faire la somme des éléments de la séquence courante. Un module nommé accumulateur effectue cette opération. En se basant sur les définitions des prédécesseurs et successeurs dans les équations (3.14) et (3.15), le calcul de la somme des éléments respectifs de chacune de ces séquences se fait comme suit :

- La somme des éléments de la séquence courante moins 1 pour le prédécesseur, la séquence courante étant la séquence de référence de l'itération courante.
- La somme des éléments de la séquence courante plus 1 pour le successeur.

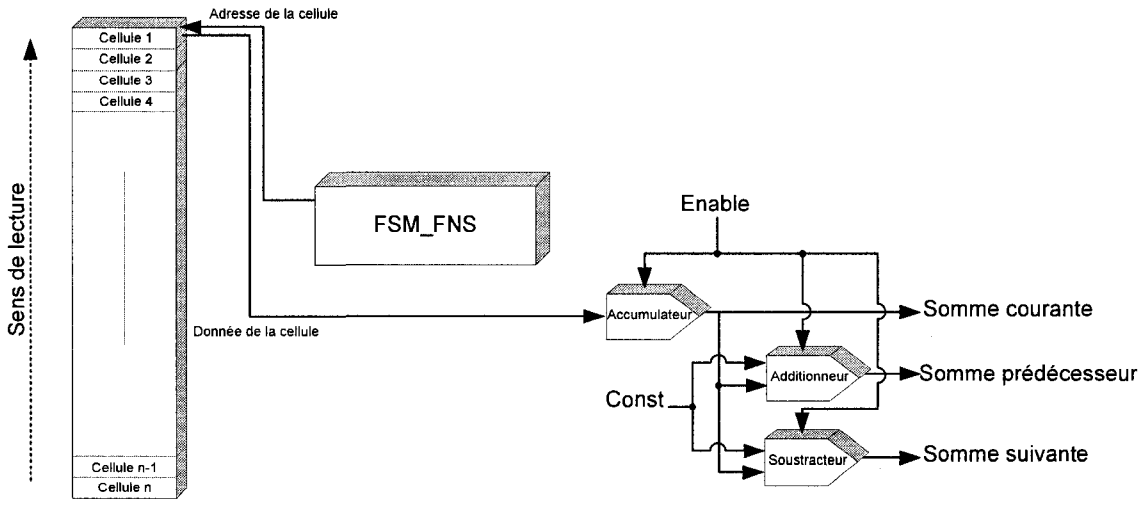


Figure 4.7. Module de calcule des sommes

Le mot somme dans la figure 4.7 fait référence aux éléments de la séquence en question. Le signal *Enable* permet de contrôler ce module et de le synchroniser avec les autres.

4.2.8. Module de pré estimation de la valeur du signal analogique

Cette étape est la dernière dans une itération de décodage. Elle consiste à additionner la somme des éléments de la séquence « précédente, courante et suivante » et de faire la division par 3 pour calculer la moyenne comme indiqué dans la figure 4.8. Ce module bien qu'il paraisse simple, son implémentation est relativement complexe étant donné que le module de division doit être générique pour pouvoir

s'adapter à la précision voulue. Cette problématique a été résolue en développant une architecture de division dont l'algorithme utilise le principe de la division par décalage et soustraction tel que montré dans l'annexe A.

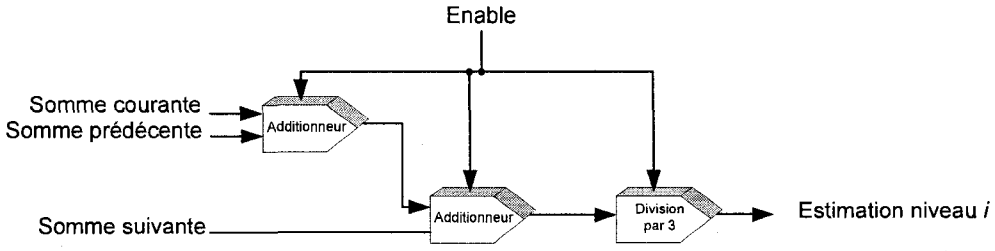


Figure 4.8. Module de pré estimation

4.2.9. Arrêt du décodage et lancement du cycle suivant

Arrivé à cette étape, le décodeur doit prendre une décision :

- Soit lancer une nouvelle itération de décodage pour une meilleure estimation de la valeur du signal analogique
- Soit arrêter le décodage en cours vu que la valeur du signal analogique a été retrouvée. Ensuite, lancer un autre cycle de décodage pour une nouvelle séquence.

Les critères d'arrêt sont cités dans 3.2.3. Une machine à états finis permet d'assurer un tel contrôle.

4.2.10. Connexion entre les différents modules

La synchronisation des opérations des différents modules est assurée par une machine à états globales activant chaque module à un instant donné dépendamment de l'évolution de décodage et ceci en respectant l'enchaînement de la figure 4.9.

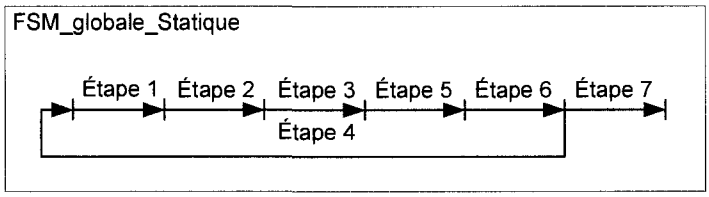


Figure 4.9. Enchaînement des différentes étapes de décodage statique

À ce stade le module statique passe le relais au module dynamique pour accélérer le processus de décodage. La description de ce module dynamique est l'objet de la section suivante.

4.3. Architecture de la partie dynamique du décodage

Durant cette phase, le décodeur fait appel à des ressources en mémoire pour accélérer le temps de décodage tel que montré à la Figure 4.10 . Puisque nous disposons de plusieurs niveaux de décodage, cette mémoire est divisée en plusieurs secteurs. Dans le cas présent, la mémoire implémentée sur FPGA est composée de 5 secteurs vu que les ressources du FPGA sont limitées. Chaque secteur est attribué à un niveau de décodage tel que montré à la figure 4.11.

À chaque bloc mémoire renferme les valeurs des séquences à la première itération de décodage. C'est-à-dire le bloc 1 contient l'ensemble des valeurs décodées à la première itération et ainsi de suite. La mémoire est composée de 5 blocs étant donné que dans le cas présent la valeur du signal analogique est retrouvée après un maximum de 4 itérations.

En plus de cette mémoire centrale, une autre mémoire connexe appelée index et montrée dans la Figure 4.12 à celle ci a été implémentée et dans laquelle le nombre maximal d'itérations pour chaque séquence est enregistré.

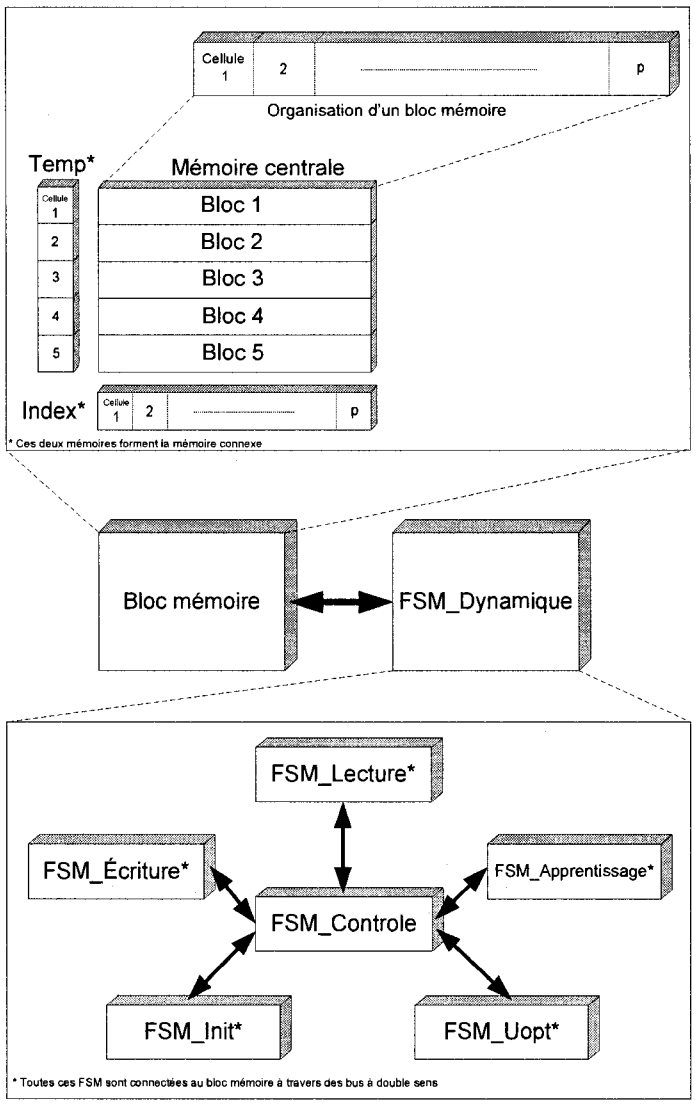


Figure 4.10. Diagramme bloc global du module dynamique

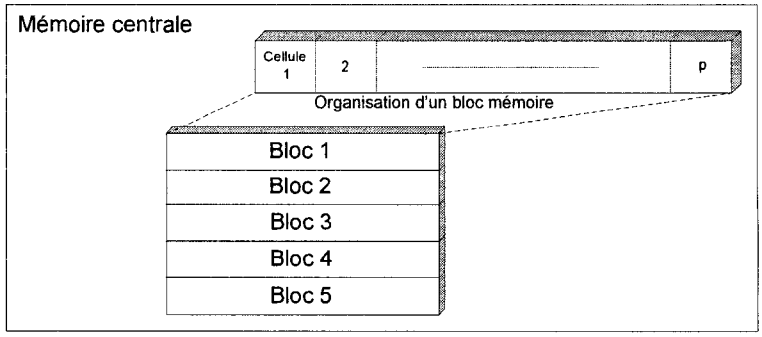


Figure 4.11. Organisation de la mémoire centrale du décodeur dynamique

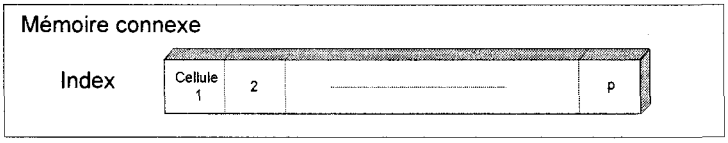


Figure 4.12. Mémoire connexe pour les index

La gestion dynamique de cette mémoire est assurée par un masque tel que montré dans l'équation (3.35). L'architecture équivalente de la matrice (3.35) serait une mémoire contenant 5 blocs et chaque bloc renfermerait p bits. Pour réduire l'espace utilisé du FPGA cette même mémoire est remplacée par un seul bloc avec une boucle de rétroaction incluant une logique combinatoire tel que montré dans la Figure 4.13.

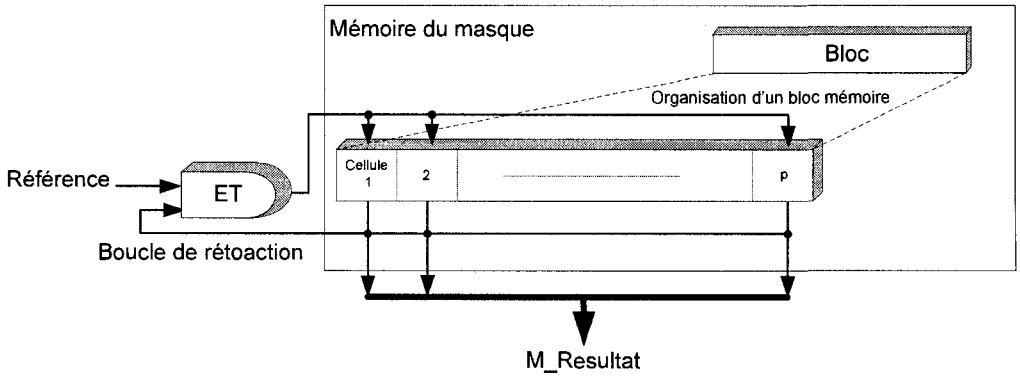


Figure 4.13. Module de masquage

La figure 4.13 présente l'architecture du module de masquage élaboré. L'entrée *Référence* est un signal indiquant l'emplacement des valeurs égales à la valeur décodée de l'itération en précédente dans le bloc mémoire correspondant tel que montré dans la figure 4.14. Le signal *val* est un bus de $p \times 14$ bits connecté d'une part à p registres dont le contenu est la valeur décodée à l'itération courante et de l'autre part à un comparateur globale. Le comparateur global est en fait un p comparateurs à 14 bits qui comparent le contenu de chaque cellule du bloc mémoire sélectionné par la MEF dynamique à la valeur décodée.

Le résultat de la comparaison se trouve sur un bus dont chaque bit indique si la cellule correspondante du bloc mémoire courant renferme la valeur recherchée ou non. Si c'est le cas alors le bit correspondant prend la valeur « 1 » dans le cas contraire il reçoit la valeur « 0 ».

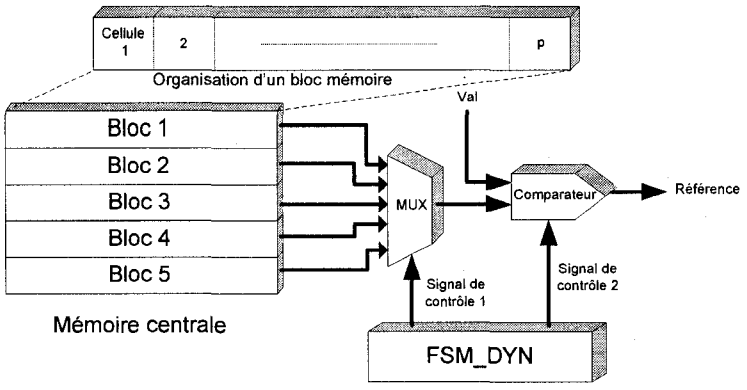


Figure 4.14. Module de comparaison

Le nombre de bits qui sont égales à 1 du signal *M_Resultat* est calculé à l'aide d'un additionneur. Si le résultat de l'addition est égal à 1, la recherche s'arrête sinon une requête pour procéder à une nouvelle itération est envoyée à la partie statique du décodeur afin de réduire le champ de recherche. L'additionneur en question est un module d'addition de 1 bit dont la première rangée est connectée au signal *M_resultat* du module de masque dans la figure 4.13.

Une MEF (*FSM_Lecture*) qui active la lecture si le résultat de l'Additionneur est strictement supérieur à 1. Autrement l'écriture serait activée si le résultat de l'addition est égal à 0 et ceci à travers une autre MEF d'écriture *(FSM_Ecriture)*. Cette dernière enregistre l'ensemble des valeurs estimées de la séquence d'entrée dans la colonne correspondante de la mémoire centrale.

Il est à noter que, tant que la valeur du signal analogique échantillonnée n'a pas été retrouvée les différentes valeurs estimées au cours des différentes itérations sont enregistrées dans un registre à part pour les transférer vers la mémoire centrale si la valeur n'a pas été retrouvée.

La mémoire connexe « *indexe* » est utilisée par la *FSM_Lecture* pour retrouver la position de la valeur estimée du signal d'entrée dans la mémoire. Le signal *M_Resultat* de la figure 4.14 indique la position de l'indexe à prendre en compte. En effet, si le signal *Add_Resultat* est égal à 1, ce qui correspond au seul cas ou le signal *M_Resultat* renferme un seul bit égal à 1. La position de ce bit correspond à la position de la cellule contenant l'indexe en question dans la mémoire connexe « *indexe* ».

Finalement la Figure 4.10 montre les interconnexions entre les différents blocs de l'architecture dynamique. Il est à noter que les mémoires n'ont pas été représentées pour simplifier la figure. La mémoire *Temp* est un « buffer » qui garde les valeurs intermédiaires d'une séquence en cours de décodage jusqu'à ce qu'une décision puisse être prise :

- Enregistrer l'ensemble des valeurs et à ce moment le contenu de *Temp* est transféré vers la mémoire centrale
- Valeur trouvée dans la mémoire centrale et à ce moment le contenu de *Temp* est vidé pour enregistrer les valeurs intermédiaires de la nouvelle séquence.

Ci-dessous un résumé de la fonction assurée par chaque MEF de la Figure 4.10 :

- *FSM_Ecriture* : Son rôle consiste en l'écriture des valeurs estimées des différents niveaux pour une séquence donnée dans la colonne adéquate de la mémoire centrale.

- *FSM_Lecture* : Cette MEF charge une colonne à la fois de la mémoire centrale dans un registre intermédiaire pour procéder à une comparaison instantanée avec toutes les valeurs enregistrées dans cette colonne en un seul coup d'horloge. En effet pour accélérer la vitesse de traitement, un registre intermédiaire enregistre les valeurs de la colonne en question de la mémoire, ainsi. il est à noter que le décodeur requiert 10 coups d'horloges pour charger le contenu d'un bloc mémoire et d'un dernier pour la comparaison (la mémoire utilisée est une SRAM de 42×10×5 bits). Ceci au lieu de 20 cycles si cette comparaison se fait une cellule à la fois. Pour accélérer encore plus la vitesse de traitement une cellule mémoire contient en fait 3 valeurs distinctes ce qui explique le nombre 42 (3 × 14 bits), 14 bits est la longueur de chaque valeur estimée.
- *FSM_Init* : Elle est dédiée à l'initialisation de tous les mémoires.
- *FSM_Uopt* : Cette MEF est utilisée pour la lecture de la valeur décodée de la séquence associée au signal échantillonné. De plus elle commande la lecture de la mémoire centrale et de la mémoire connexe (index) et doit sélectionner les 14 bits adéquats car une seule cellule de la mémoire centrale renferme 3 valeurs distinctes (14×3).
- *FSM_Apprentissage* : Cette MEF gère le fonctionnement de celle de l'écriture et de la lecture durant la phase d'apprentissage (latence) vu qu'après chaque lecture il y a une écriture.
- *FSM_Controle* : Son rôle est d'assurer le bon fonctionnement de toutes les MEF de la partie dynamique en contrôlant leur activation et désactivation.
- *FSM_Dynamique* : c'est l'entité qui gère toutes les MEF énumérées précédemment mais en plus elle assure l'interface et l'interconnexion avec la partie de décodage statique.

4.4. Conclusion

Une description détaillée de chaque module a été exposée dans ce chapitre. Dans la section suivante nous présentons les résultats montrant l'accélération du temps de conversion en utilisant l'architecture dynamique que nous avons développée par rapport à l'algorithme de décodage itératif original [34]. Ces résultats mettent en évidence l'autonomie du décodeur puisque sa vitesse de décodage dépend du signal analogique échantillonné ainsi que des signaux précédent. Cette caractéristique a motivé la mise en œuvre d'une architecture de décodage dynamique pour les convertisseurs $\Sigma\Delta$.

Chapitre 5
Résultats de l'implémentation

5.1. Introduction

Après avoir présenté l'algorithme du décodage dynamique, ainsi qu'une mise en œuvre de l'architecture dans le 3ème chapitre, nous rapportons dans cette section les résultats des simulations et ceux provenant des validations expérimentales suite à l'implémentation sur FPGA. Une comparaison de l'architecture dynamique proposée avec celle de l'algorithme de décodage statique comme celui de Dachselt et al. [34] a été également établie pour mettre en évidence les améliorations apportées.

5.2. Simulation fonctionnelle

Dans cette section, les simulations fonctionnelles faites à l'aide du logiciel Modelsim 5.8 sont présentées. Ces simulations ne donnent pas la précision recherchée de l'architecture une fois implémentée sur FPGA, mais elles permettent de valider le fonctionnement de cette dernière. La figure 5.1 montre une simulation fonctionnelle du décodeur pour une séquence arbitraire de 40 bits générée par un modulateur ΣΔ d 1er ordre 1010101011010101101010101101010101101010. Le résultat obtenu en utilisant le décodeur simulé est 5593. Nous noterons que le résultat du décodage est toujours multiplié par 1000 pour éviter de représenter des nombres avec des virgules nécessitant un très grand espace mémoire. Par suite, Le résultat final sera 0.5593. Celui obtenu par Dachselt et al. [34] est de 0.5594. La différence est due à l'arrondissement par excès, à titre d'exemple si la valeur décodée est 0.55937 la valeur qui sera retenue par le décodeur est 0.5594. Quant à la figure 5.2 elle présente une simulation fonctionnelle pour des séquences de 40 bits dans les deux cas particuliers suivants :

- La séquence à décoder est une constante (1 ou 0).
- La séquence est une suite de 1 et de 0 de période 2.

L'approche dynamique de décodage se manifeste dans ces deux cas par la diminution du temps de traitement des données. En effet après 3.2 ms, le temps de décodage a été réduit approximativement de 3 fois : de 161.3 µs dans un cas de décodage statique à 66.0 µs dans un cas de décodage dynamique. Nous notons également que l'accélération dans la figure 5.1 se fait après 3.2 ms ce qui correspond au temps d'apprentissage. Ce temps d'apprentissage a été prédéfini par le nombre de séquences décodées. En effet, dans le cas présent, après 20 séquences le circuit se met en mode de décodage dynamique. La figure 5.3 est une simulation fonctionnelle du décodeur dynamique durant laquelle 40 séquences différentes de 12 bits ont été traitées. Des séquences Ces dernières se trouvent dans le tableau 2.1. Il est clair que la fréquence d'échantillonnage varie dépendamment de la séquence. En effet le signal data_Ready qui n'est pas périodique indique si le processus de décodage a été achevé ou non. Les accélérations (encerclées dans la figure 5.3) correspondent à des cas particuliers où aucun décodage ne s'impose. Une valeur préenregistrée est automatiquement affichée si une des séquences particulières est envoyée au décodeur (ces exceptions consistent en une séquence formée par des « 0 » ou des « 1 » seulement ou des « 0 » et des « 1 » de période 2).

5.2.1. Simulation post placement et routage

Le comportement du décodeur à la suite du placement et routage est exactement le même que celui de la simulation fonctionnelle comme présenté dans la Figure 5.4. Ceci prouve le bon fonctionnement du circuit une fois implémenté sur FPGA. Un test sur le circuit une fois implémenté sur FPGA s'impose pour valider l'architecture proposée, ce qui fera l'objet des sections suivantes. Il est à noter également que toutes les simulations ont été faites à une fréquence de fonctionnement de 10 MHz.

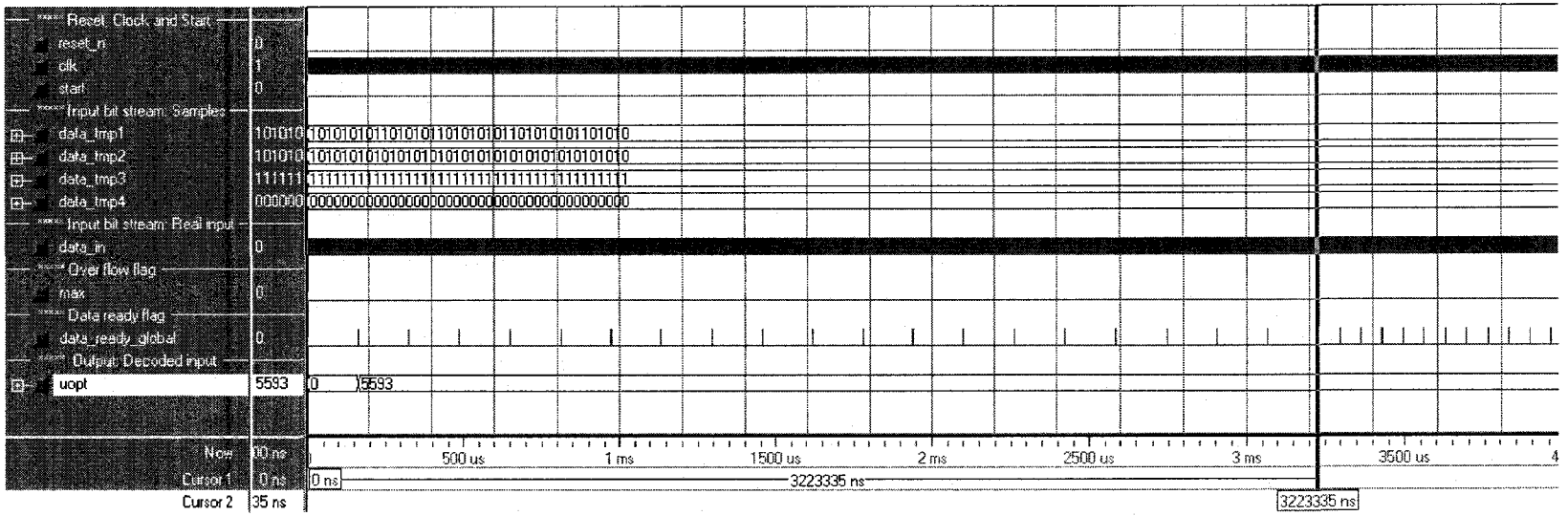


Figure 5.1. Simulation fonctionnelle : séquence 10101010110101011010101011010101011101010

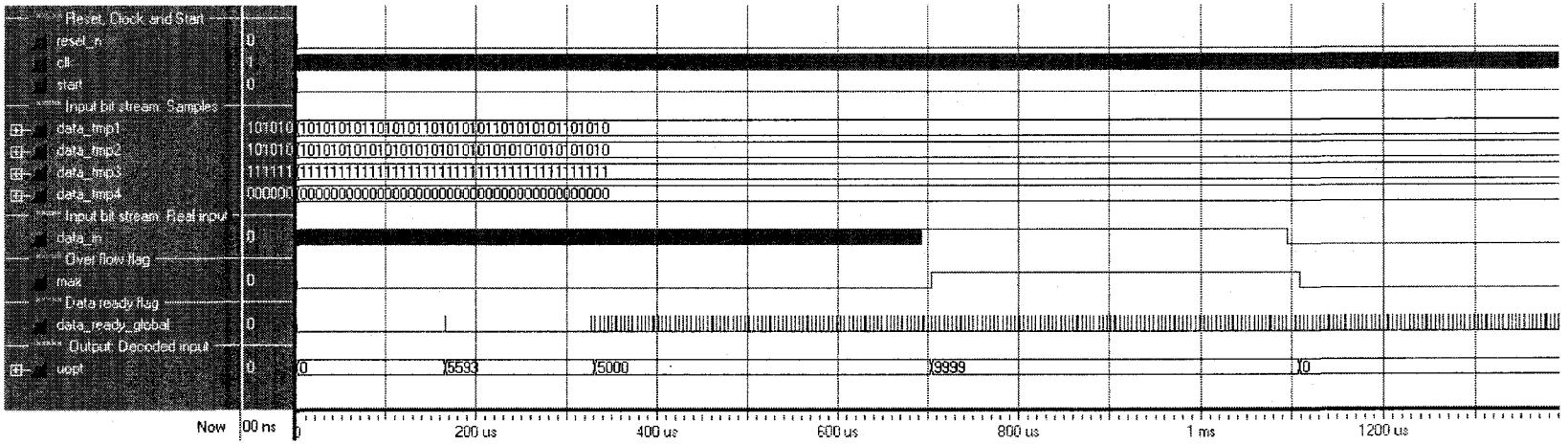


Figure 5.2. Simulation fonctionnelle: cas particuliers : la séquence à décoder est une constante (1 ou 0) ou la séquence est une suite de 1 et de 0 de période 2.

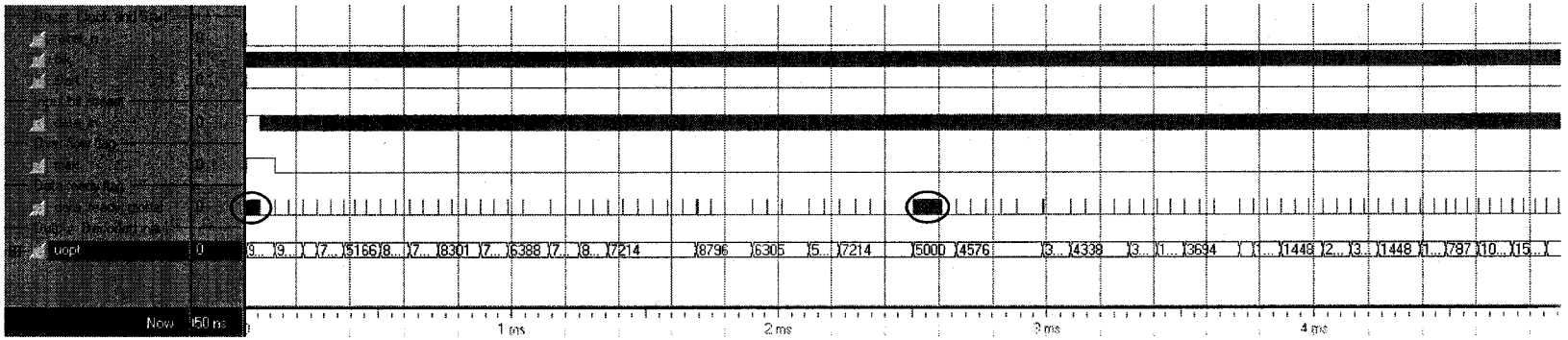


Figure 5.3. Simulation fonctionnelle: séquences de 12 bits

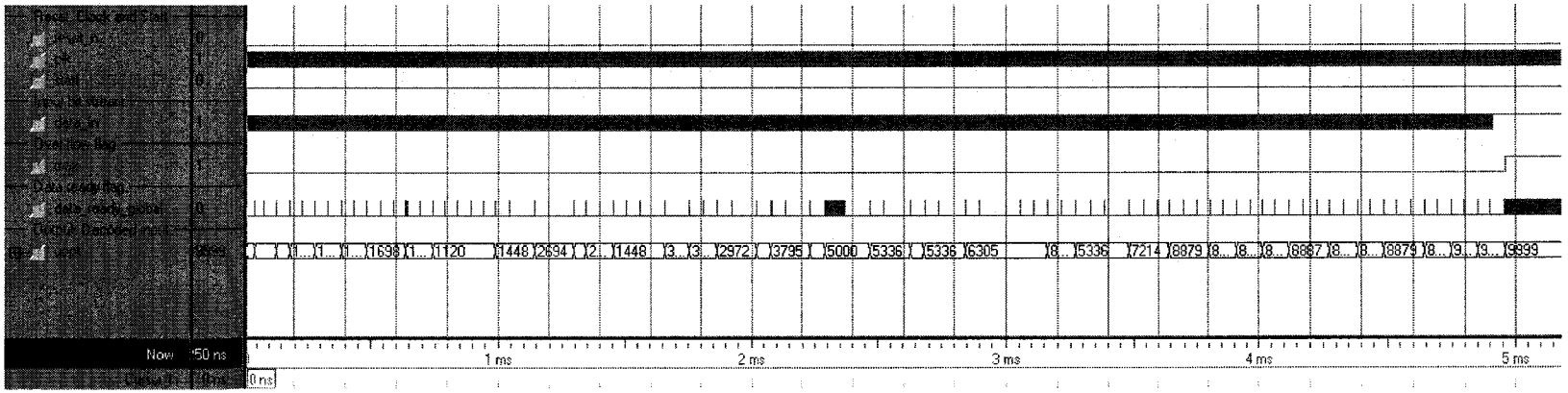


Figure 5.4. Simulation post placement et routage : séquences de 12 bits

5.3. Implémentation matérielle

L'implémentation du décodeur sur FPGA implique quelques contraintes. L'espace disponible sur le FPGA n'étant pas illimité, l'espace occupé par le circuit doit être minimisé. La fréquence de fonctionnement de ce circuit peut également se trouver réduite à cause des délais induits par les divers éléments logiques, mémoires et interconnexions du circuit. Bien sûr la consommation d'énergie est un paramètre primordial dans un circuit, en conséquence une estimation de cette dernière s'impose. Ainsi la section suivante traitera de l'espace occupé par l'architecture proposée dans le FPGA, sa fréquence de fonctionnement et enfin la puissance consommée.

5.3.1. Espace occupé

L'implémentation matérielle de l'architecture proposée a été réalisée sur un FPGA d'Actel : Fusion AFS 600 montré dans l'annexe B. Dans la Figure 5.5 une image du circuit numérique implémenté pour des séquences de 40 bits. De plus, la figure 5.6 est une configuration de décodage statique présenté au 3ème chapitre. Le tableau 5.1, et les figures 5.5 et 5.6 montrent la différence d'espace occupé dans le FPGA entre l'architecture proposée dans le cas d'une configuration de 12 bits et 40 bits et l'architecture de décodage statique. Nous constatons que les ressources en modules logiques sont plus importantes de même que les ressources mémoires dans le cas d'une architecture dynamique. Ceci confirme que l'architecture dynamique se base sur un traitement des données rétroactif et exige une logique plus élaborée ainsi qu'un espace mémoire plus important. Cependant la mémoire peut être réduite ou augmentée dépendamment de la précision voulue et de l'application, dans le cas présent la précision de décodage est de 1/10000. Si le concepteur utilise l'architecture dans une application durant laquelle les séquences ne vont pas varier énormément, il serait inutile de disposer d'une grande mémoire. L'augmentation de l'espace utilisé en modules logiques s'explique par l'addition d'un module de gestion des différentes mémoires.

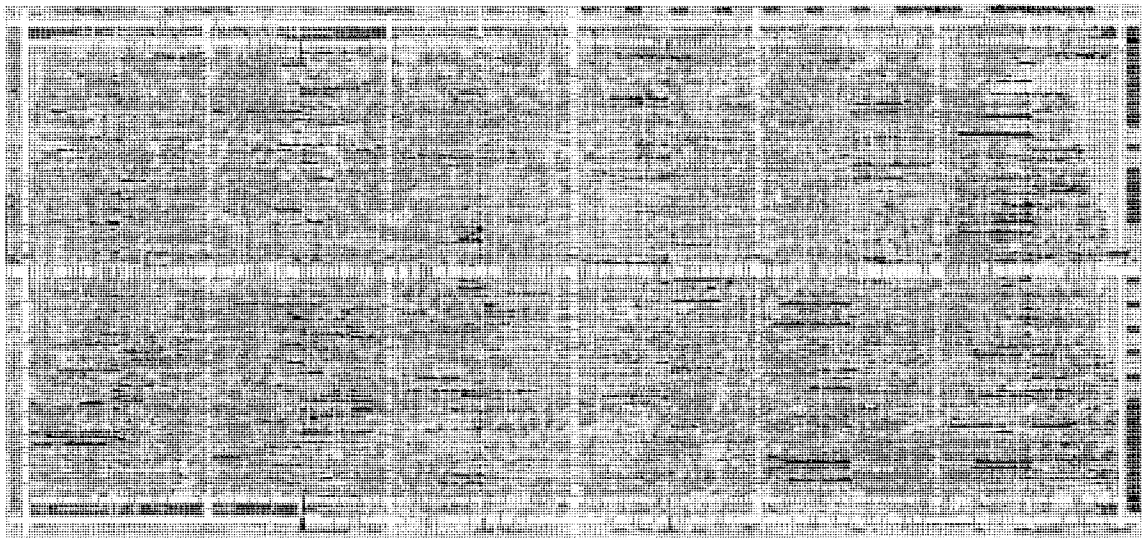


Figure 5.5. Image du FPGA d'un décodeur de 40 bits

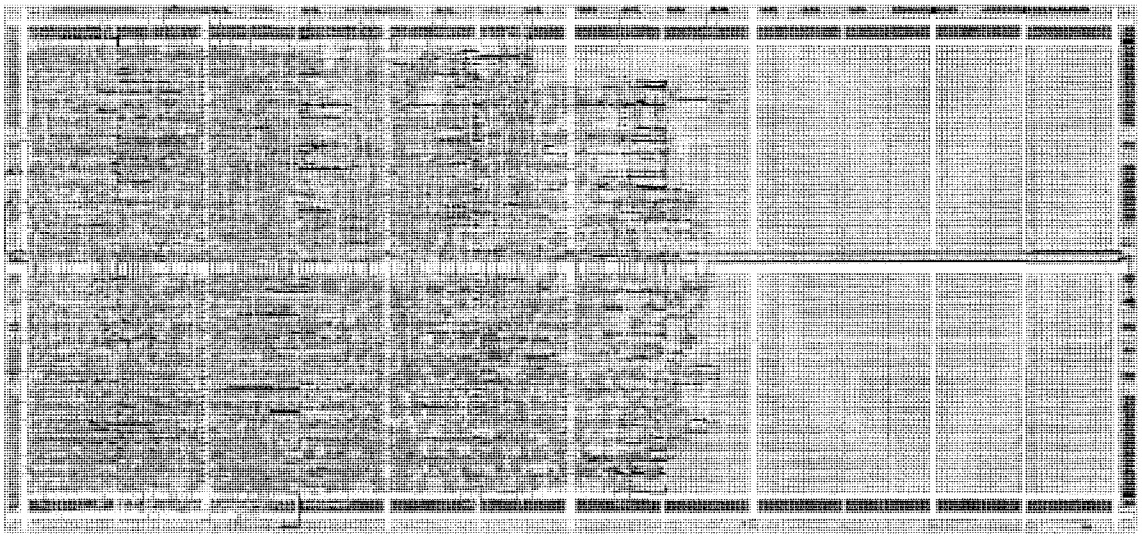


Figure 5.6. Image du FPGA d'un décodeur statique de 40 bits

Tableau 5.1. Ressources matérielles principales utilisées

	Architecture de décodage statique de 40 bits	Architecture de décodage dynamique de 12 bits	Architecture de décodage dynamique de 40 bits
modules logiques (%)	49.83%	86.57%	86.44%
mémoires (%)	8.33%	79.17%	79.17%

5.3.2. Vitesse d'opération du circuit

Ce paragraphe est un résumé des performances temporelles de l'architecture proposée.

Tableau 5.2. Fréquences maximales pour différentes architectures

	Architecture de décodage statique de 40 bits	**Architecture de décodage dynamique de 12 bits**	**Architecture de décodage dynamique de 40 bits**
Fréquence maximale de fonctionnement	43.31 MHz	37.43 MHz	37.50 MHz

Nous remarquons dans le tableau 5.2 que les fréquences estimées à l'aide de SmartPower d'Actel sont presque les mêmes à l'exception de l'architecture statique dont la fréquence dépasse de quelques MHz. Ceci s'explique par l'addition d'un module de gestion dynamique. Bien que la logique de ce dernier ait été optimisée pour qu'elle soit la plus simple possible, nous ne pouvons éviter une légère diminution de la fréquence qui reste cependant sans conséquence. Ceci est confirmé lors de La validation expérimentale qui sera présentée ultérieurement dans ce chapitre.

5.3.3. Consommation d'énergie du module proposé

Le tableau 5.3 présente l'énergie consommée dans le cas de l'architecture proposée qui est de 71 mW et dans le cas de l'architecture de décodeur statique où elle est de 32 mW. Cette différence est tout à fait naturelle, en effet elle est due principalement à l'augmentation des ressources mémoires tel que montré dans l'annexe C. Dans les paragraphes qui suivent nous présentons une comparaison entre les estimations faites par SmartPower et les valeurs réelles pour les différents paramètres cités précédemment.

Tableau 5.3. Consommation d'énergie des différents modules estimée par SmartPower

	Architecture de décodage statique de 40 bits	Architecture de décodage dynamique de 12 bits	Architecture de décodage dynamique de 40 bits
Puissance consommée	32.17 mW	71.14 mW	71.10 mW

5.4. Résultats expérimentaux

Suite à l'implémentation de l'architecture proposée, il nous a été impossible d'avoir une mesure précise de l'espace occupé par le FPGA. Ainsi, nous avons du nous fier aux résultats fournis par l'outil de conception d'Actel. Les mesures effectuées concernent principalement une vérification du bon fonctionnement du circuit réalisé, sa puissance consommée et sa fréquence maximale de fonctionnement.

5.4.1. Fonctionnement du circuit

Comme nous pouvons le noter aux Figure 5.8, le circuit de l'architecture suite à son implémentée donne exactement le même résultat que celui obtenu par la simulation post-placement et routage.

Ces résultats de l'implémentation ont été obtenus à l'aide de l'analyseur logique TLA715 de Tektronix en le connectant à la sonde P6470 (TTL/CMOS) de Sony/Tektronix. L'analyseur logique permet de générer les signaux externes à travers le « Pattern generator » qui sont : l'horloge (*Clk*), l'initialisation (*Reset*), le signal de démarrage (*Start*) et enfin le signal des données (*Data_in*). Ces signaux sont appliqués à au FPGA qui, suite au traitement des données, génère les signaux attendus. Ces derniers sont récupérés pour être affichés par l'analyseur logique tel que montré à la Figure 5.8.

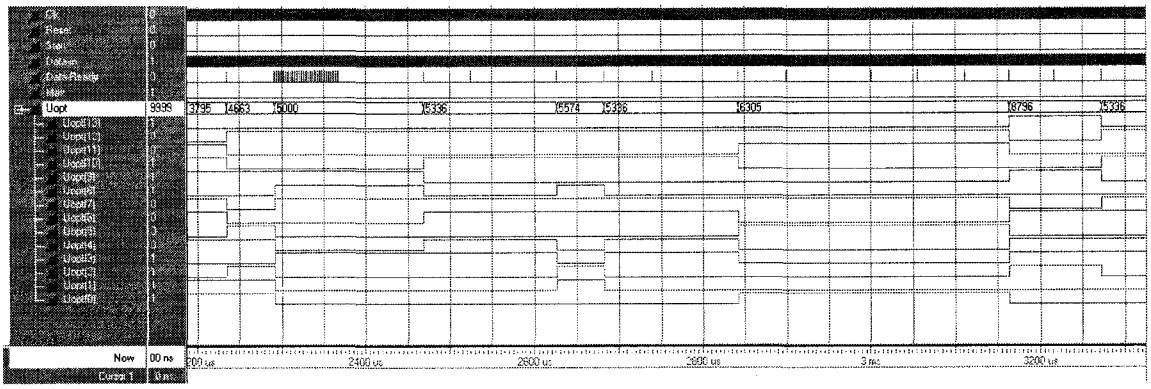


Figure 5.7. Simulation post placement et routage de l'architecture développée

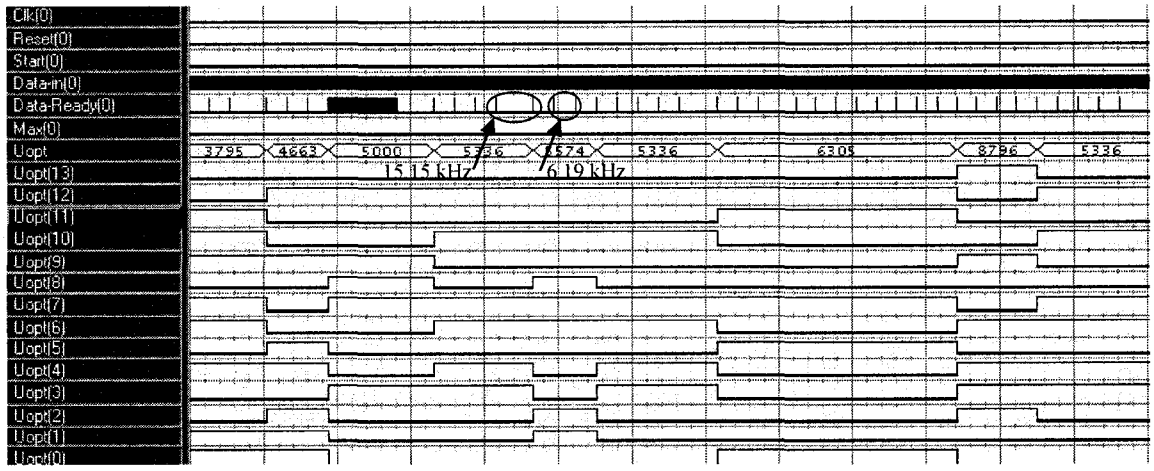


Figure 5.8. Résultats expérimentaux du FPGA AFS600 (Actel)

5.4.2. Puissance consommée

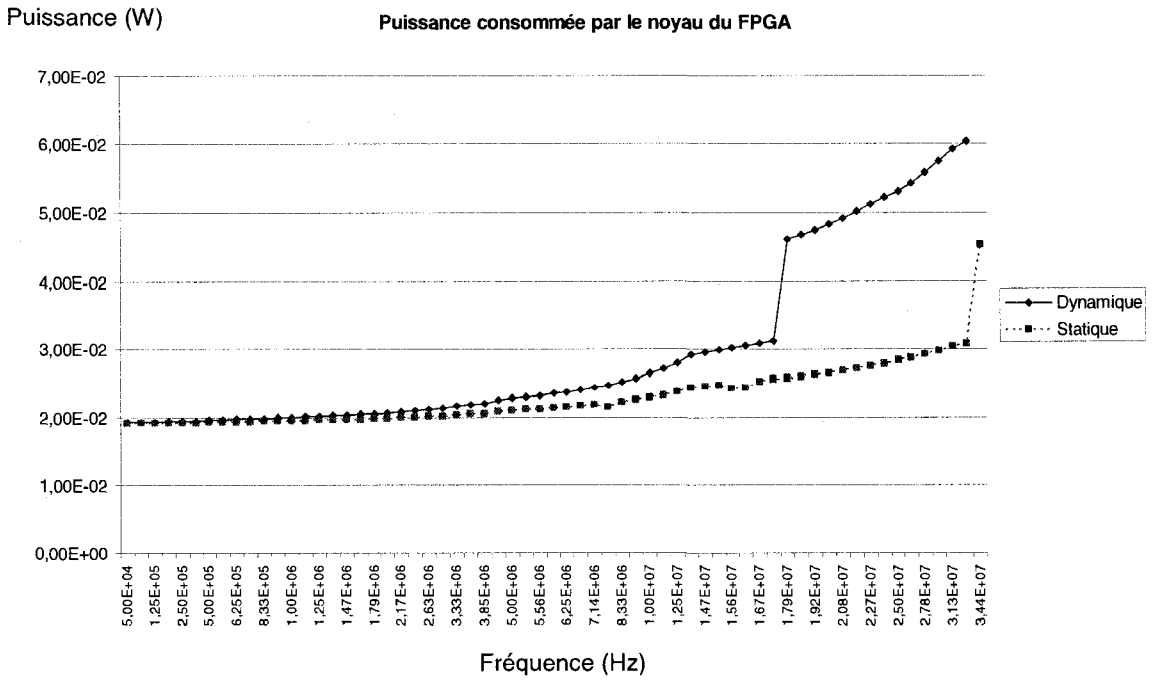
Les courbes de variations de la puissance consommée, qui sont présentées à la Figure 5.9a dépendent de la fréquence maximale de fonctionnement. Nous notons à la Figure 5.9a une augmentation de la consommation (gap) du FPGA qui se produit à 32.2 MHz dans le cas de l'architecture statique et à 17.2 MHz pour le décodage dynamique. Cette fréquence de l'architecture statique est plus élevée mais elle est, d'une part, compensée par la diminution de la durée du décodage. D'autre part, ceci n'est qu'une limitation matérielle qui peut ne pas avoir lieu sur une autre plateforme de développement.

Les gaps présents dans les courbes indiquent le passage du FPGA d'un mode de fonctionnement à un autre. Selon les informations fournies par Actel ceci peut être causé par une limitation de la puissance dynamique du FPGA qui correspond à la puissance consommée par le FPGA lorsque ce dernier fonctionne. Malgré la présence de ces gaps le FPGA continue à fonctionner correctement dans tous les cas au-delà des fréquences correspondantes au gap. L'une des explications plausibles est que le FPGA continue à fonctionner correctement sans garantir un résultat correct.

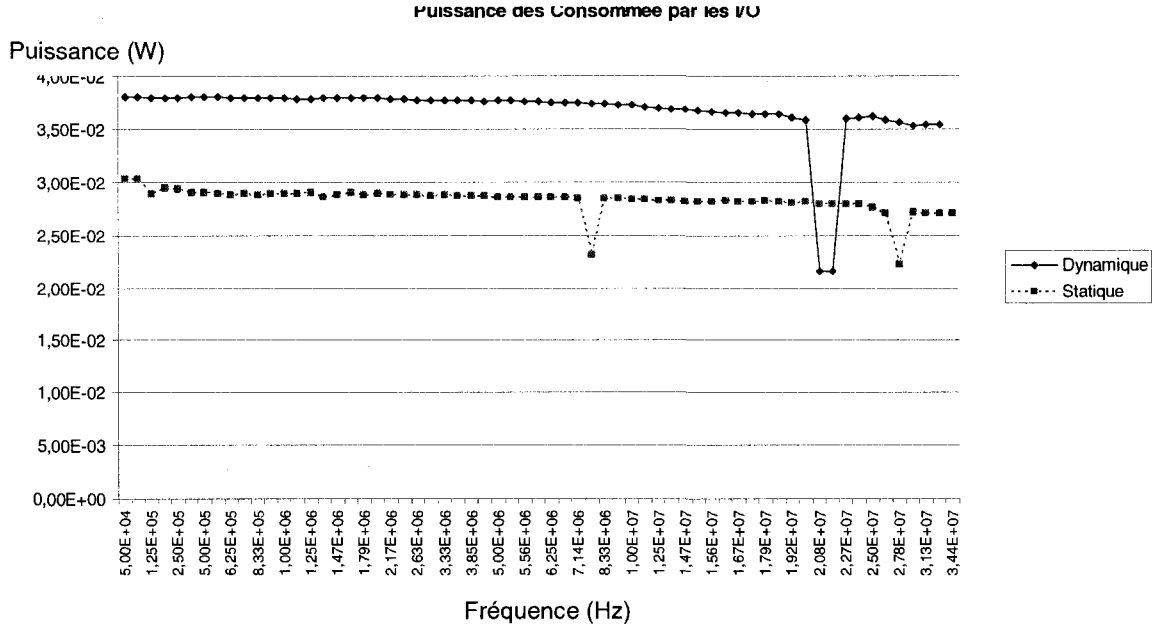
De plus, les changements brusques de consommation restent en dessous de celle estimée. Cette estimation a été réalisée par l'outil « Smart Power » d'Actel qui indique une estimation de la puissance consommée de 71.00 mW (Tableau 5.3). En combinant les diverses observations précédentes, nous concluons que le FPGA fonctionne correctement au-delà de 17.20 MHz ce qui est validé dans les différentes étapes de mise en œuvre.

À la Figure 5.9b montrant la puissance consommée par les entrée/sorties, nous remarquons que la consommation de puissance reste globalement constante sauf pour quelques fréquences ou elle accuse des diminutions. Ces cas particuliers correspondent à la fréquence 20.80 MHz et 21.70 MHz. C'est pour cette raison que nous avons constaté que le fonctionnement du FPGA n'est pas peut être garanti au-delà de la fréquence 17.20 MHz.

Il est à noter que le résultat représentant la puissance consommée par le noyau du FPGA correspond à la puissance consommée par l'architecture développée (Statique et dynamique) n'incluant pas la consommation des entrées/sorties du FPGA (figure. 5.10a).



(a)



(b)

Figure 5.9. Résultats expérimentaux :(a) Puissance consommée par le FPGA, (b) Puissance consommée par les entrées/sorties du FPGA

Tableau 5.4. Changement brusque de la consommation d'énergie

	Architecture de décodage statique de 40 bits	Architecture de décodage dynamique de 40 bits
Fréquence	32.20 MHz	17.20 MHz
Puissance consommée par le noyau	30.8 mW	31.3 mW

5.4.3. Fréquence maximale de fonctionnement de l'architecture proposée

Comme nous pouvons le noter dans le Tableau 5.5 la fréquence de fonctionnement mesurée du circuit est inférieure à celle estimée par SmarPower. Cependant, dans le cas de l'architecture dynamique, la fréquence reste sensiblement la même de celle estimée. La fréquence maximale de fonctionnement est obtenue avant que le fonctionnement du FPGA ne soit perturbé.

Tableau 5.5. Fréquence maximale de fonctionnement mesurée du FPGA

	Architecture de décodage statique de 40 bits	Architecture de décodage dynamique de 40 bits
Fréquence	34.4 MHz	32.2 MHz

En revenant aux explications du paragraphe 5.4.2 nous pouvons confirmer nos observations et conclusions puisqu'en effet le FPGA continue à fonctionner correctement jusqu'à la fréquence 32.2 MHz dans le cas de l'architecture dynamique.

La fréquence maximale de fonctionnement dans le cas d'une architecture dynamique est relativement élevée par rapport à la fréquence à laquelle le GAP se produit, soit à 17.2 MHz. Ceci peut s'expliquer par la présence des éléments mémoires dans le décodeur dynamique ce qui est plus exigeant d'un point de vue puissance.

5.5. Résumé des principales caractéristiques de l'architecture

Nous présentons dans le Tableau 5.6 une comparaison entre les 2 décodeurs présentés dans ce mémoire. Il est à rappeler que l'architecture dynamique du décodeur occupe plus d'espace que l'architecture statique incluant les mémoires, ce qui a pour impact une augmentation de la puissance consommée. Cependant plus de détails se trouvent dans l'annexe C

Tableau 5.6. Comparaison entre les deux décodeurs dynamique et statique

	Architecture de décodage statique de 40 bits	**Architecture de décodage dynamique de 40 bits**	**Total**
Fréquence maximale de fonctionnement (estimée par SmartTime d'Actel)	43.31 MHz	37.50 MHz	N/A*
Fréquence maximale de fonctionnement (mesurée)	34.40 MHz	32.20 MHz	N/A*
Puissance maximale (estimée par SmartPower d'Actel)	32.17 mW	71.10 mW	N/A*
Puissance maximale mesurée	57.80 mW	95.80 mW	N/A*
Unités logiques utilisées	6929 (50.12%)	11949 (86.44%)	13824
Routage global utilisé	6 (100.00%)	6 (100.00%)	6
RAM	2 (8.33%)	19 (79.17%)	24
I/O	21 (17.65%)	21 (17.65%)	119

* Non applicable

Cette augmentation est compensée par une augmentation de la vitesse de traitement de 2 à 4 fois dépendamment de la séquence à décoder comme montré dans la Figure 5.8. En effet cette figure montre que le décodeur dynamique augmente la vitesse de décodage de 6.19 kHz à 15.15 kHz dans ce cas de figure, soit une amélioration de 2.5 fois. Enfin, une simulation à l'aide de Matlab a permis d'illustrer la variation du gain en fonction de la longueur de la séquence. Ce résultat est reporté à la Figure 5.11. Pour cette fin, un échantillon de séquences (pour chaque longueur de séquence) a été envoyé au décodeur dynamique et statique. Une comparaison entre le nombre d'itérations

nécessaire pour aboutir au résultat final a été effectuée. Comme le montre la Figure 5.11, le décodeur dynamique est plus performant que le décodeur statique avec un meilleur résultat (niveau pic) pour les séquences de longueur 8 bits où le gain moyen est de 4.01 dB (5.1). Ce gain se stabilise à 1.7 dB pour les séquences de longueur 80 bits et plus [35].

$$gain_{séquence} = \left| 10.\log\left(\frac{Nombre\ d'itérations_{Décodeur\ dynamique}}{Nombre\ d'itérations_{Décodeur\ statique}} \right) \right| \qquad (5.1)$$

Le gain moyen est la moyenne des gains de toutes les séquences entrées pour une longueur de séquence donnée.

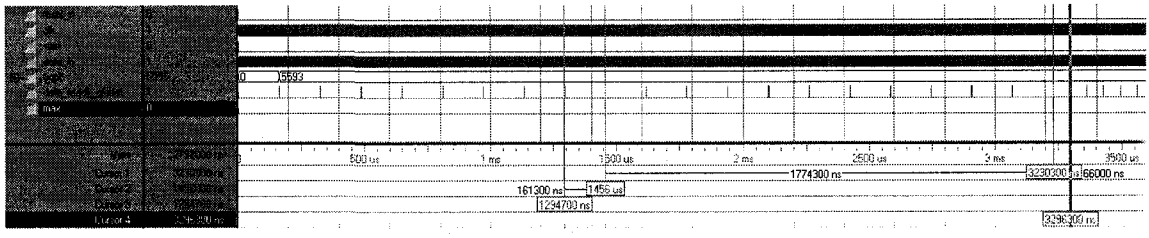


Figure 5.10. Fonctionnement du décodeur dynamique de 40 bits

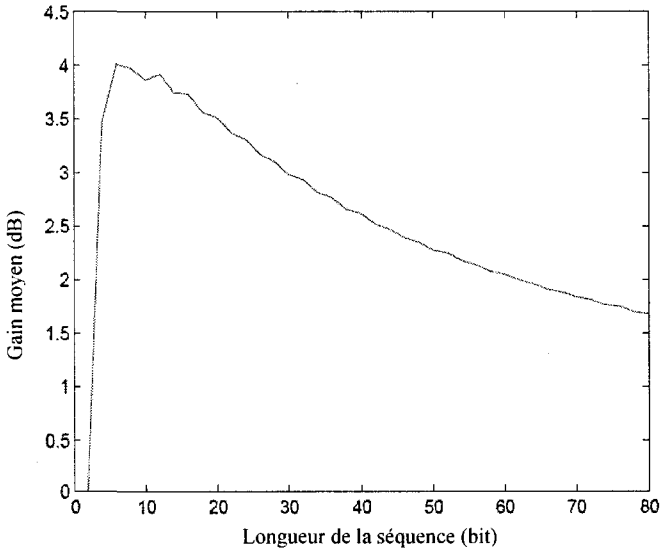


Figure 5.11. Performances de l'Architecture de décodage dynamique

5.6. Conclusion

Tout au long de ce chapitre nous avons présenté les résultats de l'implémentation matérielle du décodeur proposé. La durée de décodage a été considérablement améliorée par rapport à une technique de décodage statique. En contre partie une augmentation de l'espace occupé est l'inconvénient principal d'une telle architecture avec l'augmentation de puissance engendrée par cette dernière. Cependant et étant donnée que l'implémentation dépend considérablement de la plateforme utilisée, la puissance consommée dépend du circuit sur lequel l'architecture est implémenté. Nous notons que les comparaisons faites dans ce chapitre portent sur l'aspect dynamique vs statique appliqué à l'algorithme « Rational cyclic decoding ». Ceci dit, ces dernières peuvent s'appliquer pour n'importe quel algorithme itératif expliqué au chapitre 2.

Conclusion

De nos jours l'approche de conception dynamique des décodeurs est de plus en plus utilisée. Son principal avantage est de donner aux différents systèmes plus d'autonomie et de réduire le temps de traitement des données. Les applications pour de tels systèmes sont très diversifiées : allant de l'aérospatiale au biomédical. Les laboratoires sur puces représentent l'un des domaines biomédicaux en question. Tel est le cas pour les capteurs capacitifs qui sont utilisés pour la détection et la manipulation des cellules et des substances microfluidiques. Étant donné que de tels systèmes fonctionnent souvent dans des milieux différents et dont nous ne connaissons pas les principales caractéristiques, nous devons être capables d'adapter leur comportement à chaque situation. Cependant, avoir une architecture statique est un handicap pour la versatilité et la flexibilité du système, d'où l'utilité d'avoir une architecture dynamique de sorte que le traitement des données puisse être accéléré dépendamment du besoin. Une telle approche permet une augmentation des performances du système et ceci en augmentant sa capacité de détection. Pour cette fin, une architecture dynamique s'avère très utile.

Dans ce mémoire, nous avons proposé une nouvelle technique de décodage qui permet d'accélérer le temps de traitement de données de décodeurs dédiés aux modulateurs ΣΔ. Cette dernière a été inspirée de la technique de gestion du flux de données sur Internet. Elle est basée sur un algorithme de décodage statique cependant elle peut être adaptée à n'importe quel algorithme de décodage pour les CAN ΣΔ pourvu que ce dernier soit itératif. L'approche dynamique a permis une amélioration du temps de traitement des données en accélérant la vitesse de décodage de 2 à 4 fois et ceci dépendamment de la séquence fournie par le modulateur ΣΔ, ce qui est crucial pour augmenter la capacité de détection des substances fluidiques. Cette dernière a été implémentée sur un FPGA mixte analogique et numérique (AFS600) qui est utilisé

également pour contrôler les autres fonctions du capteur capacitif telles que la calibration, la surveillance et le contrôle de la puce analogique. Les résultats de cette implémentation se résument en un espace occupé de 86.44%, une fréquence de fonctionnement de 32.20 MHz et une puissance consommée de 71.10 mW.

Quant aux recommandations pour améliorer les performances de l'architecture proposée nous proposons de:

- Optimiser l'architecture pour aboutir à une réduction de l'espace et par conséquence de la consommation de puissance de la puce électronique. Comme exemple d'optimisation possible, nous proposons de développer une architecture plus rapide pour la division et une compression des mémoires utilisées en les rendant plus petites.

- Appliquer la méthode dynamique à divers signaux. En effet, l'architecture dynamique a été appliquée pour un signal constant, il serait utile d'appliquer cette même technique pour des signaux périodiques pour n'avoir à décoder qu'une période et non tout le signal.

- Rendre le temps d'apprentissage reconfigurable. Le temps d'apprentissage peut être configuré selon les besoins : soit par l'usager ou une autre interface, soit codé à l'intérieur de l'architecture ou bien préprogrammé et peut être changé dépendamment de l'application.

Bibliographie

[1] E. Ghafar-Zadeh and M. Sawan, "A high precision and linearity differential capacitive sensor circuit dedicated to bioparticule detection," CCECE, pp 1613-1616, May 2006.

[2] E. Ghafar-Zadeh, M. Sawan, M. Hajj-Hassan and M.A. Miled "A CMOS based microfluidic detector: design, calibration and experimental results," MWSCAS/NEWCAS 2007, pp 193-196, August 2007.

[3] I. Evans and T. York, "Microeletcronic capacitance transducer for particle detection," IEEE Sensors journal, Vol. 4, No. 3, June 2004.

[4] N. Manaresi et al., " A CMOS chip for individual cell manipulation and detection," IEEE journal of Solid-State circuits, Vol.38, No 12 December 2003 pp 2297-2305.

[5] A. Romani et al., "Capacitive sensor array for localization of bioparticles in CMOS Lab-on-a-chip," ISSCC 2004, Session 12, Biomicrosystems 12.4.

[6] V. Srinivasan et al., "Clinical diagnostics on human whole blood, plasma, serum, urine, saliva, sweat, and tears on digital microfluidic platform," μTAS 2003 conf, October 5-9, 2003.

[7] B. H. Weigl, "Lab-on-a-chip for drug development," journal Elsevier, Advanced Drug Delivery Reviews, Volume 55, Issue 3, 24 February 2003, Pages 349-377.

[8] Zhang et al., "Dynamic Iterative decoding for balancing quality of service parameters," United States Patent, Patent No: US 6,233,709 B1, date of patent May 15, 2001.

[9] J.A. Schwartz et al., "Dropled-based chemistry on a programmable micro-chip," Lab chip 2004, RCS Publishing, Lab chip 2004, 4, pp11-17.

[10] Jean-Paul Troadec, "Principes de conversion analogique-numérique et numérique-analogique," Edition Dunod, Paris 2004.

[11] F. Bailleux et al., "Capacités commutées & applications," Édition Dunod, Paris 1996.

[12] Paul Horowitz and Winfield Hill, "Traité de l'électronique analogique et numérique," Édition Publitronic/Elektor, juin 1996.

[13] Georges Asch et al., "Acquisition de données, du capteur à l'ordinateur," Édition Dunod, Paris 1999.

[14] Price et al., "Successive Approximation Analog-to-digital converters," United States Patent, Patent No: US 6351231 B1, date of patent: December 23, 1999.

[15] J. Park et al., "A 1mW 10-bit 500KSPS *SAR* A/D Converter," ISCAS 2000, May 28-31, 2000.

[16] J. Gan et al., "Effects of noise and nonlinearity on the calibration of a non-Binary capacitor array in a successive approximation analog-to-digital converter," Asia and South Pacific Design Automation Conference 2004, 27-30 Jan. 2004, pp. 292-297.

[17] P.Arpaia et al., "Compensation of intrinsic Nonlinearity of SAR ADCs," IEEE instrumentation and measurement technology, USA, 21-23 May 2002.

[18] T.P. Kearney., "Programmable input range SAR ADC," United States Patent, Patent No: US 6731232 B1, date of patent: May 4, 2002.

[19] B. E. Boser and B. A. Wooley, "The Design of Sigma-Delta modulation Analog-to- Digital converters," IEEE journal of solid state-state circuits, Vol. 23, No. 6, December 1988.

[20] P. M. Aziz et al., "An overview of Sigma-Delta converters: How a I-bit ADC achieves more than 16-bit resolution," IEEE Signal processing magazine, January 1996.

[21] Richard Schreier, "An Empirical Study of High-Order Single-Bit Delta-Sigma Modulators," IEEE Trans. on circuits and systems II: Analog and digital signal processing, Vol. 40, No. 8, August 1993.

[22] N. T. Thao and M. Vetterli., "Vector quantization view of ΣΔ modulation," 1994 International symposium on speech, image processing and neural network, 13-16, April 1994, Hing Kong.

[23] S. A. Jantzi et al., "A Fourth-Order Bandpass Sigma-Delta Modulator," IEEE journal of solid state-state circuits, Vol. 28, No. 3, December 1993.

[24] S. H. Lewis and P. R. Gray, "A Pipelined 5-Msample/s 9-bit Analog-to-Digital converter," IEEE journal of solid state-state circuits, Vol. SC-22, No. 6, December 1987.

[25] S. Sutarja and P. R. GRAY, "A Pipelined 13-bit, 250-ks/s, 5-V Analog- to- Digit al Converter," IEEE journal of solid state-state circuits, Vol. 23, No. 6, December 1988.

[26] W. T. Colleran and A. A. Abidi, "A 10-b, 75-MHz two-stage pipelined bipolar AD converter," IEEE journal of solid state-state circuits, Vol. 25, No. 12, December 1993.

[27] B. Black, "Analog-to-Digital Converter Architectures and Choices for System Design," Analog device Analog dialogue Vol. 33, Number 8, September, 1999

[28] R.M. Gray, "Oversampled sigma-delta modulation," IEEE Trans. Commun., vol. COM-35, pp. 481-489, May 1987.

[29] Soren Hein, Khalid Ibraham and Avideh Zakhor, "New properties of sigma-delta modulators with dc inputs," IEEE Trans. Commun., vol. 40, pp. 1375-1387, August 1992

[30] R.M. Gray, "Quantization Noise Spectra," IEEE Trans. Information theory., vol. 36, pp. 1220-1244, May 1990.

[31] James C. Candy, "Decimation for sigma delta modulator," IEEE Trans. Commun., vol. COM-34, NO. 1, pp. 72-7, January 1986.

[32] Soren Hein and Avideh Zakhor, "Optimal decoding for data acquisition applications pf sigma delta modulator," IEEE Trans. Signal processing., vol. 41 N°2, pp. 602-616, May 1993.

[33] Lisa G. McIlrath, "A robust O(N logN) Algorithm for optimal decoding of first-order sigma delta sequences," IEEE Trans. On signal processing, vol. 50, NO. 6, June 2002.

[34] Frank Dachselt, Stefan Quitzk and ingo Wiemer, "First-order Sigma-delta Modulator: Hierarchical generation model and optimal decoding," Dresden university of technology, May 2004.

[35] M.A. Miled, E. Ghafar-Zadeh and M. Sawan, "Fast decoding algorithm for first order DC-input sigma-delta modulator," MWSCAS2007, pp 1380-1383, August 2007.

[36] M. Götz and W. Schwartz, "Correlation theory of a class of n-dimensional piecwise linear marcov system," in Proc. ISCAS, vol. 2, pp. 1049-1052, 1997.

[37] M. Götz, W. Schwartz and A. Abel, "Evolution operator based analysis of the non-overloaded ΣΔ-modulator – case study," in Proc. NDES, 2001.

[38] Thomas Wiegand, Gary J.Sullivan, Gusle Bjontegaard and Ajay Luthra, "Overview of the H.264/AVC Video coding standard," IEEE transactions on circuits and systems for video technology, vol 13, No 7, July 2003.

Annexe A

Algorithme de division binaire

Initialiser le quotient à 0

Aligner le numérateur et le dénominateur de sorte que le bit le plus significatif du dénominateur se trouve sous le bit le plus significatif du numérateur

Répéter

Si (la partie du dividende située au dessus du diviseur est supérieure ou égale au dénominateur)

Alors

Soustraire le diviseur de la partie du diviseur aligner avec ce dernier.
Concaténer la partie restante du dividende avec le résultat de la soustraction
Concaténer 1 à l'extrémité droite du quotient.

Sinon

Concaténer 0 à l'extrémité droite du quotient

Décaler le dividende de 1 bit à droite

Jusqu'à (dividende inférieur au diviseur)

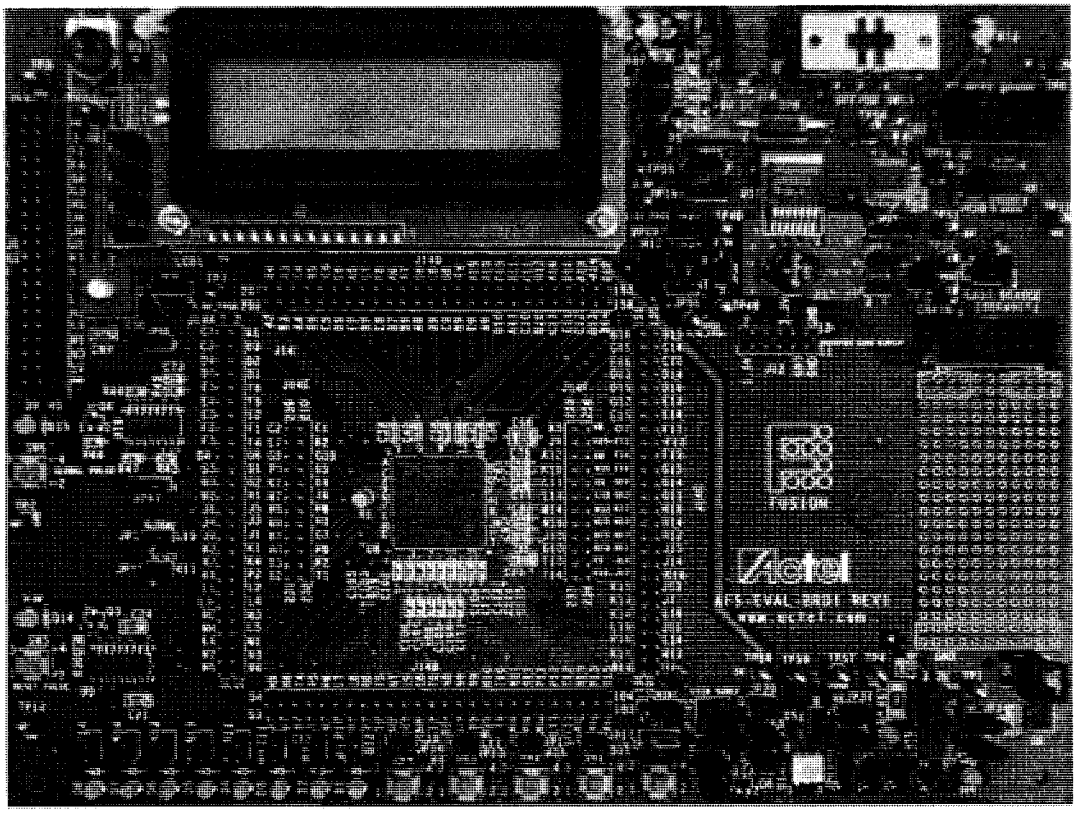
Stop

Exemple de division binaire : dividende = 10110, diviseur=11

Étape	Opération		Quotient
0 (Initialisation)	**10**110- 11	10110>11 10-11=11	0
2	**101**10- 11	10110>11 101-11=010	00
3	**0101**0- 11	1010>11 0101-11=0010	001
4	**00100**- 11	100>11 00100-11=0001	0011
5	00001- 11	00001<11	00111
6 Stop	Reste=**00001**		**Quotient=00111**

Annexe B

Carte de développement AFS-EVAL-BRD1



Annexe C

Ressources matérielles utilisées

C.1. Architecture de décodage dynamique

Compile report:

==========

CORE	Used:	11949	Total:	13824	(86.44%)
IO (W/ clocks)	Used:	21	Total:	119	(17.65%)
Differential IO	Used:	0	Total:	58	(0.00%)
GLOBAL (Chip+Quadrant)	Used:	6	Total:	18	(33.33%)
PLL	Used:	0	Total:	2	(0.00%)
RAM/FIFO	Used:	19	Total:	24	(79.17%)
Low Static ICC	Used:	0	Total:	1	(0.00%)
FlashROM	Used:	0	Total:	1	(0.00%)
User JTAG	Used:	0	Total:	1	(0.00%)
RC oscillator	Used:	0	Total:	1	(0.00%)
XTL oscillator	Used:	0	Total:	1	(0.00%)
NVM	Used:	0	Total:	2	(0.00%)
AB	Used:	0	Total:	1	(0.00%)
AnalogIO	Used:	0	Total:	46	(0.00%)
VRPSM	Used:	0	Total:	1	(0.00%)
No-Glitch MUX	Used:	0	Total:	2	(0.00%)

Global Information:

Type	Used	Total
Chip global	6	6 (100.00%)
Quadrant global	0	12 (0.00%)

Core Information:

Type	Instances	Core tiles
COMB	7879	7879
SEQ	2907	4070

I/O Function:

Type	w/o register	w/ register	w/ DDR register
Input I/O	4	0	0
Output I/O	17	0	0
Bidirectional I/O	0	0	0
Differential Input I/O Pairs	0	0	0
Differential Output I/O Pairs	0	0	0

I/O Technology:

	Voltages		I/Os		
I/O Standard(s)	Vcci	Vref	Input	Output	Bidirectional
LVTTL	3.30v	N/A	4	17	0

C.2. Architecture de décodage statique

Compile report:

CORE	Used:	6929	Total:	13824	(50.12%)
IO (W/ clocks)	Used:	19	Total:	119	(15.97%)
Differential IO	Used:	0	Total:	58	(0.00%)
GLOBAL (Chip+Quadrant)	Used:	6	Total:	18	(33.33%)
PLL	Used:	0	Total:	2	(0.00%)
RAM/FIFO	Used:	2	Total:	24	(8.33%)
Low Static ICC	Used:	0	Total:	1	(0.00%)
FlashROM	Used:	0	Total:	1	(0.00%)
User JTAG	Used:	0	Total:	1	(0.00%)
RC oscillator	Used:	0	Total:	1	(0.00%)
XTL oscillator	Used:	0	Total:	1	(0.00%)
NVM	Used:	0	Total:	2	(0.00%)
AB	Used:	0	Total:	1	(0.00%)
AnalogIO	Used:	0	Total:	46	(0.00%)
VRPSM	Used:	0	Total:	1	(0.00%)
No-Glitch MUX	Used:	0	Total:	2	(0.00%)

Global Information:

Type	Used	Total
Chip global	6	6 (100.00%)
Quadrant global	0	12 (0.00%)

Core Information:

Type	Instances	Core tiles
COMB	4654	4654
SEQ	1354	2275

I/O Function:

Type	w/o register	w/ register	w/ DDR register
Input I/O	4	0	0
Output I/O	15	0	0
Bidirectional I/O	0	0	0
Differential Input I/O Pairs	0	0	0
Differential Output I/O Pairs	0	0	0

I/O Technology:

	Voltages		I/Os		
I/O Standard(s)	Vcci	Vref	Input	Output	Bidirectional
LVTTL	3.30v	N/A	4	15	0

Annexe D

Puissance consommée par l'architecture proposée

Le fichier Puissance.xls regroupe l'ensemble des valeurs expérimentales de la puissance consommée par l'Architecture proposée pour différentes fréquences. La plage de fréquence varie de 50 kHz jusqu'à la fréquence maximale de fonctionnement du FPGA. Le tableau contenu dans le fichier Puissance.xls renferme des mesures de l'architecture dynamique ainsi que statique.

Annexe E

Algorithme de décodage dynamique

L'algorithme de décodage dynamique a été implémenté sur Matlab. Les fichiers suivants reconstituent l'algorithme de décodage de Dachselt et al. [34] utilisé :

RCS.m : Détermination des $E_{courant}$ et des $E_{correctif}$.

CNIS_NCR.m : Créer une Nouvelle Séquence Intermédiaire incluant les NCR.

SNCR.m : Détermination des NCR.

FNS.m : Générer la séquence finale.

CNC.m : Générer les différent cœfficients $w_{i,r}$, $w_{i,c}$, $l_{i,r}$, et $l_{i,c}$

C3U.m : Calculer les estimations de la séquence précédente, courante et suivante $u_{pred,i}$, $u_{curr,i}$, et $u_{succ,i}$.

Uest.m : Calcul de la valeur moyenne de $u_{pred,i}$, $u_{curr,i}$, et $u_{succ,i}$. qui n'est autre que $u_{est,i}$.

Les fichiers suivant constituent la partie dynamique de l'algorithme développé :

Check.m : Vérifie si la valeur recherchée existe dans les lignes de mémoire autre que la 1ère ligne.

Check1.m : Vérifie si la valeur recherchée existe dans a 1ère ligne ou non.

Checking.m : Permet de gérer les instructions contenues dans les fichiers Check.m et Check1.m dans le but de parcourir tout l'espace mémoire disponible.

Backup.m : Sauvegarde les valeurs des différents niveaux pour une séquence donnée. Dans le cas où la séquence n'a pas été précédemment décodée l'algorithme extrait les valeurs du fichier de backup pour les enregistrer dans la mémoire.

Memory.m : Crée une image de la mémoire sous forme de matrice ainsi que le masque correspondant à cette dernière.

Intelligence.m : Permet de gérer le flux des données en appliquant le principe de recherche dynamique tout en utilisant les fichiers Check.m et Check1.m.

Le fichier suivant assure l'interfaçage entre l'algorithme de Dachselt et al. [34] et l'algorithme de décodage dynamique :

Decoding.m : Assure l'interfaçage entre les deux parties en incluant les conditions d'arrêts pour l'algorithme de Dachselt.

Annexe F

Architecture de décodage dynamique

Fichiers VHDL

Nom du fichier	*: description*
FSM_init.vhd	: Initialisation de toutes les mémoires et les registres de l'architecture.
Preload.vhd	: Sauvegarde la séquence entière en un registre en attendant qu'elle soit chargée dans la perspective de son décodage.
FSM_Load.vhd	: Charge la séquence provenant du modulateur ΣΔ dans la mémoire du décodeur.
FSM_Stat.vhd	: Contrôle toutes les machines à états qui gèrent la partie statique du décodeur.
FSM_RCS.vhd	: Contrôle la recherche de l'$E_{courant}$ et de l'$E_{correcrif}$ dans la séquence chargée.
RCS.vhd	: Renferme les registres contenant l'$E_{courant}$ et l'$E_{correcrif}$ ainsi qu'une logique permettant de les retrouver qui est contrôlée par la machine à états FSM_RCS contenue dans le fichier FSM_RCS.vhd.
RCS_Scan.vhd	: Connecte RCS avec FSM_RCS.
FSM_CNIS_NCR.vhd	: Contient la machine à état permettant de générer la séquence intermédiaire contenant les NCR et l'enregistre dans le bloc mémoire 2.
FSM_SNCR.vhd	: Contient une machine à état permettant d'indiquer si des NCR existent ou non.

FSM_FSN.vhd : Contient une machine à état qui génère la séquence finale en tenant compte du résultat de FSM_SNCR.vhd.

FSM_C3U.vhd : renferme une MEF qui permet de contrôler le module logique U_{est} pour générer la valeur restaurée de la séquence courante ainsi que son prédécesseur et son successeur.

C3U_element.vhd : Renferme le module permettant de restaurer la valeur de la séquence entrée.

C3U.vhd : Connecte FSM_C3U avec 3 composant de C3U_elements. Chaque composant correspond soit au précédente soit au successeur soit à la séquence courante.

CNC.vhd : Permet de calculer les cœfficients du niveau courant.

Pre_Uest.vhd : Sauvegarde la valeur estimée du niveau précédent et fait la somme du prédécesseur, successeur et la valeur courante.

FSM_Uest.vhd : Contrôle l'architecture Pre_Uest.

Uest.vhd : Connecte Pre_Uest et FSM_Uest ensemble.

Accumulator.vhd : Additionne les différentes valeurs contenues dans une séquence donnée.

Division.vhd : Effectue une division binaire

FSM_Uopt.vhd : Contrôle le module de division pour faire la division par 3 du prédécesseur, successeur et la valeur courante.

Period.vhd : Indique si la période de la séquence courante est 2 ou non.

Static_Comp.vhd : Rassemble tous les composants du module statique sauf les MEF : mémoires, composants logiques...

FSM_Stat.vhd : Rassemble toutes les MEF du module statique en un seul module globale.

Static_Modul.vhd : Connecte le module des MEF avec les composants logiques.

FSM_Learning.vhd : Assure le contrôle des modules et mémoires qui sont utilisés durant la phase d'apprentissage.

FSM_Learning_ctrl.vhd : Active les machines à états de la phase d'apprentissage.

Lear_Comp.vhd : Renferme les composants logiques de la phase d'apprentissage.

FSM_Read.vhd : Contrôle tous les composants de la partie dynamique lorsque la phase de lecture est activée.

FSM_Write.vhd : Contrôle tous les composants de la partie dynamique lorsque la phase d'écriture est activée.

Pos_Select.vhd : Étant donné que chaque cellule de la mémoire renferme trois valeurs différentes, le rôle de ce composant est de sélectionner la bonne valeur dépendamment de l'état du décodeur et de l'envoyer à ce dernier.

Comparator.vhd : Ce comparateur spécialement développé pour ce décodeur compare les trois valeurs d'une cellule avec une valeur de référence et renvoie un signal sur 3 bits indiquant le résultat de la comparaison. Chaque bit correspond à une valeur de la cellule mémoire (chaque cellule renferme trois valeurs différentes).

Adder.vhd : Le rôle de cet additionneur est de compter le nombre de valeurs susceptibles de représenter la séquence fournie par le modulateur $\Sigma\Delta$ dans toute la zone de recherche.

Memory_bloc.vhd : Connecte tous les sous blocs de la mémoire principale pour former la zone de recherche. En effet la mémoire principale est formée par 5 sous blocs. Chacun correspond à un niveau de décodage.

Depth.vhd : Renferme des indices. Chaque indice correspond aux nombres de valeurs contenues dans chaque colonne. Ce dernier correspond au bloc mémoire *indexe*.

Reg_Memory.vhd : Pour accélérer le traitement les données de chaque ligne sont chargées dans un registre. Ceci permet de faire tout le traitement en un seul coup d'horloge au lieu de lire la mémoire cellule par cellule.

Shift_Reg.vhd : Registre à décalage utilisé pour charger les données de la mémoire.

FSM_Dyn.vhd : Regroupe toutes les machines à états utilisées par le module dynamique.

Dyn_Comp.vhd : Contient tous les composants utilisés par le module dynamique.

Dyn_Modul.vhd : Connecte le module des MEF avec les composants logiques.

FSM_Exception.vhd : Contrôle le traitement des données lorsque la séquence fournie par le modulateur $\Sigma\Delta$ n'est formée que par des 1 ou que des 0 ou bien 1 et 0 et de période 2.

Global.vhd : Connecte le module statique avec le module dynamique.

FSM_Global.vhd : Chapote le fonctionnement des machines à états du module statique et dynamique.