الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne démocratique et populaire
وزارة التعليم العالي و البحث العلمي
Ministère de l'enseignement supérieur et de la recherche scientifique
جامعة سعد دحلب البليدة
Université SAAD DAHLAB de BLIDA
كلية التكنولوجيا
Faculté de Technologie
قسم الإلكترونيك
Département d'Électronique

# Mémoire de Master

Mention Électronique
Spécialité Électronique des systèmes embarqués

présenté par

TAIHI Fatiha

&

BENMAIZA Samia

# Etude et implémentation d'un filtre RIF adaptatif sous VHDL

Proposé par : **Pr DJENDI Mohamed**

Année Universitaire 2017-2018

# Remerciements

Avant tout, l'éloge à Dieu tout puissant pour tout ce qu'il nous a donné et nous avoir accordé la force, le courage et les moyens afin de pouvoir accomplir ce modeste travail, **« Dieu merci ».**

En premier lieu, nous souhaitons vivement remercier notre promoteur **Mr DJENDI Mohamed** pour la précieuse assistance, sa disponibilité et son soutien qu'il nous accordé tout au long de ce projet.

Notre gratitude va aussi à nos **Enseignants de Département d'électronique** pour tout le savoir qu'ils ont su nous transmettre durant ces deux dernières années.

Nous tenons à remercier **Mr le président** et **les membres du jury**, pour avoir accepté de participer en tant qu'examinateurs à notre soutenance**.**

Enfin, un grand merci à nos parents, nos collègues et tous ceux qui ont contribué de près ou de loin à la réalisation du présent travail.

**ملخص:**

يتحدث مشروع نهاية دراستنا على مجال الغاء الصدى الصوتي بواسطة الترشيح المتكيف. لقد درسنا خوارزمية جديدة للتصفية التكييفية (المربعات القياسية الصغرى المحسنة) مخصصة لالغاء الصدى الصوتي ثم قمنا بتنفيذها في VHDL.

النتائج التي تم الحصول عليها من هذا المشروع واعدة.

**كلمات المفاتيح:** الصدى الصوتي, الترشيح المتكيف, المربعات القياسية الصغرى المحسنة (NLMSA) .

**Résumé :**

Notre projet de fin d'étude concerne le domaine de l'annulation d'écho acoustique par des approches adaptatives. Nous avons étudié puis implémenter sous VHDL un nouvel algorithme (NLMSA) de filtrage adaptatif dédié pour l'annulation d'écho acoustique.

Les résultats obtenus de ce projet sont prometteurs.

**Mots clés :** écho acoustique ; filtrage adaptatif; moindre carré normalisées amélioré (NLMSA).

**Abstract :**

Our project is about acoustic echo cancellation using an adaptive filtering approach. We have studied and implemented an adaptive filtering algorithm for echo cancellation on VHDL.

This algorithm has shown promising results.

**Keywords:** acoustic echo, adaptive filtering, normalized least mean squares ameliorate (NLMSA)

## Listes des acronymes et abréviations

**AEA**: Annulation d'écho Acoustique.

**CPLD**: Complex programmable Logic Device.

$\boldsymbol{d(n)}$ : Réponse impulsionnelle désirée, scalaire.

**EQM**:  Erreur quadratique Moyenne.

**ERLE**: Echo Return Loss Enhancement.

$\boldsymbol{E(n)}$ : Vecteur d'erreur de Filtrage.

$\boldsymbol{E}[.]$ : Espérance Mathématique.

$\boldsymbol{e(n)}$ : L'erreur de sortie du Filtre.

**FPGA**: Field Programmable Gte Array.

$\boldsymbol{f_c}$ : Fréquence de coupure.

$\boldsymbol{h}$ : Réponse Impulsionnelle du milieu.

**LMS**: Least Mean Square.

**MSE**: Mean Square Error.

**NLMS**: Normalized Least Mean Square.

**NLMSA:** Normalized Least Mean Square Ameliorate.

$\boldsymbol{Pgn_{Fe}}$ : Peigne de Dirac (suite de pic de Dirac).

$\boldsymbol{R_{xx}}$ : Matrice d'autocorrélation.

$\boldsymbol{R_{xy}}$ : Matrice d'inter-corrélation.

**RIF** : Réponse Impulsionnelle Finie.

**RII** : Réponse Impulsionnelle Infinie.

**TFD** : Transformation de Fourier Discrète.

**TFTD** : Transformé de Fourier à Temps Décret.

**USASI**: United State Of American Standard Institute.

**VHDL**: Very High Speed Integrated Circuit.

$\boldsymbol{W}$: Les coefficients du Filtre Adaptatif.

$\boldsymbol{x(n)}$ : Signal d'entrée.

$\boldsymbol{y(n)}$ : Signal de sortie.

$\boldsymbol{\alpha}$ : Facteur de Lissage.

$\boldsymbol{\beta}$ : L'énergie de l'erreur.

$\boldsymbol{\delta(n)}$ : Impulsion Unité Discrète.

$\boldsymbol{\nabla J_k}$ : Le gradient.

$\boldsymbol{\mu}$ : Pas d'adaptation.

$\boldsymbol{\lambda}$ : Facteur d'oubli, scalaire.

$\boldsymbol{\sigma_x^2}$ : L'énergie du signal d'entrée $x(n)$.

$\boldsymbol{\delta}$ : Paramètre de régularisation.

$\boldsymbol{r}$ : Vecteur d'inter-corrélation.

$.^{\boldsymbol{T}}$ : Transposé d'une Matrice ou d'un Vecteur.

$.^{\boldsymbol{H}}$ : Conjugué d'un Vecteur, Matrice ou Nombre Complexe.

# Table des matières

# Liste des figures

# Liste des tableaux

# Introduction générale

La parole, moyen de communication privilégié entre les humains, constitue une grande partie des messages transmis en télécommunications. Un phénomène d'écho, qui est la réverbération du signal à l'émission, pose généralement un problème dans toutes les communications de type "PC à Téléphone" ou "Téléphone à Téléphone".

Certains nouveaux services de télécommunications ont considéré le milieu acoustique (salle, cabine téléphonique, habitacle d'une voiture etc...) comme faisant partie de la chaîne de communication. Ces nouveaux services correspondent à l'apparition des postes à haut-parleur, des postes mains-libres et des systèmes de téléconférences (audio et visioconférences).

Le phénomène d'écho acoustique : lors de la mise en place d'une communication bidirectionnelle entre deux salles, une boucle de transmission fermée est établie. Le signal émis par la salle distante est réémis vers cette même salle à cause du couplage existant entre le haut-parleur et le ou les microphones de prise de son d'une même salle. Si la transmission introduit un retard important (de l'ordre de plusieurs centaines de millisecondes), les personnes présentes dans une salle réentendent leur propre voix ; c'est le phénomène d'écho acoustique dû au canal acoustique de couplage qui, par définition, représente la transformation du signal diffusé par le haut-parleur et capté par les microphones de prise de son.

Lorsque l'écho acoustique est présent de façon gênante, c'est à dire clairement distinct subjectivement de son signal d'origine, un traitement spécifique, appelé "annulation d'écho acoustique", doit être impérativement mis en œuvre pour préserver la qualité de la communication. Le but d'un tel traitement est d'estimer l'écho acoustique entre

le signal reçu (signal envoyé dans le haut-parleur) et la sortie de la salle (signal capté par le microphone) puis de retrancher une estimation de ce signal de sortie, ceci sans affecter le signal de parole locale dans le cas de double parole (les deux locuteurs parlent en même temps). L'annulation d'écho acoustique est un problème d'identification d'un système linéaire (canal acoustique de couplage) excité par un signal de référence connu (parole alimentant le haut-parleur). Le problème est compliqué par le fait que le signal d'excitation est fortement non stationnaire et le canal acoustique de couplage varie au cours du temps (mouvements des personnes, déplacements d'objet, etc.…). Pour tenir compte de ces problèmes, nous utilisons un annulateur d'écho acoustique où l'identification de la réponse impulsionnelle finie (FIR : Finie Impulse Response), représentant le canal acoustique de couplage, est réalisée par des algorithmes du type gradient stochastique (LMS : Least Mean Squares, NLMS : Normalized LMS, NLMSA: NLMS amélioré, etc.…).

Notre travail a pour objectif d'implémenter l'algorithme NLMSA qui a été proposé par le professeur **Mohamed Djendi**, sous VHDL pour l'annulation d'écho acoustique par le filtrage adaptatif, pour cela nous présentons ci-après les différents chapitres de ce rapport :

**Chapitre 1** présente une généralité sur le filtre et le filtrage numérique.

**Chapitre 2** traite le problème d'écho acoustique et les algorithmes utilisés pour l'annulation d'écho acoustique par le filtrage adaptatif.

**Chapitre 3** présente les notions de base sur le langage VHDL et les logiciels de simulation et d'implémentation.

**Chapitre 4** présente les résultats de simulations des algorithmes d'annulation d'écho acoustique sous Matlab et sous VHDL.

# Chapitre 1 Filtre et filtrage numérique

## 1.1 Introduction

Ce chapitre permet de présenter les filtres numériques et leurs différents types, caractéristiques et structures. Pour cela nous allons étudier les filtres à réponse impulsionnelle finie et infinie ainsi que les structures de leurs réalisations.

## 1.2 Définition d'un filtre

C'est un élément nécessaire dans le traitement du signal, il est utilisé pour sélectionner les fréquences, c'est à dire Permet aux fréquences de passer et bloquer les autres fréquences. Il constitue la base d'une discipline d'ingénierie, le traitement du signal, qui s'applique à des signaux de tout type (sons, images, vidéo, vibrations sismiques, ...)

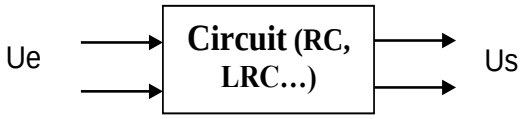


***Figure 1. 1*** Shéma d'un filtre.

## 1.3 Différents types du filtre

- **Filtre passe bas**

Le filtre passe bas laisse passer les fréquences qui sont au-dessous de la fréquence de coupure, et attenue les autres fréquences (voir figure 1.2).

- **Filtre passe haut**

Le filtre passe haut laisse passer les fréquences au-dessus de $f_C$ (voir figure 1.2).

- **Filtre passe bande**

Le filtre passe bande contient un intervalle fréquentiel entre $f_{C1}$ et $f_{C2}$qui s'appelle bande passante, il laisse passer les fréquences comprises entre $f_{C1}$ et $f_{C2}$ et attenue les autres fréquences (voir figure 1.2).

- **Filtre coupe-bande**

Le filtre coupe-bande est une combinaison entre le filtre passe bas et le filtre passe haut, il contient deux fréquences de coupure $f_{C1}$ et $f_{C2}$**,** il atténue les fréquences comprises entre $f_{C1}$ et $f_{C2}$ et laisse passer les autres fréquences (voir figure 1.2).

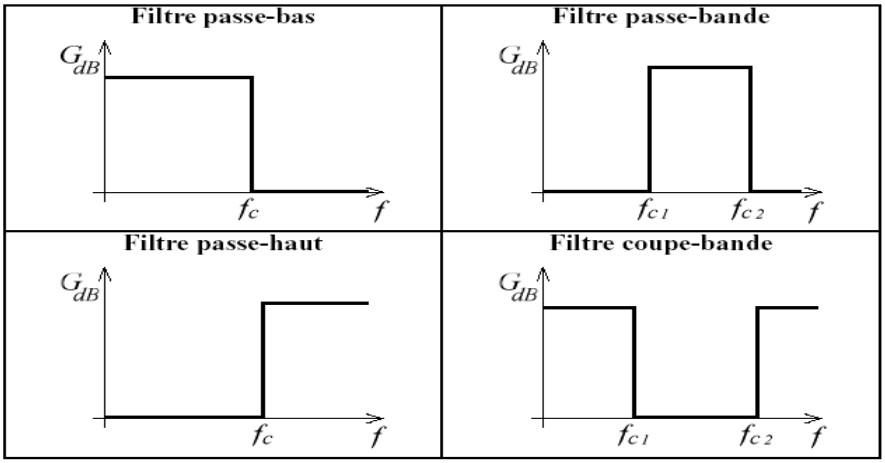


***Figure 1. 2*** Réponses fréquentielles idéales des 4 filtres de base.

## 1.4 Filtrage numérique

Le filtrage numérique est une technique de base dans le traitement du signal, il consiste à éliminer le bruit d'un signal numérique d'entrée pour obtenir un signal désirable en sortie en raison du développement des circuits intégrés rapides, les filtres numériques deviennent plus intéressants que les filtres analogiques en apportant de nombreux avantages tel que :

La précision, la fiabilité, la stabilité, l'adaptabilité et la facilité de commande.

### 1.4.1 Les spécifications d'un filtre numérique

#### *a Linéarité*

Un système est linéaire si la propriété de superposition s'applique c'est-à-dire (voir figure 1.3) :

Le système produit la sortie $y_1(n)$ pour l'entrée $x_1(n)$ et la sortie $y_2(n)$ pour l'entrée $x_2(n)$ Où :

$$y_1(n) = h(x_1(n)) \ et \ \ y_2(n) = h(x_2(n)) \tag{1.1}$$

Le système h est linéaire si

$$x_3(n) = ax_1(n) + bx_2(n) \quad \text{donc} \quad y_3(n) = h(x_3(n)) \ \Leftrightarrow \ ah\big(x_1(n)\big) + bh\big(x_2(n)\big) = ay_1(n) + by_2(n) \tag{1.2}$$

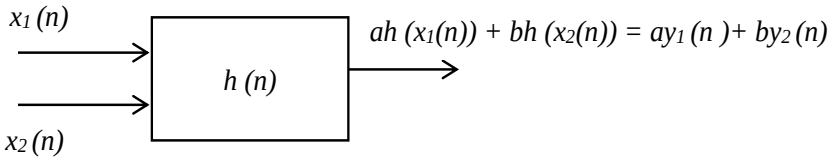


***Figure 1. 3*** Système linéaire.

### 1.4.2 Invariance dans le temps

Un système est invariant en temps si un délai à l'entrée ne cause que le même délai à la sortie, si $y_1(n)$ est la sortie qui correspond à l'entrée $x_1(n)$ et $x_2(n) = x_1(n - n_0)$ est l'entrée qui produit une sortie $y_2(n)$. Dire que ce système est invariant en temps si $y_2(n) = y_1(n - n_0)$ (voir figure 1.4).


***Figure 1. 4*** Système invariant dans le temps.

### 1.4.3 Causalité

Un filtre est dit causal si à une entrée nulle pour $n < 0$ correspond une réponse impulsionnelle $s = h(e)$ nulle pour $n < 0$, autrement dit si sa réponse impulsionnelle $h(n)$ est nulle pour tout $n < 0$. Sa transformée en z converge pour $|z| > R_1 \geq 0$

Un filtre numérique est dit anti causal, si sa réponse impulsionnelle $h(n)$ est nulle pour $n \geq 0$. Sa transformée en z converge pour$|z| < R_2 \leq +\infty$.

### 1.4.4 Les caractéristiques des filtres numériques

- **La réponse impulsionnelle**

La réponse impulsionnelle est la sortie qui est obtenue lorsque l'entrée est une impulsion unité discrète $\delta(n)$ , c'est-à-dire une variation soudaine et brève du signal.

- **Principe de stabilité**

Un filtre est stable si pour toute entrée $e(n)$ bornée, la sortie $h(n)$ est elle aussi bornée.

- **L'équation aux différences**

Un filtre numérique peut être définie par une équation aux différences, cette équation régi l'entrée $e(n)$ et la sortie $s(n)$ de système.

$$s(n) + a_1 s(n-1) + \cdots + a_N s(n-N) = b_0 e(n) + b_1 e(n-1) + \cdots + b_M e(n-M) \quad (1.3)$$

- **La fonction de transfert**

La fonction de transfert permet de définir un filtre numérique dans le domaine fréquentiel (transformée en $z$). La fonction de transfert générale d'ordre *N* d'un filtre numérique est la suivante :

$$h(z) = \frac{y(z)}{x(z)} = \frac{\sum_{k=0}^{N} b_k z^{-k}}{1+\sum_{k=1}^{M} a_k z^{-k}} \quad (1.4)$$

Les valeurs des coefficients $a_k$ et $b_k$ fixeront le type du filtre.

### 1.4.5 Réalisation d'un filtre numérique

Les filtres numériques peuvent être réalisés à l'aide des trois éléments ou opérations de base. Soit l'élément gain, l'élément de sommation et le retard unitaire. Ces éléments sont suffisants pour réaliser tous les filtres numériques linéaires possibles.

## 1.5 Convolution

La convolution est une loi de composition entre deux signaux, permettant de calculer le signal de sortie à partir du signal d'entré et de sa réponse impulsionnelle, cette loi est définie de la manière suivante :

$$z(n) = x(n) * y(n) = \sum_{m=-\infty}^{+\infty} x(m)y(n-m) \quad (1.5)$$

## 1.6 Corrélation

La corrélation est un ensemble d'applications fondamentales de détection, extraction, détermination de caractéristiques dynamiques d'un système linéaire. Les fonctions de corrélation traduisent la similitude d'un signal ou de deux signaux au niveau de la forme et de la position en fonction du paramètre de translation t [1].

- **Autocorrélation**

Autocorrélation est une étude de la ressemblance du processus avec lui-même au cours du temps, et par conséquent, si le signal est périodique, la fonction d'autocorrélation permettra de détecter cette périodicité [1]. La fonction d'autocorrélation est définie par:

$$R_{xx}(\tau) = \sum_{i=-\infty}^{+\infty} x(n)x^*(n-\tau) \quad (1.6)$$

- **Intercorrélation**

Intercorrélation est une étude de la ressemblance de deux signaux et sa fonction est définie par :

$$R_{xy}(\tau) = \sum_{i=-\infty}^{+\infty} x(n)y^*(n-\tau) \quad (1.7)$$

## 1.7 Classification des filtres numériques

Il existe deux catégories des filtres numériques linéaires. Selon la durée de la réponse impulsionnelle.

### 1.7.1 Filtre à réponse impulsionnelle finie (RIF)

Les filtres numériques à réponse impulsionnelle finie sont caractérisés par une réponse impulsionnelle basée sur un nombre limité d'échantillons d'un signal, ils sont cependant très largement utilisés car ils possèdent des propriétés uniques (phase, linéarité, stabilité, flexibilité), ces filtres sont aussi connus par nom filtres non récursif.

De façon générale le filtre à réponse impulsionnelle infini est décrit par la fonction de transfert (1.8) et l'équation aux différences (1.9) suivantes :

$$H(z) = \sum_{i=0}^{N-1} b_i z^{-i} \tag{1.8}$$

$$y(n) = \sum_{i=0}^{N-1} b_i\, x(n-i) = \sum_{i=0}^{N-1} h(i)x(n-i) \tag{1.9}$$

$N$ est la longueur de la réponse impulsionnelle

On remarque qu' en exploitant l'équation (I.8) et que les coefficients $b_{\mathrm{i}}$ de filtre sont également les valeurs de la réponse impulsionnelle $h(n)$, qui se trouve donc être limitée dans le temps [2].

$$\mathrm{h(n)} = \sum_{\mathrm{i=0}}^{\mathrm{N\text{-}1}} \mathrm{b_i}\delta\,(\mathrm{n\text{-}i}) \tag{1.10}$$

$$h(n) = \begin{cases} b_i, & 0 \le n \le N-1 \\ 0, & ailleurs \end{cases} \tag{1.11}$$

Puisque la réponse (1.10) est une somme d'un nombre fini de valeurs, le filtre RIF est naturellement stable d'après le critère Entrée Bornée/Sortie Bornée.

#### *a Structure de filtre RIF*

Le filtre RIF peut être réalisé par deux façon : la structure directe (Voir figure 1.5) et la structure transposée (Voir figure 1.6).

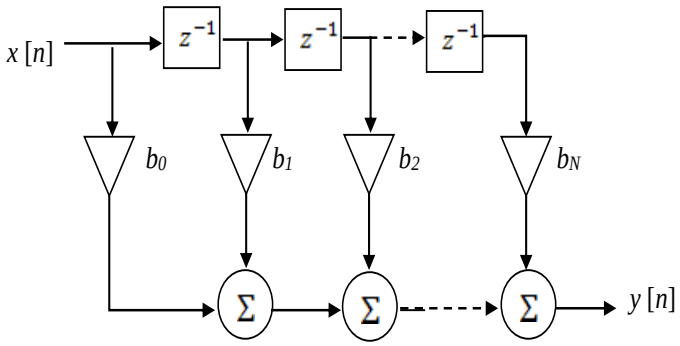


***Figure 1. 5*** Structure directe d'un filtre RIF.

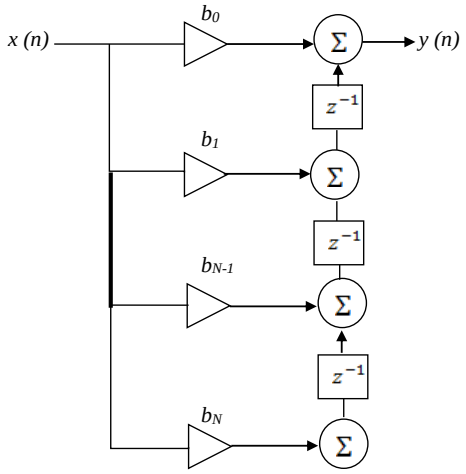


***Figure 1. 6*** Structure transposée d'un filtre RIF.

Ces structures nécessites $N-1$ cases mémoire et a une complexité de calcul de $N$ multiplications et $N-1$ additions par échantillons de sortie.

### *b Les principales caractéristiques des filtres RIF*

- Les méthodes de conception de filtre RIF simple met en œuvre un système numérique de traitement.
- Toujours stables.
- Une phase qui peut être exactement linéaire.
- Des méthodes de synthèse permettant de dériver n'importe quelle réponse fréquentielle.
- Une bande de transmission qui sera toujours plus large qu'un filtre RII ayant le même nombre de coefficients.
- Pas d'équivalent analogique [3].

### *c Synthèse des filtres numériques à réponse impulsionnelle finie*

L'emploi des filtres RIF peut se révéler attrayant eu égard à ses nombreux avantages : stabilité inconditionnelle (Tous les pôles sont en 0), phase linéaire possible. Néanmoins, ils présentent l'inconvénient de nécessiter un plus grand nombre de coefficients que les filtres RII pour obtenir les mêmes caractéristiques fréquentielles à cause de l'absence de pôles hors 0. Ainsi, toute fonction de filtrage numérique stable et causale peut être approchée par la fonction de transfert d'un filtre RIF [4].

- **Méthode de la fenêtre**

Cette technique consiste, connaissant l'expression analytique $H(f)$ de la réponse fréquentielle continue (dont la formulation mathématique connue) à approcher, à déterminer par utilisation de la transformée de Fourier à temps discret inverse, la réponse impulsionnelle. Cette réponse temporelle non causale obtenue sera retardée pour la rendre causale. Ainsi :

A partir du gabarit idéal du filtre, on détermine les coefficients du filtre en limitant le calcul à $N$ valeurs réparties symétriquement autour de $n = 0$. Puis, on calcule de la TFTD inverse du filtre idéal qui nous permettra de retrouver les échantillons de la réponse impulsionnelle soient les coefficients du filtre :

$$h(n) = \begin{cases} \frac{1}{f_e}\int_{-f_e/2}^{f_e/2} h(f)e^{2\pi jfnTe}\,df & N\ impair \\ \frac{1}{f_e}\int_{-f_e/2}^{f_e/2} h(f)e^{j2\pi f(n-\frac{1}{2})Te}\,df & N\ pair \end{cases} \quad (1.12)$$

Cette méthode produit une série infinie de coefficients, on limite, alors la réponse impulsionnelle à N échantillons (troncature). Sachant que la troncature induit des ondulations, on peut faire appel aux fenêtres de pondération pour les atténuer. Ainsi, la réponse impulsionnelle idéale $h(n)$ sera multipliée par la fenêtre discrète $\omega_N(n)$ de longueur $N$ [4].

$$h'_N(n) = h(n)\omega_N(n) \quad (1.13)$$

- **Méthode de l'échantillonnage fréquentiel**

La méthode de synthèse par échantillonnage en fréquence est appliquée depuis la réponse fréquentielle d'un filtre continu idéal $H(f)$ dont on ne connait pas la formule mathématique (on ne peut alors calculer $h(n)$ par TF inverse de $H(f)$. On utilise alors la transformation de Fourier Discrète inverse. C'est-à-dire que l'on "échantillonne" la réponse désirée dans le domaine fréquentiel, on obtient $N$ points de cette réponse fréquentielle auxquels on fait correspondre $N$ points de la réponse temporelle équivalente obtenus par TFD inverse comme suit :

On commence par échantillonner

$$H(k) = H(f)|f = k/n \qquad k = -(N-1)/2 \text{ à } (N-1)/2 \tag{1.14}$$

Puis on applique la TFD inverse

$$h(n) = \frac{1}{N}\sum_{k=-(N-1)/2}^{(N-1)/2} H(k)e^{2\pi jkn/N} \tag{1.15}$$

Cette méthode de synthèse est très simple et permet de réaliser toute forme de filtre (chose qu'on ne peut réaliser avec la méthode précédente). Cependant, cette méthode de synthèse ne garantit que les points fréquentiels $H(k)$. Entre ces points, la valeur de $H(f)$ n'est pas maitrisée, il peut y avoir des oscillations qui ne sont pas également réparties avec un maximum d'erreur entre la réponse idéale et la réponse obtenue se situant autour de la bande de transition. Pour obtenir la réponse en fréquence du filtre finalement obtenu, on peut par exemple appliquer une TFD à la réponse impulsionnelle $h(n)$ de taille $N$ obtenue, après avoir ajouté un grand nombre de zéros. Par ailleurs, du fait de l'emploi d'une TFD inverse sur $N$ points, la réponse impulsionnelle $h(n)$ obtenue est périodique de période $N$ bien que la réponse impulsionnelle idéale souhaitée ne soit pas de durée limitée [4].

### 1.7.2 Filtre à réponse impulsionnelle infinie (RII)

Les RII sont des filtres à "mémoire infinie", c'est-à-dire qu'ils gardent pendant un temps infini la mémoire du signal d'entrée, grâce à une boucle de retour, Ces filtres sont aussi connus par nom filtres récursif.

De façon générale le filtre à réponse impulsionnelle infini est décrit par la fonction de transfert (I.16) et l'équation aux différences (I.17) suivantes :

$$\mathrm{H(z)} = \frac{\mathrm{N(z)}}{\mathrm{D(z)}} = \frac{\sum_{\mathrm{i=0}}^{\mathrm{N}} \mathrm{b_i\, z^{-i}}}{1+\sum_{\mathrm{i=1}}^{\mathrm{N}} \mathrm{a_i\, z^{-i}}} \quad (1.16)$$

$$y[n] = \sum_{K=0}^{N} b_i\, x[n-i] - \sum_{i=0}^{n} a_i\; y(n-i) \quad (1.17)$$

Avec $N$ est l'ordre de filtre.

A Partir de la fonction de transfert, deux cas se présentent :

- Si $N(z)$ n'est pas divisible par $D(z)$, on a un nombre infini de termes dans la division polynomiale de $N(z)$ par $D(z)$ [2]:

$$H(z) = \sum_{n=0}^{N} c_n\; z^{-n} \quad (1.18)$$

$$h(n) = c_n \quad \text{pour } n = 0 \ldots . \infty \quad (1.19)$$

$H(z)$ est un filtre RII.

- Si $N(z)$ est divisible par $D(z)$, on a un nombre fini dans la division polynomiale de $N(z)$ par $D(z)$ [2]:

$$H(z) = \sum_{n=0}^{N} c_n\; z^{-n} \quad (1.\ 20)$$

$$h(n) = c_n \quad \text{pour} \quad n = 0 \ldots . N-1 \quad (1.\ 21)$$

#### *a Structure de filtre RII*

Le filtre RII peut être réaliser par deux structures : la structure direct (voir figure 1.7) et la structure cascade (voir figure 1.8).

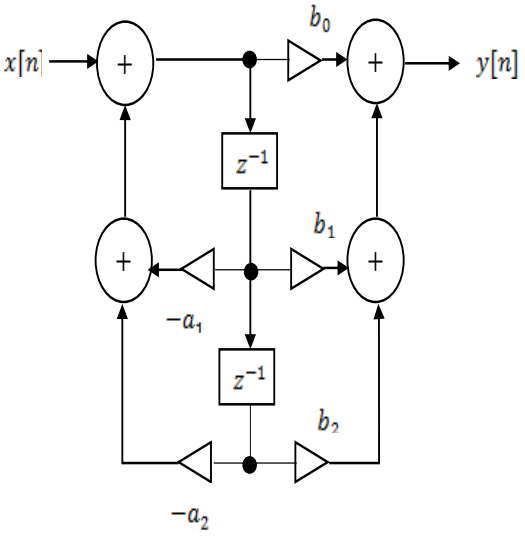


***Figure 1. 7*** Structure direct d'un filtre RII.

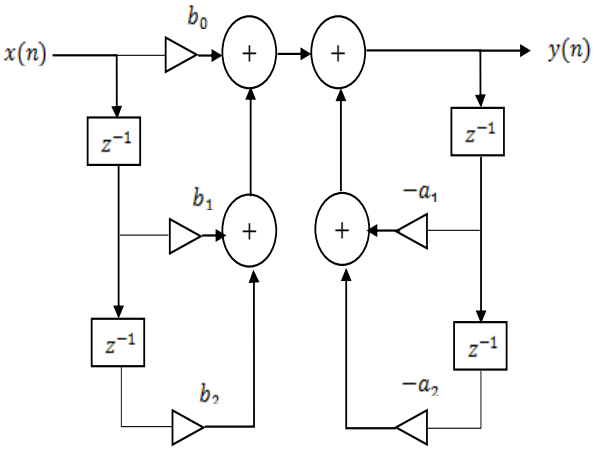


***Figure 1. 8*** Structure cascade d'un filtre RII.

### *b Les principales caractéristiques des filtres RII*

- Une bande de transition qui peut être étroite.
- Une instabilité potentielle due à des pôles de $H\ (z)$ situés en dehors du cercle unité.
- Phase non linéaire distorsion de phase.
- Une plus grande sensibilité numérique (quantification des coefficients, bruits de calculs).
- Nécessitent moins d'opérations et de places mémoires.
- Plus efficaces que RIF.
- Peuvent être dériver des filtres analogiques [3].

### *c Synthèse des filtres numériques à réponse impulsionnelle infinie*

La conception et la réalisation des filtres numériques à réponse impulsionnelle infinie sont essentiellement basées sur la fonction de transfert $H(z)$ : gabarit de filtrage de type passe-bas, passe-haut, passe-bande ou coupebande. Le principe est de calculer un filtre analogique $H(s)$ et le transformer en un filtre numérique équivalent $H(z)$ dont la contrainte est de conserver la stabilité du filtre analogique c'est-à-dire qu'on doit transformer le demi-plan complexe gauche en l'intérieur du cercle unité, et l'axe des imaginaires en cercle unité.

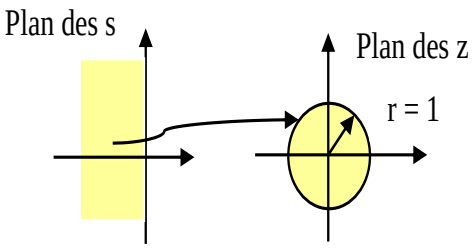


***Figure 1. 9*** Transformation du plan des s au plans des z.

Il existe plusieurs méthodes de synthèse des filtres RII, on va citer deux méthode parmi eux qui sont les suivantes :

- **Transformation standard ou méthode d'invariance impulsionnelle**

Par cette méthode on obtient un filtre numérique dont la réponse impulsionnelle est égale à la réponse impulsionnelle échantillonnée du filtre analogique correspondant. En considérant la fonction de transfert $H(p)$ ou $H(f)$ et la réponse impulsionnelle $h(t)$ du filtre analogique, la réponse impulsionnelle, échantillonnée à la période *Te*, s'exprime par :

$$h_e(t) = T_e \sum_{k=0}^{+\infty} h(kT_e)\delta(t - kT_e) \qquad (1.22)$$

Le coefficient $T_e$ correspond au fait que la réponse impulsionnelle étant échantillonnée, la fonction de transfert est périodisée avec la fréquence $F_e$. Soit la relation:

$$H_e(f) = H(f) * Pgn_{Fe}(f) \text{ d'où } h_e(t) = h(t)\,[T_e Pgn_{Te}(t)] \qquad (1.23)$$

Par conséquent la transformée en $zH_e(z)$ de $h_e(t)$ est donnée par [1]:

$$H_e(z) = T_e \sum_{k=0}^{+\infty} h(kT_e)\, z^{-kT_e} \quad (1.24)$$

- **Synthèse par transformation bilinéaire**

Cette méthode permet de définir le gabarit du filtre numérique et le convertir en un gabarit correspondant au filtre analogique par la relation suivante :

$$f_d = \frac{1}{\pi T_e} \tan^{-1}(\pi f_a T_e) \quad (1.25)$$

Ensuite Faire la synthèse du filtre analogique (Butterworth, Tchebychev …) $\Longrightarrow H_a(s)$, et pour Transformer $H_a(s)$ en $H_d(z)$ en utilisant :

$$s = \frac{2}{T_e} \frac{1-z^{-1}}{1+z^{-1}} \quad (1.26)$$

## 1.8 Conclusion

La présentation faite dans ce chapitre nous a permis d'avoir une vue globale sur les filtres numériques, pour cela nous allons intéresser aux filtres à réponse impulsionnelle finie car ils sont plus stables et réalisables.

Dans le prochain chapitre nous allons traiter le problème de l'écho acoustique et les algorithmes utilisés pour l'annulation de ce dernier par le filtrage adaptatif.

# Chapitre 2 Annulation d'écho par le filtrage adaptatif

## 2.1 Introduction

Dans les systèmes de communication main-libre et conférence (téléconférence, conférence de bureau…), on trouve le phénomène d'écho acoustique qui est provoqué à partir du couplage entre le haut-parleur et le microphone, ce phénomène causant la mauvaise qualité de communication et la fatigue d'auditeur, pour cela on a besoin des techniques pour la résolution de ce problème et l'amélioration de qualité de communication. Parmi ces dernières les plus courantes, c'est bien le filtrage adaptatif. Le filtrage adaptatif intervient quand il faut réaliser, simuler ou modéliser un système dont les caractéristiques évoluent dans le temps. Il conduit à la mise en œuvre de filtre aux coefficients variant dans le temps. Les variations des coefficients sont définies par un critère d'optimisation et réaliser suivant un algorithme d'adaptation, qui est déterminé en fonction de l'application [5].

## 2.2 L'écho acoustique

L'origine de l'écho acoustique provient de l'utilisation de nouveaux systèmes de télécommunication dits ''mains libres''. Au début des télécommunications l'utilisateur était obligé de coller son oreille à combiné pour entendre son interlocuteur distant.

Aujourd'hui, les nouveaux systèmes de télécommunications permettent la liberté de mouvement du locuteur en restituant le son de l'interlocuteur sur un haut-parleur (voir figure 2.1).

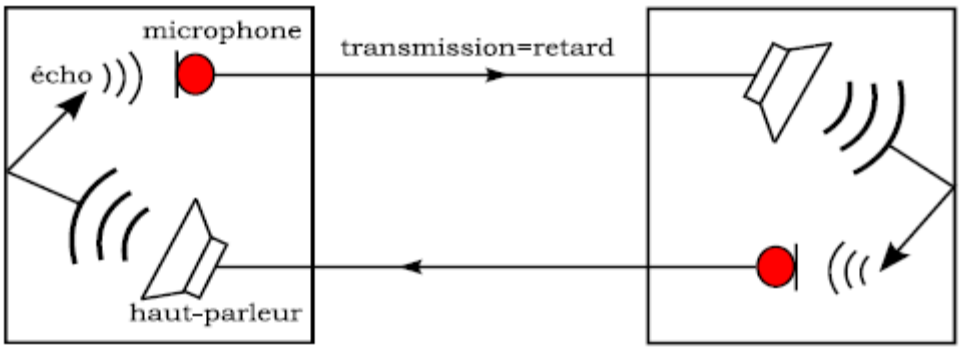


***Figure 2. 1*** Exemple de communication bi-directionnelle avec écho acoustique.

Le couplage acoustique, généré par l'utilisation de tels systèmes de télécommunications mains libres, provoque certains effets indésirables comme le phénomène de l'écho acoustique ou encore l'instabilité de la boucle de communication [6].

L'écho acoustique est provoqué par la transmission du signal émis par le haut-parleur et reçu par le microphone : cette transmission est composée d'un trajet direct et de multiples réflexions captées par le microphone, et a pour conséquence de renvoyer vers le locuteur qui a prononcé la parole dans une salle distante son propre signal. C'est donc la propagation acoustique d'une onde sonore à l'intérieur d'un volume donné qui provoque l'écho acoustique.

Le phénomène de l'écho acoustique présente des complexités du fait que ses propriétés acoustiques sont très variables en fonction de l'environnement correspondant. Il suffit de s'intéresser à quelques exemples d'utilisation de systèmes mains libres pour apprécier la difficulté du problème. Les réseaux de télécommunications actuels supportent des débits d'information considérables et autorisent la transmission simultanée et en temps réel d'images et de son. Cette avancée technologique permet par conséquent l'organisation de téléréunions entre locuteurs de sites distants en leur offrant une sensation de présence et de naturel.

Les exemples les plus standards d'applications de téléréunions sont par exemple, la téléconférence et la visioconférence sur PC. Pour la téléconférence, une salle spécialement conçue pour cette application est généralement utilisée.

Lorsqu'un son est émis à l'intérieur d'une salle (ou d'une voiture), il subit des transformations physiques qui peuvent être comprises grâce aux principes de l'acoustique des salles. Des interprétations théoriques précises peuvent être obtenues en faisant appel aux domaines de l'acoustique géométrique, ondulatoire et statistique [7]. Néanmoins, le phénomène physique peut être décrit simplement et succinctement comme suit. Une onde sonore, émise par un émetteur, se propage suivant les lois de l'acoustique vers un récepteur. Au cours de son trajet, l'onde subite l'influence de l'environnement acoustique dans lequel elle se propage. Le phénomène se résume, dans le cas d'une propagation dans un espace libre, à l'absorption d'une onde sonore par l'air qui dépend de paramètres (température, pression atmosphérique, etc.…) variant lentement dans le temps par rapport à l'échelle de stationnarité du signal sonore. S'ajoutent à cela des phénomènes de réflexion, diffraction, diffusion, et absorption provoqués par les parois et obstacles présents dans l'espace clos. Le trajet de propagation d'une onde sonore est appelé canal acoustique. Le canal acoustique dépend directement de ces différents paramètres.

La réponse impulsionnelle d'un canal acoustique se présente sous la forme d'une onde directe et d'une succession d'ondes réfléchies par les parois d'une salle particulière.

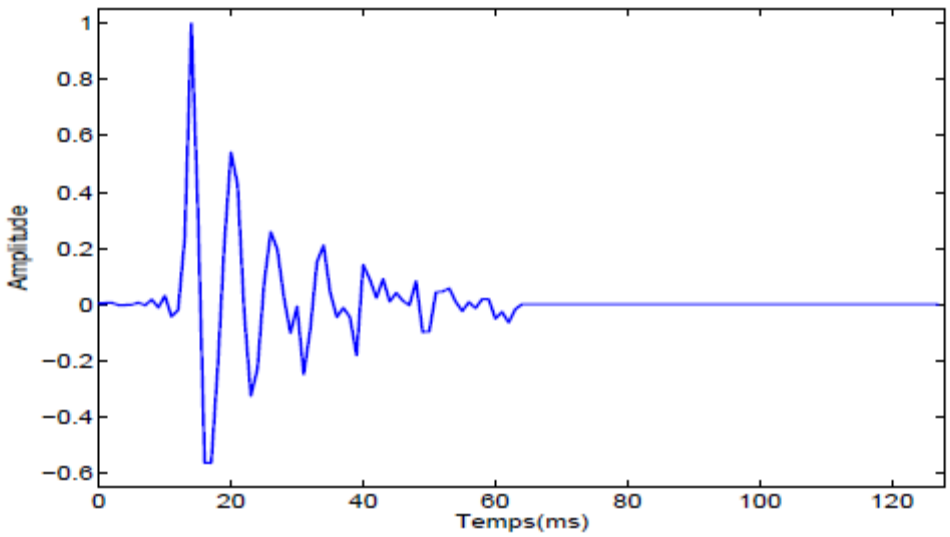


***Figure 2. 2*** Exemple de réponse impulsionnelle du canal acoustique.

Les ondes se propagent en trajet direct (le trajet le plus court emprunté par l'onde sonore) jusqu'à ce qu'elles rencontrent un obstacle sur lequel elles se réfléchissent tout en perdant de l'énergie. En réalité, la réponse impulsionnelle de couplage acoustique est de durée infinie mais il est généralement admis que son support temporel significatif est de l'ordre de 50 à 100 ms dans une voiture, et de 250 ms à 300 ms dans une salle de téléconférence.

L'écho acoustique, résultant du couplage acoustique entre un haut-parleur et un microphone, peut donc être caractérisé par la réponse impulsionnelle du canal acoustique correspondant. Cette réponse impulsionnelle qui est très sensible et dépendante de son environnement acoustique, peut varier rapidement d'un instant à un autre, puisque la taille de la salle, le revêtement des murs, la présence d'objets ou de personnes dans la salle, etc…., sont autant de paramètres qui influent sur la nature du couplage acoustique et modifient cette réponse.

La figure (2.2) montre que la perceptibilité d'un signal de parole diminue de 0,5 dB par milliseconde (de retard d'écho). Le système binaural d'audition humaine gère remarquablement bien l'effet de réverbération lorsque les personnes en conversations sont dans la même salle même très réfléchissante.

Ce n'est pas le cas si les mêmes personnes sont en salles différentes et ils utilisent un haut-parleur pour la conversation. L'écho acoustique devient très gênant et inquiétant et donc doit être enlevé.

L'acoustique de la salle affectera toujours le son et peut entraver la communication.

En outre, un sifflement peut se produire si le microphone est placé trop près du haut-parleur (ce phénomène est connu par l'effet Larzen) et il doit être éliminé par annulation d'écho acoustique. La salle acoustique est une question complexe, mais elle a été largement étudiée en profondeur à partir de deux perspectives théoriques et pratiques (voir figure 2.3).

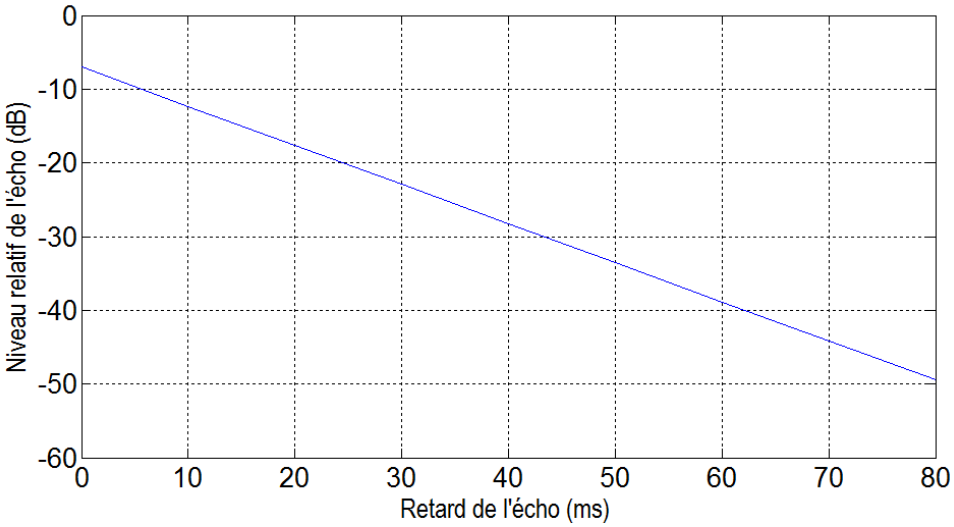


***Figure 2. 3*** Influence de l'écho sur la perception de la parole.

### 2.2.1 La gêne provoquée par l'écho

Avant de présenter les différentes techniques de traitement de l'écho acoustique, nous présentons quelles conditions l'écho est perçu comme une perturbation gênante.

L'écho acoustique est présent de façon gênante pour un délai de transmission global de 30 ms. Ce délai est largement dépassé que ce soit dans le cadre de la radiotéléphonie mains libres où le retard de transmission est de l'ordre de 180 ms ou dans des contextes de téléconférence où le traitement et la transmission de la voix introduisent un retard supérieur. Dans ce cas, un traitement spécifique doit être impérativement mis en œuvre pour préserver la qualité de la communication.

Dans le cas de simple parole et pour des retards inférieurs à 25ms, un système d'annulation d'écho doit fournir une atténuation de l'écho de l'ordre de 24 dB. Ce même système doit être capable de fournir une atténuation de l'écho de 40 dB pour des retards excédant 25 ms [8].

### 2.2.2 Traitements classiques de l'écho

Dans ce paragraphe, nous proposons de décrire le concept du système d'annulation d'écho acoustique et de présenter les principales méthodes algorithmiques existantes.

Nous nous intéressons plus spécifiquement à la résolution du problème posé par l'écho acoustique, résolution basée sur des techniques de filtrage adaptatif. Les algorithmes de filtrage adaptatif sont très nombreux et ont été largement étudiés dans la littérature.

Nous en rappelons les principaux en les classifiant par famille, même s'il a été démontré que tous les algorithmes adaptatifs sont liés entre eux et peuvent se déduire les uns des autres au moyen d'approximations [9]. Dans ce qui suit, nous présentons l'algorithme LMS (Least mean square), sa version normlisée NLMS (normalized LMS), et une version améliorée NLMSA (NLMS améliorée) pour l'annulation d'écho acoustique.

## 2.3 Le filtrage adaptatif

Un filtrage est rendu adaptatif si ses paramètres, les coefficients, sont modifiés selon un critère donné, dès qu'une nouvelle valeur du signal devient disponible. Ces modifications doivent suivre l'´évolution des systèmes dans leur environnement aussi rapidement que possible. Le filtrage adaptatif est généralement associé avec un fonctionnement en temps réel (dans le cas où les coefficients du filtre seraient variables dans le temps) [10].

### 2.3.1 Le rôle du filtre adaptatif

Le rôle primordial d'un filtre adaptatif est d'ajuster le paramètre $w$ pour un objectif bien défini (minimisation de l'EQM : erreur quadratique moyenne).Le principe d'un filtre adaptatif bouclé par un algorithme d'adaptation est présenté sur la figure suivante.

### 2.3.2 Principe de base du filtre adaptatif

La technique de filtrage adaptatif se décompose classiquement en deux étapes (voir figure 2.4):

- une étape de filtrage qui permet d'obtenir une estimation du signal inconnu en convolant le signal d'entrée $x(n)$ avec les coefficients du filtre adaptatif $W$ .

L'erreur d'estimation $e(n) = d(n) - y(n)$ est ensuite utilisée dans la partie adaptation pour mettre à jour les coefficients du filtre.

- une étape d'adaptation qui permet d'ajuster les coefficients du filtre adaptatif $W$ suivant un algorithme donne [11].

À chaque itération les coefficients du filtre varient en du signal d'erreur $e(n)$ et ce pour faire diminuer la différence entre la sortie du filtre $y(n)$ et le signal désirée $d(n)$. Le signal d'erreur diminuer jusqu'atteindre dans certain cas une valeur nulle. À ce moment les coefficients du filtre adaptatif cessent de s'adapter.

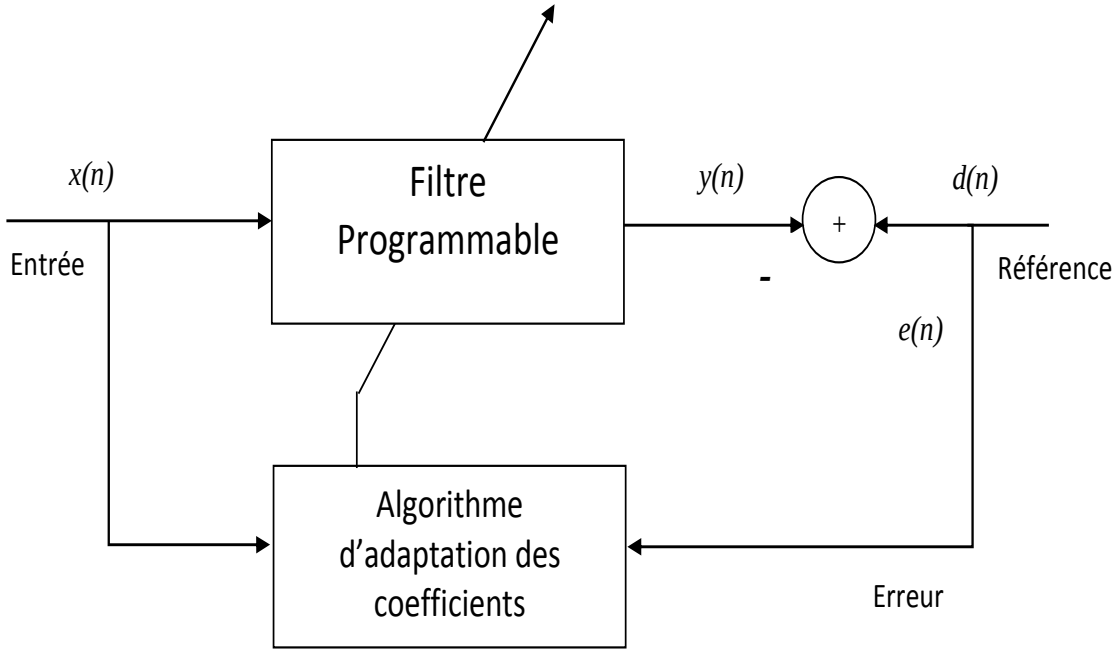


***Figure 2. 4*** Principe d'un filtre adaptatif.

$x(n)$ : signal de l'entrée.

$y(n)$ : signal de sortie du filtre.

$d(n)$ : écho ou signal désirer (Echo or desired signal).

$e(n)$ : erreur obtenu par la méthode de filtrage adaptatif qui consiste que les deux ensembles de coefficients sont égaux.

Les filtres adaptatifs peuvent être classés en fonction des choix qui sont faits sur les points suivants:

• le critère d'optimisation,

• l'algorithme de mise `à jour des coefficients,

• la structure du filtre programmable,

• le type de signal traité, mono ou multidimensionnel.

Il existe deux classes importantes de filtres linéaires optimaux:

• filtrage de Wiener (ou les signaux considérées $d(n)$ et $x(n)$ sont stationnaires),

• filtrage de Kalman (qui est une généralisation du filtre de Wiener valable aussi dans le cas de processus (Ou de signaux) non stationnaires).

### 2.3.3 Critères de comparaison des algorithmes adaptatifs

- Taux de convergence : nombre d'itérations pour converger suffisamment prés de solution de Wiener.
- Désajustement : moyen d'ensembles de l'erreur quadratique finale) - (erreur quadratique minimale obtenue avec Wiener)
- Robustesse : résistance au mauvais conditionnement des données.
- Complexité : nombre d'opération par itération + place mémoire nécessaire (programme et données).
- Structure : Aspect hardware, complexité de l'implantation matérielle.
- Stabilité numérique : influence des erreurs de quantification problème de la propagation des erreurs.

## 2.4 Définition d'un algorithme

Un algorithme est un énoncé d'une suite d'opérations permettant de donner la réponse à un problème. Un algorithme est une spécification d'un schéma de calcul sous forme d'une suite finie d'opérations élémentaires obéissant à un enchainement déterminé.

### 2.4.1 Choix de l'algorithme

Le choix de l'algorithme se fera en fonction des critères suivants [12]:

- La mesure de cette 'proximité' entre cette solution e la solution obtenue,

- ❖ La capacité de poursuite (traking) des variations (non-stationnarités) du système,
- ❖ La robustesse au bruit,
- ❖ La complexité,
- ❖ Les propriétés numériques (stabilité et précision) dans le cas d'une précision limitée sur les données et les coefficients du filtre.

## 2.5 Filtre de Wiener

Le filtrage de Wiener est adéquat pour les situations dans lesquelles le signal ou le bruit sont stationnaires (Voir figure 2.5)

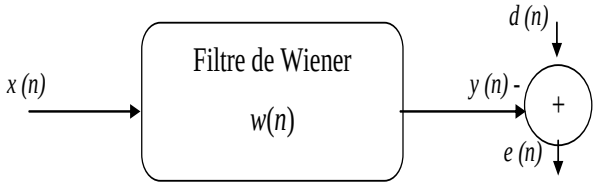


***Figure 2. 5*** Schéma principale du filtre de Wiener.

Le principe du filtre de Wiener est de trouver en sortie une réponse $y(n)$ la plus proche possible d'une réponse désirée $d(n)$ lorsque l'entrée est une certaine séquence $x(n)$.

On note : $e(n) = d(n) - y(n)$ l'erreur entre la réponse désirée $d(n)$et la sortie de filtre $y(n)$.On note également $w_n$ la réponse impulsionnelle du filtre.

La sortie du filtre $y(n)$ s'écrit :

$$\boldsymbol{y(n)} = \sum_{\boldsymbol{k=0}}^{\boldsymbol{M-1}} \boldsymbol{w}(k)\, \boldsymbol{x(n-k)} \qquad (2.1)$$

Où

$\boldsymbol{w}(n) = [w(0), w(1) \dots w(M-1)]^T$

$\boldsymbol{x}(n) = [x(n), x(n-1) \dots . x(n-M+1)]^T$

Le filtre de Wiener est celui qui minimise l'erreur quadratique moyenne (EQM).

$$J = E[e(n)e^*(n)] = E[e(n)^2] \qquad (2.2)$$

En introduisant les vecteurs $\boldsymbol{w}(n)$ et $\boldsymbol{x}(n)$ on aura :

$$e(n) = d(n) - \boldsymbol{w}^H(n)\boldsymbol{x}(n) \quad (2.3)$$

D'où : $J = E[\big(d(n) - \boldsymbol{w}^H(n)\boldsymbol{x}(n)\big)\big(d^*(n) - \boldsymbol{w}^T(n)\boldsymbol{x}^*(n)\big)]$ (2.4)

$$J = E[d^2(n)] - \boldsymbol{w}^H(n)E[\boldsymbol{x}(n)d^*(n)] - \boldsymbol{w}^T(n)E[\boldsymbol{x}^*(n)d(n)] - \boldsymbol{w}^H(n)E[\boldsymbol{x}(n)\boldsymbol{x}^*(n)]\boldsymbol{w}(n) \quad (2.5)$$

Par conséquent on aura :

$$J = \sigma_d^2 - \boldsymbol{w}^H(n)\boldsymbol{r} - \boldsymbol{w}^T(n)\boldsymbol{r}^* + \boldsymbol{w}^H(n)\boldsymbol{R}\,\boldsymbol{w}(n) \quad (2.6)$$

Avec

$\boldsymbol{r}$ : le vecteur d'inter-corrélation de l'entrée.

$\boldsymbol{R}$ : la matrice d'auto-corrélation entre la sortie désirée $d(n)$et l'entrée $\boldsymbol{x}(n)$, cette matrice est définie positive, et a symétrie hermitienne ($\boldsymbol{R} = \boldsymbol{R}^H$).

Cherchons le vecteur optimum celui qui annule le gradient de critère : $\nabla J_k = 0$.

En écrivant $J$ sous la forme $J = E[e(n)e^*(n)]$

On prend

$$w(k) = a(k) + jb(k) \quad (2.7)$$

Donc on aura :

$$\nabla J_k = E\left(e_n \frac{\partial e_n^*}{\partial a(k)} + e_n^* \frac{\partial e_n}{\partial a(k)} + e_n \frac{\partial e_n^*}{\partial b(k)} + e_n^* \frac{\partial e_n}{\partial b(k)}\right)$$

(2.8)

Ou les dérivées partielles intervenant dans l'équation (2.7) sont comme suit :

$$\frac{\partial e_n^*}{\partial a(k)} = \frac{\partial[d(n) - \sum_{k=0}^{M-1} w^* x(n-k)]}{\partial a(k)} = \frac{\partial[d^*(n)]}{\partial a(k)} - \frac{\partial[\sum_{k=0}^{M-1} w(n)x^*(n-k)]}{\partial a(k)}$$

$$= -x^*(n-k)$$

$$\frac{\partial e_n}{\partial a(k)} = \frac{\partial[d(n) - \sum_{k=0}^{M-1} w^* x(n-k)]}{\partial a(k)} = \frac{\partial[d(n)]}{\partial a(k)} - \frac{\partial[\sum_{k=0}^{M-1} w(n)x^*(n-k)]}{\partial a(k)}$$

$$= -x(n-k)$$

$$\frac{\partial e_n^*}{\partial b(k)} = \frac{\partial[d(n) - \sum_{k=0}^{M-1} w^* x(n-k)]}{\partial b(k)} = \frac{\partial[d^*(n)]}{\partial b(k)} - \frac{\partial[\sum_{k=0}^{M-1} w(n)x^*(n-k)]}{\partial b(k)}$$

$$= -Jx^*(n-k)$$

$$\frac{\partial e_n}{\partial b(k)} = \frac{\partial [d(n) - \sum_{k=0}^{M-1} w^* x(n-k)]}{\partial b(k)} = \frac{\partial [d(n)]}{\partial b(k)} - \frac{\partial [\sum_{k=0}^{M-1} w(n) x^*(n-k)]}{\partial b(k)}$$

$$= -Jx(n-k)$$

Par conséquent on a :

$$\nabla_k J = -2E[\boldsymbol{x}(n-k)e^*(n)]k = 0,1,\ldots \quad (2.9)$$

On note $e_0$ la valeur à optimum :

$$E[e_0^* \boldsymbol{x}(n)] = 0 \quad (2.10)$$

C'est le principe d'orthogonalité [13],signifiant que toutes les entrées $x(n)$ sont décorrélées de $e_n\ (n)$. (2.11)

En développent l'équation (2.8), on obtient :

$$E = (\boldsymbol{x}(n)d^*(n) - \boldsymbol{x}(n)\boldsymbol{x}^H(n)\boldsymbol{w}(n)) = 0 \quad (2.12)$$

Soit :

$$\boldsymbol{R}\boldsymbol{w}\ (n) = \boldsymbol{r} \quad (2.13)$$

Cette relation (2.13) est appelée formule de Wiener ou équation de wiener-Hopf. Cette solution donne le filtre optimal de Wiener :

$$\boldsymbol{w}(n) = \boldsymbol{R}^{-1}\boldsymbol{r}_{dx} \quad (2.14)$$

L'équation de Wiener-Hopf qui permet de calculer le filtre de Wiener optimal conduit à résoudre un système de M équations a M inconnues. Il peut être préférable de résoudre ce système par une méthode itérative (algorithme), notamment en se souvenant que la fonction de cout de quadratique, ce qui entraine que le minimum est unique. Les algorithmes adaptatifs permettent l'estimation du filtre adaptatif par le vecteur $w_n$de taille M à l'aide d'un critère basé sur l'erreur d'estimation a priori [10]. Cette erreur d'estimation appelée précédemment signal de différence, s'écrit pour chaque échantillon n :

$$e(n) = d(n) - \boldsymbol{w}^H(n)\boldsymbol{x}(n) \quad (2.15)$$

Où $\boldsymbol{x}(n)$: est le vecteur colonne des M derniers échantillon du signal haut-parleur. $\boldsymbol{w}^H(n)$ : désigne un vecteur ligne d'ordre M contenant des coefficients de la réponse impulsionnelle finie, l'exposant. $T$ désigne l'opérateur de transposition. La mise à jour du filtre à chaque instant est effectuée par un contre réaction de l'erreur d'estimation proportionnellement au gain d'adaptation (terme de correction).

### 2.5.1 Les algorithmes de gradient stochastique

#### *a La famille LMS*

L'un des premiers algorithmes est l'algorithme du gradient stochastique ou LMS (least-mean-square) conçu par Widrow et Hoff en 1959.Cet algorithme est basé sur une estimation simple et peu complexe du gradient. Ce type de filtre est très simple, mais il est inefficace pour la problématique de l'annulation d'écho en raison de la grande variation d'énergie contenue dans la vois, ces variation d'énergie provoque une divergence de filtre [9]. Pour résoudre ce problème, Haykin a introduit le gradient normalisé NLMS, la modification apportée consiste à normaliser la correction des coefficients en fonction de l'énergie du signale, le pas dans ce calcule varie de façon inversement proportionnelle à l'énergie contenue dans le signale. Ainsi en présence d'une grande énergie, l'adaptation du filtre et ralentie. Ce ralentissement permet d'éviter les cas de divergence qui pourraient subvenir avec le filtre LMS.

#### *b Algorithme du gradient stochastique LMS*

L'idée de type gradient stochastique et de remplacer la moyenne statistique dans l'algorithme de gradient déterministe de l'équation suivante par sa valeur instantanée [14]:

$$e(n) = d(n) - \boldsymbol{w}^H(n)\boldsymbol{x}(n)$$

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \mu[\boldsymbol{r} - \boldsymbol{R}h_n] \tag{2.16}$$

$\mu$ est le pas d'adaptation

$$\boldsymbol{r} \cong y(n)\boldsymbol{x}(n) \tag{2.17}$$

et

$$\boldsymbol{R} \cong \boldsymbol{x}(n)\boldsymbol{x}(n)^T \tag{2.18}$$

Remplaçant (2.17) et (2.18) dans (2.16) on obtient la relation suivante :

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \mu e(n)\boldsymbol{x}(n) \tag{2.19}$$

Tel que :

$$e(n) = d(n) - \boldsymbol{w}^H(n)\boldsymbol{x}(n) \tag{2.20}$$

La condition nécessaire et suffisante de convergence de l'algorithme (LMS) est :

$$0 < \alpha < \frac{2}{\lambda max} \tag{2.21}$$

Une étude plus précise (en moyenne quadratique), mais qui repose également sur des hypothèses contestables conduit à la condition plus contraignante :

$$0 < \alpha < \frac{2}{trace(\boldsymbol{R})} = \frac{2}{M\sigma_x^2} \tag{2.22}$$

Trace ($R$) : désigne la somme des éléments de la diagonale de la matrice $R$.

$\sigma_x^2$: désigne l'énergie du signal d'entrée $\boldsymbol{x}(n)$.

### *c Algorithme du gradient stochastique normalisé NLMS*

L'algorithme de gradient stochastique normalisé NLMS est une variante du LMS dont le gain d'adaptation est normalisé par l'énergie de signale d'entré $x(n)$.

Pour des signaux non stationnaire l'énergie de signal $x(n)$ varie avec le temps, l'algorithme LMS aura du mal à fonctionnes correctement puisque μ est constant. L'algorithme LMS normalisé est obtenu en minimisant la fonction cout suivant [15] :

$$J(n) = \|\boldsymbol{w}(n+1) - \boldsymbol{w}(n)\|^2 \tag{2.23}$$

Avec la contrainte :

$$d(n) = \boldsymbol{w}^T(n+1)\boldsymbol{x}(n) \tag{2.24}$$

Cela revient à minimiser la mise à jour des coefficients du filtre tout en minimisant le signal d'erreur pour$x(n)$. La solution de ce problème est obtenue en utilisent la technique des multiplieurs de LaGrange. En effet, on cherchera à minimiser par rapport à $w(n+1)$.

$$J(n) = \|\boldsymbol{w}(n+1) - \boldsymbol{w}(n)\|^2 + \lambda\ [d(n) - \boldsymbol{w}^T(n+1)\boldsymbol{x}(n)] \quad (2.25)$$

Où λ est le multiplieur de Lagrange on obtient :

$$\frac{\partial J(n)}{\partial w(n+1)} = 2\left[\boldsymbol{w}(n+1) - \boldsymbol{w}(n)] + \frac{\lambda}{2}\boldsymbol{x}(n)\right] = 0\ L \times 1 \quad (2.26)$$

Soit

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \frac{\lambda}{2}\boldsymbol{x}(n) \quad (2.27)$$

Or d'après la contrainte :

$$d(n) = \boldsymbol{w}^T(n+1)\boldsymbol{x}(n) \quad (2.28)$$

$$= \boldsymbol{w}^T\boldsymbol{x}(n) + \frac{\lambda}{2}\boldsymbol{x}^T(n)\boldsymbol{x}(n) \quad (2.29)$$

Ce qui donne :

$$\lambda = \frac{2e(n)}{\boldsymbol{x}^T(n)\boldsymbol{x}(n)} \quad (2.30)$$

Finalement nous obtiens l'algorithme NLMS :

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \frac{\lambda}{2}\boldsymbol{x}(n) = \boldsymbol{w}(n) + \frac{1}{\boldsymbol{x}^T(n)\boldsymbol{x}(n)}\boldsymbol{x}(n)e(n) \quad (2.31)$$

En pratique, pour mieux contrôler la mise à jour des coefficients du filtre, on introduit un facteur positif $\alpha$ où (0<$\alpha$<2):

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \frac{\alpha}{\boldsymbol{x}^T(n)\boldsymbol{x}(n)}\boldsymbol{x}(n)e(n) \quad (2.32)$$

En fait, pour $l$ assez grand et pour un signal stationnaire on a:

$$\frac{\alpha}{\boldsymbol{x}^T(n)\boldsymbol{x}(n)} = \frac{\alpha}{\sum_{i=0}^{L-1} x^2(n-1)} \approx \frac{\alpha}{L\sigma_x^2} = \mu \quad (2.33)$$

Qui est le pas d'adaptation du LMS. Pour éviter des difficultés numériques (division par des petits nombres) quand l'énergie du signal d'entrée est petite, on modifie l'algorithme comme suit:

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \frac{\alpha}{\delta + \boldsymbol{x}^T(n)\boldsymbol{x}(n)}\boldsymbol{x}(n)e(n) \quad (2.34)$$

Ou` $\delta$> 0 est un paramètre de régularisation.

### *d Stabilité de l'algorithme NLMS*

Pour simplifier, on suppose que $d(n) = 0$. L'erreur du signal:

$$e(n) = d(n) - \boldsymbol{w}^T(n)\boldsymbol{x}(n) \quad (2.35)$$

Et aussi appelée erreur "a priori" car elle utilise les coefficients du filtre avant la mise à jour. L'erreur "a postériori" est définie par:

$$è(n) = d(n) - \boldsymbol{w}^T(n+1)\boldsymbol{x}(n) \quad (2.36)$$

Et se calcule une fois que la mise à jour a été effectuée. L'algorithme peut être considéré comme stable si la valeur absolue de l'erreur "a posteriori" est plus petite que celle de l'erreur "a priori", ce qui est logique puisque e(n) exploite davantage d'informations. En remplaçant l'équation du NLMS:

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \frac{\alpha}{x^T(n)\boldsymbol{x}(n)}\boldsymbol{x}(n)e(n) \quad (2.37)$$

Dans l'erreur « a posteriori », on obtient:

$$è(n) = d(n) - \boldsymbol{w}^T(n+1)\boldsymbol{x}(n) = d(n) - \boldsymbol{w}^T(n)\boldsymbol{x}(n) - \alpha e(n) = e(n)[1-\alpha] \quad (2.38)$$

Donc

$$|è(n)| < |e(n)|$$
$$|e(n)[1-\alpha]| < |e(n)|$$
$$|[1-\alpha]| < 1$$
$$0 < \alpha < 2$$

Qui est la condition de stabilité de l'algorithme NLMS.

### *e Algorithme NLMS amélioré* [23]

L'algorithme NLMS (Normalized Least Mean Square) consiste à normaliser le pas d'adaptation µ dans l'algorithme LMS par l'énergie du signal d'entrée pour réduire au minimum l'effet de la variation de la puissance du signal d'entrée et de rendre ainsi la convergence plus au moins uniforme en passant d'une étape d'adaptation à une autre. Le pas d'adaptation µ de l'algorithme LMS est alors remplacé par un pas d'adaptation défini à chaque itération par :

$$\mu(n) = \frac{\mu}{x^T(n)\boldsymbol{x}(n)} \quad (2.39)$$

La convergence de cet algorithme est garantie pour un pas d'adaptation 0<μ<2. La mise à jour des coefficients du filtre adaptatif par l'algorithme NLMS est alors donnée par :

$$\boldsymbol{w}(n+1)= \boldsymbol{w}(n) + \frac{\mu}{\boldsymbol{x}^T(n)\boldsymbol{x}(n)+\delta}\boldsymbol{x}(n)e(n) \quad (2.40)$$

Ou` $\delta$> 0 est un paramètre de régularisation. L'algorithme NLMS a une convergence lente dans les régions de fréquences où le signal d'excitation a une faible énergie [11]. Par exemple, étudions la situation où le signal d'excitation $x$(n) a une faible énergie dans les fréquences hautes que dans les faibles, comme par exemple le signal de parole. Donc, le vecteur gradient estimé $x(n)e(n)$ sera faible dans les régions de hautes fréquences. Le facteur de normalisation, c'est à dire le scalaire $x^T(n)$ $x$(n), sera important à cause de la grande énergie dans les petites fréquences de $x$(n). Alors, pour les faibles énergies du signal d'excitation, les mises à jour de $w(n)$sont lentes. Notre algorithme NLMS amélioré est une variante du NLMS dont le gain d'adaptation est normalisé par l'énergie de l'erreur pour réduire l'erreur quadratique moyenne et augmenter la vitesse de convergence, le pas d'adaptation à chaque itération est donné par :

$$\mu(n)= = \frac{\mu}{\alpha\boldsymbol{x}^T(n)\boldsymbol{x}(n)+(1-\alpha)\beta} \quad (2.41)$$

Avec 0<α<1 est un facteur de lissage. $\beta$ est l'énergie de l'erreur $\beta = \sum_{i=0}^{Z-1}|E(n)|^2$. E(n) est le vecteur d'erreur de filtrage et donné par : $\boldsymbol{E}(n)$=$[e(n), e(n-1), \ldots\ldots, e(n-z+1)]$. z est le nombre des points sur lesquelles nous calculons l'énergie de l'erreur. La mise à jour des coefficients du filtre adaptatif par l'algorithme NLMS amélioré est alors donnée par:

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \frac{\mu}{\alpha\boldsymbol{x}^T(n)\boldsymbol{x}(n)+(1-\alpha)\beta}e(n)\boldsymbol{x}(n) \quad (2.42)$$

### *f Stabilité de l'algorithme NLMS amélioré* [23]

L'erreur du signal:

$$e(n) = d(n) - \boldsymbol{w}^T(n)\boldsymbol{x}(n) \quad (2.43)$$

Et aussi appelée erreur *a priori* car elle utilise les coefficients du filtre avant la mise à jour. L'erreur "a postériori" est définie par:

$$è(n) = d(n) - \boldsymbol{w}^T(n+1)\boldsymbol{x}(n) \quad (2.44)$$

Et se calcule une fois que la mise à jour a été effectuée.

L'algorithme peut être considéré comme stable si la valeur absolue de l'erreur *aposteriori* est plus petite que celle de l'erreur *a priori*, ce qui est logique puisque e(n) exploite l'avantage d'informations.

En remplaçant l'équation du NLMSA:

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \frac{\mu}{\alpha \boldsymbol{x}^T(n)\boldsymbol{x}(n) + (1-\alpha)\beta} e(n)\boldsymbol{x}(n) \quad (2.45)$$

Dans l'erreur "a posteriori", on obtient:

$$è(n) = d(n) - \boldsymbol{w}^T(n+1)\boldsymbol{x}(n) \quad (2.46)$$

$$= d(n) - \boldsymbol{w}^T(n)\boldsymbol{x}(n) - \frac{\mu}{\alpha \boldsymbol{x}^T(n)\boldsymbol{x}(n) + (1-\alpha)\beta} e(n)\boldsymbol{x}^T(n)\boldsymbol{x}(n) \quad (2.47)$$

Puisque $\beta$ converge ver 0 on trouve :

$$è(n) = e(n)[1 - \frac{\mu}{\alpha}] \quad (2.48)$$

Donc

$$|è(n)| < |e(n)|$$

$$\left|e(n)\left[1 - \frac{\mu}{\alpha}\right]\right| < |e(n)|$$

$$\left|\left[1 - \frac{\mu}{\alpha}\right]\right| < 1$$

$$0 < \frac{\mu}{\alpha} < 2$$

$$0<\mu<2\alpha \quad (2.49)$$

Qui est la condition de stabilité de l'algorithme NLMSA.

- **Principe de fonctionnement du NLMSA en annulation d'écho acoustique** [23]

L'annulation d'écho acoustique requiert la connaissance d'un modèle pour le chemin d'écho acoustique identifié. Toutefois, la réponse impulsionnelle du chemin d'écho en

question peut varier en fonction du temps. Le filtrage adaptatif est l'approche la plus appropriée dans ce cas pour estimer les paramètres variables du chemin d'écho.

Le schéma suivant représente un système d'annulation d'écho dans un système de communication sonore (téléphone mains libres, téléconférence,…), où$x(n)$est le signal reçu du locuteur lointain, $d(n)$ est le signal d'écho du locuteur lointain vers lui-même. Lorsqu'un locuteur parle dans la pièce A, le haut-parleur de la pièce B émet le signal $x(n)$. Le microphone de la pièce B reçoit une version filtrée de $x(n)$. Etant directement relié au haut-parleur de la pièce A, le locuteur va s'entendre parler.

Pour éviter cela, nous estimons de manière adaptative le filtre $w_n$après la normalisation avec l'énergie de l'erreur et nous envoyons sur le haut-parleur de la pièce A uniquement l'erreur commise.

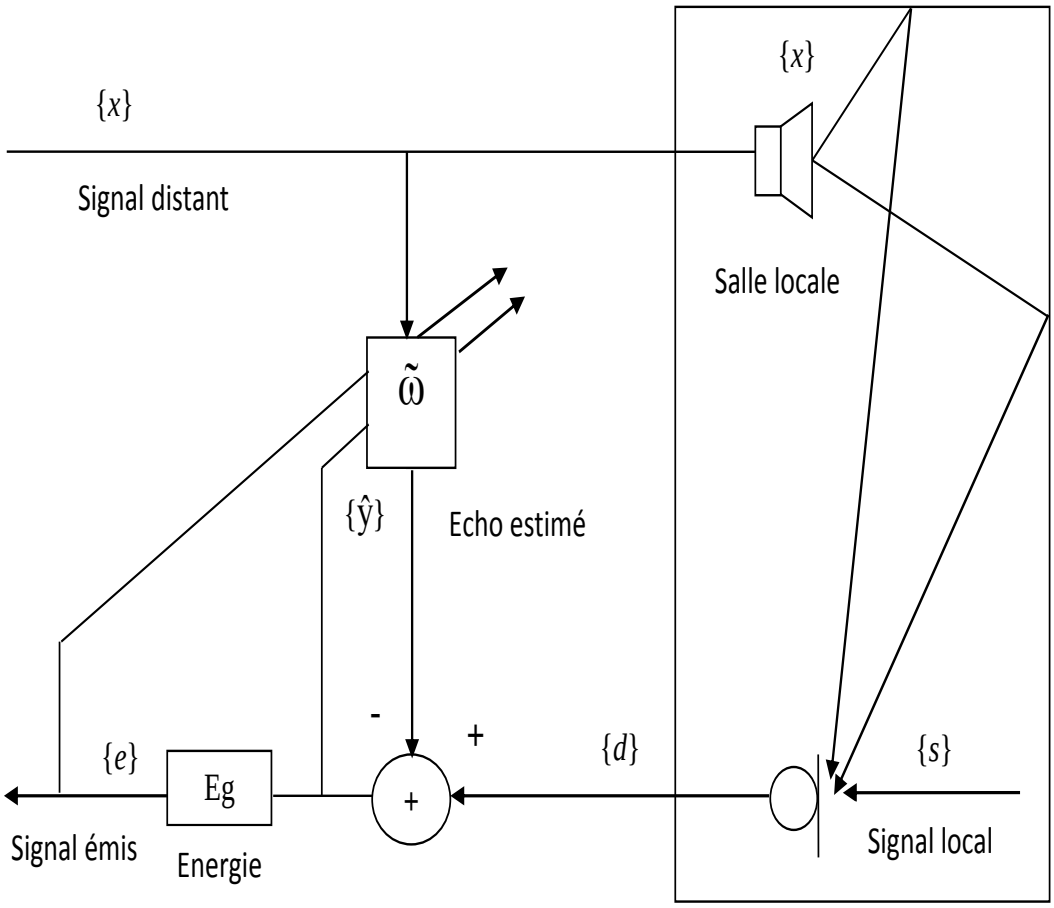


***Figure 2. 6*** Principe de l'annulation d'écho acoustique par ajout d'énergie [23].

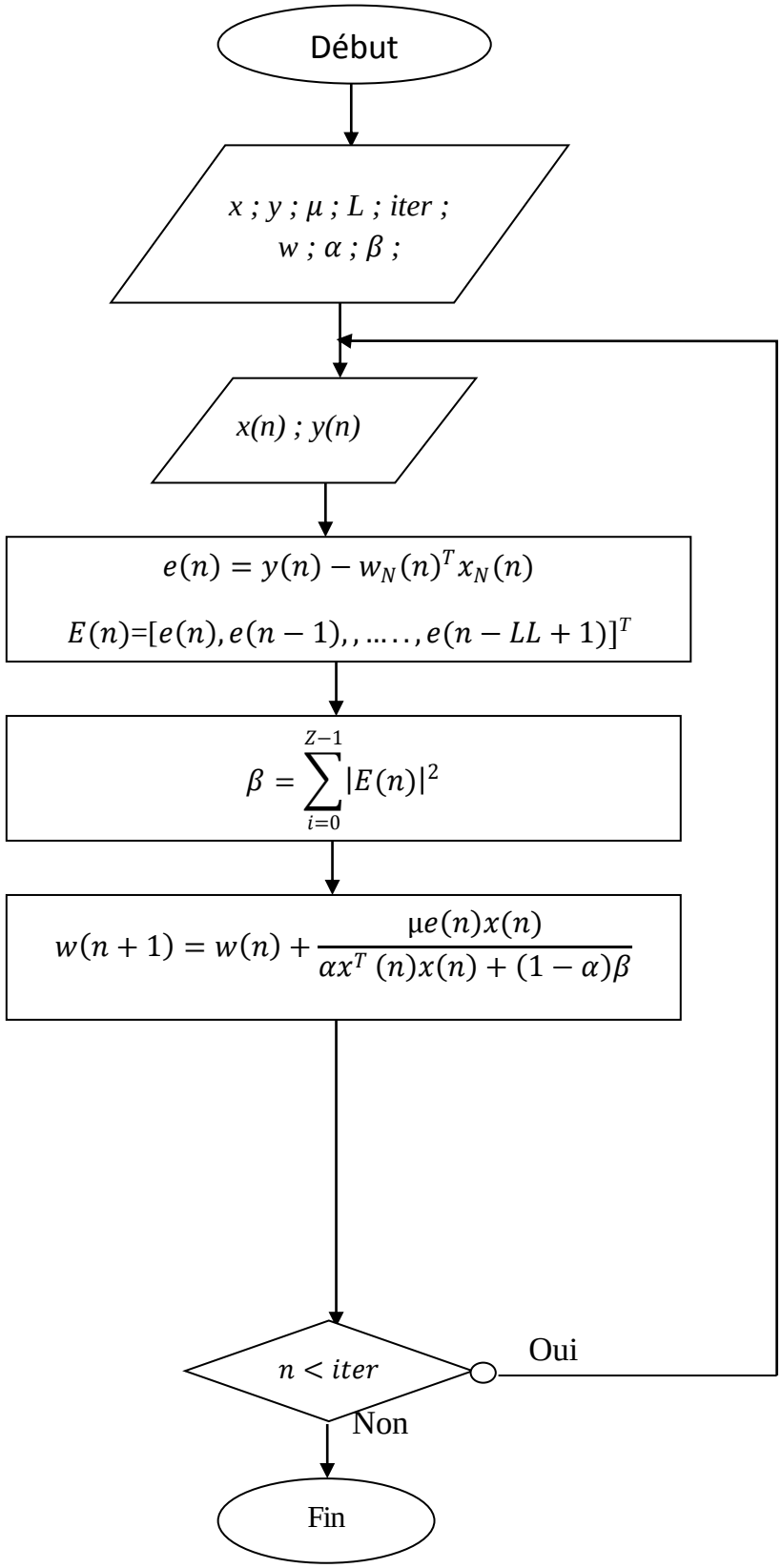


***Figure 2. 7*** L'organigramme du l'algorithme NLMS amélioré [23].

## 2.6 Conclusion

Dans ce chapitre nous avons défini le problème posé par la présence de phénomène d'écho acoustique, ensuite nous avons présentés les algorithmes adaptatifs LMS, NLMS et sa version améliorée qui permet d'améliorer le taux de convergence du filtre adaptatif par apport à l'algorithme NLMS.

Dans le prochain chapitre, nous allons présenter un descriptif du langage VHDL et son circuit FPGA.

# Chapitre 3 Principe du VHDL

## 3.1 Introduction

Avant la réalisation de ce projet, il faut d'abord commencer par étudier les notions de base du langage VHDL qu'on va l'utiliser pour implémenter un filtre RIF pour l'annulation d'écho acoustique, d'autre part nous allons présenter les outils de conception et simulation ainsi que les circuits FPGA.

## 3.2 Le langage VHDL

### 3.2.1 Définition

VHDL (le synonyme de langage de description matériel d'un circuit intégré à très haute vitesse), est un langage de programmation dont la vocation est de donner naissance à un circuit logique et non à un programme exécutable. Ce circuit peut se matérialiser dans un CPLD (Complex Programmable Logic Device) ou plus grand encore, dans un FPGA (Field ProgrammableGate Array).

### 3.2.2 Historique

Le langage VHDL a été commandé dans les années 1980 par le Département de la Défense des États-Unis dans le cadre de l'initiative VHSIC. Dans un effort de rationalisation, le VHDL reprend la même syntaxe que celle utilisée par le langage Ada (ce dernier étant aussi développé par le département de la défense).

La version initiale de VHDL, standard IEEE 1076-1987, incluait un large éventail de types de données, numériques (entiers, réels), logiques (bits, booléens), caractères, temps, plus les tableaux de bits et chaînes de caractères.

L'un des principaux problèmes concernait le type bit. Celui-ci ne pouvant prendre que deux valeurs (0, 1), il était impossible de représenter les signaux de valeur inconnue ou encore les signaux en haute impédance, ainsi que la « force » d'un signal (faible, forte ou nulle). La norme IEEE 1164 définit le type std_logic avec 9 états possibles. Ceci a été adopté dans le VHDL-93 (seconde version de la norme IEEE 1076).

Afin de répondre aux différents problèmes de l'électronique, la norme VHDL a dû évoluer. L'IEEE Design Automation Standards Committee (DASC) a créé la norme IEEE 1076.1(1999), ou VHDL-AMS (*VHDL-Analog and Mixed Systems*).

Cette nouvelle norme est une extension de la norme IEEE 1076-1987 déjà existante. Elle permet la description et la simulation de circuits analogiques, numériques, et mixtes (analogique et numérique). Pour cela elle utilise en complément des instructions séquentielles et concurrentes un nouveau type d'instructions, dites « simultanées », et qui ont valeur d'équations. En pratique, de plus en plus de simulateurs implémentent cette extension. Par contre, les outils de synthèse analogique associés n'en sont encore qu'à leurs balbutiements.

### 3.2.3 Les caractéristiques du VHDL

- Standard (indépendant du logiciel ⇒ échange facile)
- Haut niveau d'abstraction (indépendant de la technologie)
- Méthodologies de conception diverses
- Outil complet (design, simulation, synthèse)

### 3.2.4 Structure du langage VHDL

#### *a Déclaration des bibliothèques*

Toute description VHDL utilisée pour la synthèse a besoin de bibliothèques. Et plus particulièrement la bibliothèque IEEE 1164. Elle contient les définitions du type de signaux électroniques, fonctions et sous programmes permettant de réaliser des opérations arithmétiques et logiques. Les bibliothèques sont spécifiées par le mot clé library. Pour avoir Accès à une partie de la bibliothèque on utilise l'énoncé use. On a donc accès dans l'entité Incrémenter à tous les types définis dans le package

std_logic_1164 de la libraire IEEE. Cela nous permet entre autres d'utiliser le type std_logic [16].

```
Library ieee ;

Use ieee.std_logic_1164.all;

Use ieee.numeric_std.all ;

Use ieee.std_logic_unsigned .all;
```

***Tableau 3. 1*** Déclaration des bibliothèques.

### *b Entité*

Un bloque d'entité est le début de bloque de construction d'un VHDL, chaque conception a un seul bloque qui décrit l'interface des signaux dans et hors l'unité de conception. Une même entité peut être associée à plusieurs architectures différentes. Elle décrit alors une classe d'unités de conception qui présente au monde extérieur le même aspect, avec des fonctionnements internes éventuellement différents [16].

La syntaxe générale pour une déclaration d'entité est:

```
entity Nom_de _l_entite is

 port (Nom_de_signal, Nom_de_signal: type de mode

 Nom_de_signal, Nom_de_signal: type de mode);

end Nom_de _l_entite;
```

***Tableau 3. 2*** Déclaration d'entité.

- **Port:** l'instruction port est utilisé pour la déclaration des entrées et des sorties.

- **Le nom du signal:** est composé de caractères, le premier caractère doit être une lettre, sa longueur est quelconque, mais elle ne doit pas dépasser une ligne de code.
- **La direction du signal**

    - **in :** pour un signal en d'entrée.
    - **out :** pour un signal de sortie.
    - **inout :** pour un signal bidirectionnelle, un signal en entrée sortie.

- **Type**

La notion de type est très importante en VHDL, chaque entrée, sortie, signal, variable ou constante, est associé à un type. Sans artifice particulier, il n'est pas question d'effectuer une opération entre deux grandeurs de type différent.

- **Bit:** il s'agit là, évidemment, du type de base le plus utilisé en électronique numérique. Un objet de type bit peut prendre deux valeurs : '0' et '1', il s'agit, en fait, d'un type énuméré prédéfini.
- **Bit_vector:** un vecteur de bits est un bus déclaré par exemple pour une largeur de N bits par bit_vector (N downto 0) ou bien bit_ vector (0 to N) [16].

- **Generic:** est placée devant le port déclaration dans le bloque d'entité. Cette instruction est utilisée pour faciliter la modification ou la mise à jour de conceptions. Comme par exemple la largeur d'un bus. Le paramètre peut être initialisé directement à une certaine valeur, puis modifié ultérieurement lors le son utilisation.

### *c Architecture*

Une fois l'entité définie, il faut décrire son fonctionnement, c'est ce que l'on fait avec l'architecture. La syntaxe générale pour une déclaration d'architecture est:

```
Architecture RTL of HALFADD is

 -- Déclarations préalables

Begin SUM <= A xor B;

 CARRY <= A and B;

 End RTL
```

***Tableau 3. 3*** Déclaration d'architecture.

Une architecture fait toujours référence à une entité (ici, HALFADD). Elle est définie par un nom quelconque (ici RTL), que l'on pourra choisir pour expliciter la façon dont on code. A la suite de la déclaration de l'architecture, on définira les déclarations préalables (signaux internes, composants).

Ensuite, viens le mot clé « begin ». A sa suite, on trouvera le code qui décrit le fonctionnement de l'architecture. Dans une architecture, toutes les lignes combinatoires ou bloque de process sont exécutées en parallèle.

L'architecture peut être décrite de plusieurs façons:

- **Description structurelle**

C'est une description schématique, souvent utilisée au niveau le plus élevé de la hiérarchie, chaque composant étant lui-même défini par un programme VHDL [17].

- **Description comportementale**

Une description comportementale (behavioral) se présente comme des bloques d'algorithmes séquentiels exécutés par des processus indépendants. Elle peut traduire un fonctionnement séquentiel ou combinatoire du circuit modélisé [16].

- **Description flot de données (fonctionnelle)**

Ce mode de description permet de reproduire l'architecture logique en décrivant la fonction du circuit par des équations booléennes [16].

On donne, ci-dessous, quelques notes sur la syntaxe d'un programme VHDL [18]:

- Pas de différenciation entre majuscules et minuscules
- Format libre
- Toute phrase termine par un point-virgule
- Le début d'un commentaire est signalé par un double trait (“--”). Le commentaire termine avec la fin de ligne
- Un signal doit être défini par son type et sa taille.
- Tout objet doit être déclaré avant d’être utilisé.
- Les noms de signaux, labels, etc. … ne doivent contenir que des lettres, nombres et underscore. Ils doivent impérativement commencer par une lettre, et ne pas terminer par un underscore.

### 3.2.5 Types du langage VHDL

VHDL est un langage typé où il est obligatoire de spécifier le type des objets utilisés. Ses types prédéfinis sont [19]:

***a Scalaire***

- **Entier:** le type entier integer prédéfini dans le paquetage standard STD permet de définir des nombres signés sur 32 bits entre $-2^{-31}$ et $2^{31} - 1$.
- **Enuméré:** est un type défini par une énumération exhaustive par exemple :

```
type COULEURS is (ROUGE, JAUNE, BLEU, VERT, ORANGE);
```

***Tableau 3. 4*** Exemple d'une déclaration scalaire énuméré.

Dans le paquetage STANDARD de la bibliothèque STD, plusieurs types sont définie :

```
type boolean is (FALSE, TRUE);
type bit is ('0', '1');
type severity_level is (NOTE, WARNING, ERROR, FAILURE);
type character is ( NUL, SOH, STX, ETX,..., '0','1', ...);
```

***Tableau 3. 5*** Les types définis dans STD.

- **Physique:** VHDL permet de définir des types physiques pour représenter une grandeur physique, comme le temps, la tension, etc... Un type physique est la combinaison d'un type entier et d'un système d'unité.

- **Réel:** prédéfini REAL. Il permet de représenter des nombres entre -1.0E+38 à 1.0E+38. Voici quelques exemples d'affectation d'un signal de type réel :

```
A <= 1.0;
B <= 5.9E10;
C <= -8.5E20;
```

***Tableau 3. 6*** Exemples d'affectation d'un signal de type réel.

### *b Composite*

- **Tableau:** Les types TABLEAU ou array sont des collections d'objets de même type, indexés par des entiers ou des énumérés. Exemples :

```
type bus is array (0 to 31) of bit;
type RAM is array (0 to 1024, 0 to 31) of bit;
type PRIX is ranger 0 to 1000;
```

***Tableau 3. 7*** Exemple d'une déclaration d'un type tableau.

- **Enregistrement (RECORD):** ce type permet de définir un objet dont les composantes sont hétérogènes.

```
type clock_time is record hour : integer range 0 to 12 ;

minute , seconde : integer range 0 to 59 ;

end record ;

variable time_of_day : clock_time ;
```

***Tableau 3. 8*** Déclaration d'un type RECORD.

### *c Fichier et pointeur*

- **Fichier:** Les types fichiers FILE permet l'échange de données entre l'extérieur et le simulateur VHDL. Il est utilisé principalement pour créer des fichiers de test ou TESTBENCH de modèles. Un fichier peut être soit en lecture soit en écriture mais pas les 2 en même temps.
- **Pointeur:** sont peu utilisés en VHDL donc il est préférable d'utiliser les tableaux indicés qui peuvent être synthétisables.

## 3.2.6 Les opérateurs

Il y a sept classes d'opérateurs [20]:

- Logiques : and, or, nand, nor, xor, xnor .
- Relationnel : =, /=, <=, >, >= .
- Décalage : sll, srl, sla, sra, rol, ror (VHDL93).
- Addition : +, -, & (concaténation).
- Signe : +, - .
- Multiplication : *, /, mod, rem.
- Divers : **, abs, not.

### 3.2.7 Les instructions du VHDL

Comme tout langage de description de matériel, le VHDL décrit des structures par assemblage d'instructions concurrentes dont l'ordre d'écriture n'a aucune importance, contrairement aux instructions séquentielles qui sont exécutées les unes après les autres, comme c'est le cas du langage C.

#### *a Les instructions concurrentes*

Les instructions concurrentes sont les bases du langage VHDL, elles servent essentiellement à l'affectation. Le mode concurrent représente l'aspect combinatoire du fonctionnement des circuits. Ces instructions sont prédéfinies comme suit [20]:

- **Affectation simple**

  Y <= … oper_logic …;

*Exemple:*

```
sortie <= sel0 or sel1 or sel2;

signal <= vect_8bits(3);

vect(0) <= (A or B) and C;

signal <= entree;
```

***Tableau 3. 9*** Exemple d'une affection simple.

- **Affectation conditionnelle**

```
Y <=.. when .. else .. ⇔ signal1 <= expression_true when condition else
expression_false;
```

***Tableau 3. 10*** Syntaxe d'une affectation conditionnelle.

*Exemple :*

```
egal <= ‘1’ when (valeur="1001") else '0';

result <= A or B when (c='1') and (en='1') else '0';
```

***Tableau 3. 11*** Exemple d'une affectation conditionnelle.

Il existe aussi une syntaxe avec plusieurs conditions :

```
signal1 <= expression when cond_booleen else

expression when cond_booleen else

 ...

expression;
```

***Tableau 3. 12*** Syntaxe avec plusieurs conditions.

***Remarque:*** L'instruction **when…else** implique une notion de priorité entre les différentes conditions.

- **Affectation sélective**

```
with .... select Y <= …. when …. <=> with signal_commande select

signal_affecte <= expression1 when "com_etat1",

expression2 when "com_etat2",

...

expressionN when others;
```

***Tableau 3. 13*** Syntaxe d'une affectation sélective.

***Remarque:*** le terme **others** définit toutes les autres combinaisons possibles de l'état du signal de commande (! type std_logic !).

- **Instanciation de composants (port map)**

  Le mot component (composant) permet de déclarer un prototype(modèle) de composant, l'instanciation se fait alors dans le corps du l'architecture par exemple

```
architecture struct of exemple is

component porte_et is

port (A_i, B_i : in std_logic; Z_o : out std_logic);

end component;

for all : porte_et use

entity work.porte_et(flot_don);

begin

..
```

```
U1: porte_et  port map (A_i => entr_i,
                        B_i => signal_s,
                        Z_o => sortie_o);
..
end struct;
```

***Tableau 3. 14*** Exemple d'instanciation de composants.

- **Processus (process)**

  Tout modèle VHDL peut se décrire de manière équivalente comme un ensemble de processus communiquant par l'intermédiaire de signaux. Un processus encapsule une séquence d'instructions exécutées dans un ordre donné. L'exécution des instructions d'un processus est conditionnée par des événements (event) sur des signaux.

```
Nom : process (liste de sensibilité) Partie déclarative : variables begin
Corps du processus. Instructions séquentielles End process [Non] ;
```

***Tableau 3. 15*** Syntaxe du processus.

### *b Les instructions séquentielles*

Les instructions séquentielles sont internes aux processus, aux procédures et aux fonctions suivante [21]:

- **Instruction conditionnelle IF**

```
If condition then

Instructions

elsif condition

then instructions

else instructions

end if;
```

***Tableau 3. 16*** Syntaxe d'instruction conditionnelle *IF*.

- **Instruction sélective case**

```
case expression is when valeur1=>……sequence1;

                    when valeur2| valeur3=>……sequence2;

                    when valeur4 to valeur8=>……sequence3;

                    when valeur4 downto valeur8=>……sequence4;

end case;
```

***Tableau 3. 17*** Syntaxe d'instruction conditionnelle *case*.

- **Instruction de boucle**

LOOP: loop ……..end loop;

```
while condition loop …….;

end loop;

for……in 1 to ……. loop …….;

end loop;
```

***Tableau 3. 18*** Syntaxe d'instruction de boucle *loop*.

LOOP imbriquées:

```
first_loop : loop

second_loop : loop

……do something…….

end loop second_loop;

end loop first_loop;
```

***Tableau 3. 19*** Syntaxe d'instruction de boucle *loop imbriqué*.

**Remarque:** l'instruction NEXT arrête l'itération en cours de boucle, l'instruction EXIT permet de sortir d'une boucle

- **Instruction WAIT**

  wait on ….. until …for…..;

  *Exemple:*

```
wait on clock;

wait on clock until data ='1';

wait on clock until data ='1' for 10ns;

wait for 10ns;

wait;
```

***Tableau 3. 20*** Exemple d'instruction *wait*.

## 3.3 Quartus

**Définition:** Quartus est un logiciel développé par la société Altera, permettant la gestion complète d'un flot de conception CPLD ou FPGA. Ce logiciel permet de faire une saisie graphique ou une description HDL (VHDL ou verilog) d'architecture numérique, d'en réaliser une simulation, une synthèse et une implémentation sur cible reprogrammable.

### 3.3.1 Création d'un projet

Au début cliquez sur Démarrer —Programmes —>Altera —>Quartus II 5.1 pour lancer le programme

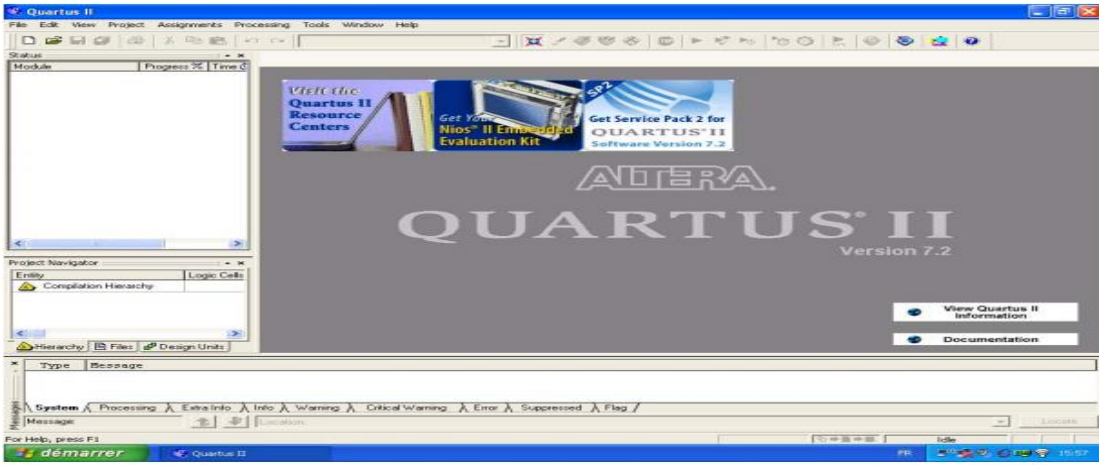


***Figure 3. 1*** Fenêtre de création d'un projet.

Après avoir lancé le programme , faites **File —>New Project Wizard** . Une nouvelle fenêtre apparaît qui permettant de configurer le projet

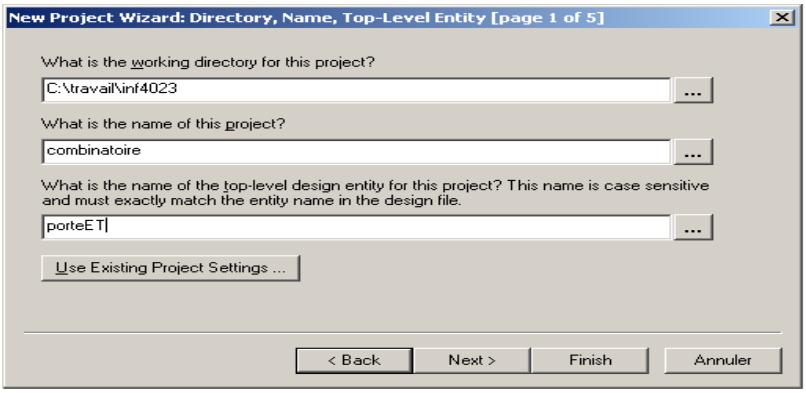


***Figure 3. 2*** Fenêtre de configuration du projet.

Choisir un nom représentatif pour votre projet et spécifier un répertoire où le projet sera sauvegardé. Il faut aussi définie le nom de l'entité maître du projet (On appelle entité maître, le design qui correspond à la couche hiérarchique la plus haute Design).

Cliquer sur **Next** la fenêtre **Add Files** apparaît elle nous propose d'inclure des fichiers déjà crées au projet. recliquez sur **Next**.

Dans la fenêtre suivante intitulée Family & Device Settings, Cocher la case « **SPECIFIC DEVICE SELECTED IN 'AVAILABLE LIST'** »,Choisir la famille du composant programmable ainsi que le circuit cible et cliquer sur **Next**.

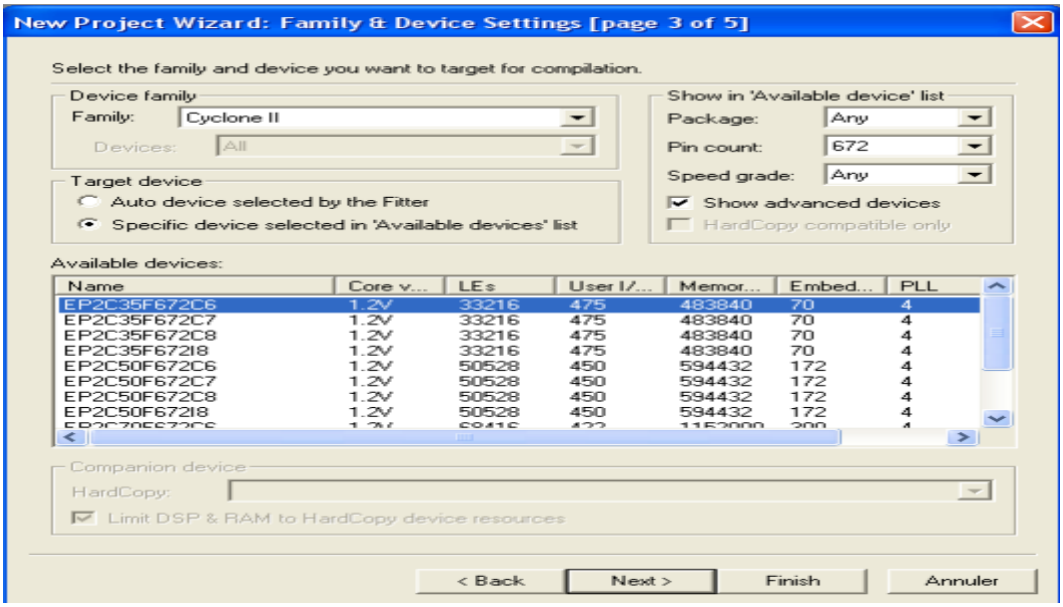


***Figure 3. 3*** Fenêtre Family & Device Settings.

***Remarque***

Il est également possible de ne pas choisir de composant en cochant la case "AUTO DEVICE SELECTED BY THE FITTER", ce qui signifie que la cible optimale sera choisie par le logiciel une fois le design saisi.

Quand la fenêtre **EDA Tool** Settings apparaît cliquer sur **Next**. Une fenêtre récapitulative apparaît:

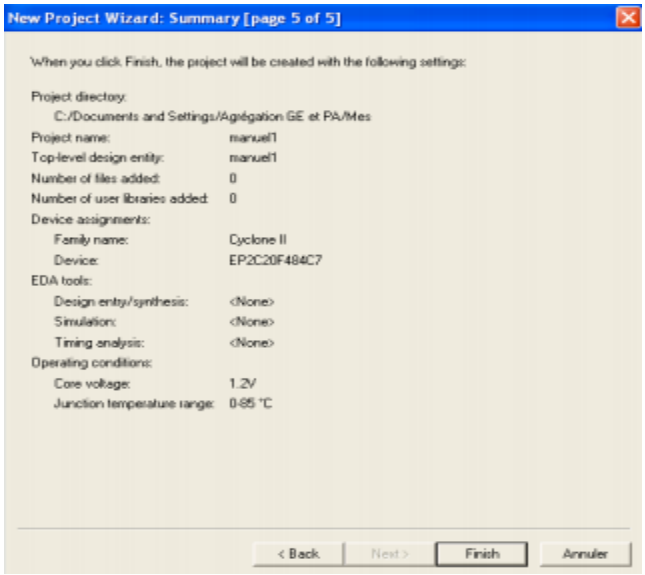


***Figure 3. 4*** Fenêtre de récapitulation.

Valider les choix par Finish ou bien faire Back pour des modifications éventuelles. Dans le navigateur de Projet, un onglet avec le type composant et l'entité maître apparaît:

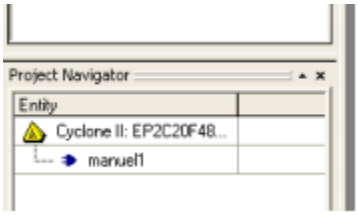


***Figure 3. 5*** Création du projet dans le navigateur de projet.

Le projet est maintenant crée.

### 3.3.2 Saisie d'un projet

Cette étape permet de définir et configurer les différentes parties du projet. Quartus accepte plusieurs types de saisie à savoir:

- Une saisie graphique en assemblant des symboles.
- Une saisie textuelle à l'aide de différents langages (VHDL, Verilog, AHDL...).

### 3.3.3 Saisie graphique

Pour saisir un projet en mode graphique, aller dans le menu : File —>New La fenêtre suivante apparaît:

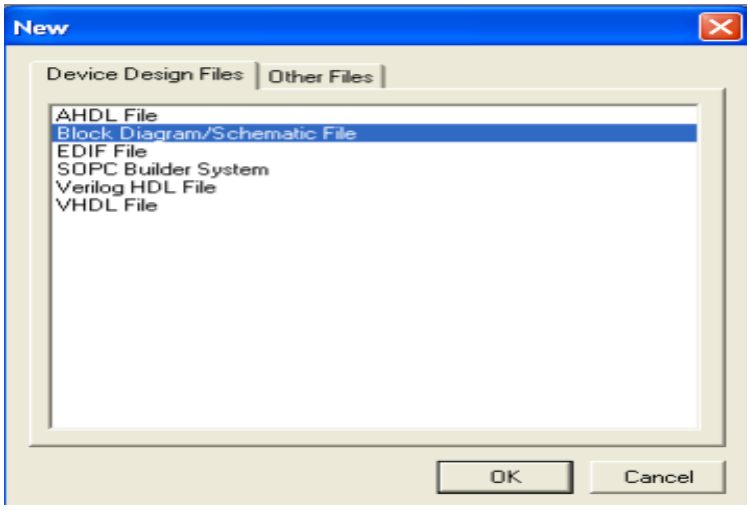


***Figure 3. 6*** Saisie un projet en mode graphique.

Sélectionner le type **Block diagram / Schematic** file puis faite **OK**.

Une feuille blanche se crée intitulée Block1.bdf. On prendra soin de sauver cette feuille sous le nom de l'entité maître C'est maintenant cette feuille de saisie graphique qui a la hiérarchie la plus haute dans le projet.

Il convient maintenant d'insérer des symboles dans notre feuille. Pour cela, nous pouvons soit choisir des composants de la librairie Altera soit en créer en les décrivant en VHD.

L'insertion d'un symbole se fait en cliquant dans la feuille avec le bouton de droite et en allant dans Insert —>Symbol. La fenêtre suivante s'ouvre:

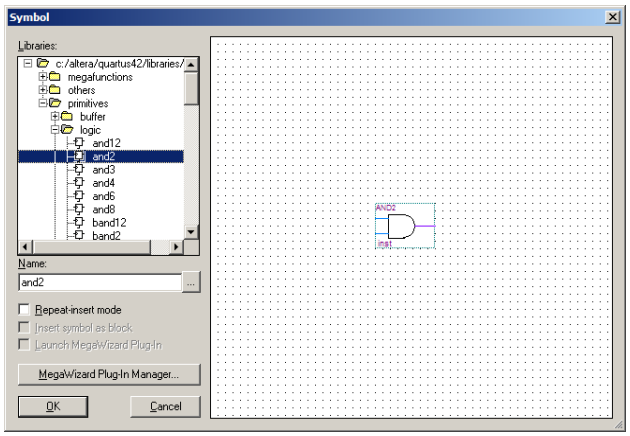


***Figure 3. 7*** Insertion d'un symbole.

### 3.3.4 Création d'un fichier VHDL

La saisie d'un composant VHDL se fait de la même manière que précédemment. Pour cela, aller dans le menu : File —>New La fenêtre suivante apparaît:

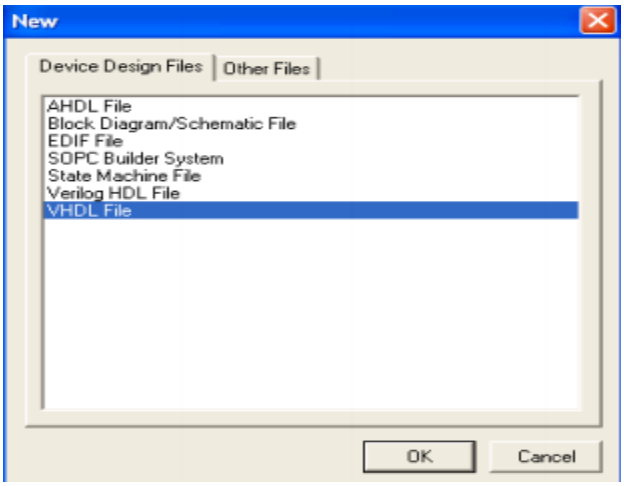


***Figure 3. 8*** Création d'un fichier VHDL.

Sélectionner le type **VHDL File** puis faite **OK**,Un petit éditeur de texte apparaît.

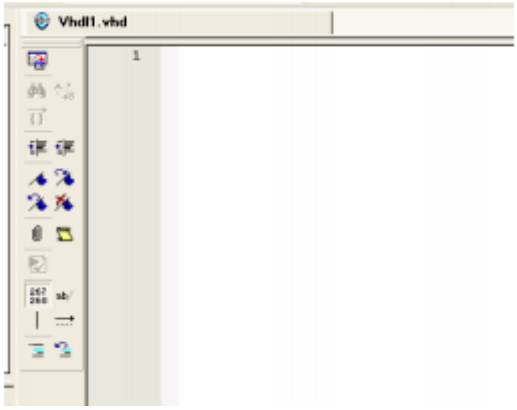


***Figure 3. 9*** Editeur de texte.

Une fois le code VHDL saisie, il convient de le sauver (**File → puis Save As**) puis d'en vérifier la bonne syntaxe de la description en cliquant sur **Analyse → Current File** et Corriger les éventuelles erreurs.

Lorsque l'édition du fichier est terminée et qu'il est sauvegardé, vérifier la bonne syntaxe de la description en cliquant sur Analyse Current File.

***Remarque:*** il est important de sauver le fichier sous le même nom que l'entité. Bien que cela ne soit pas indispensable comme sous MaxplusII, cela évite des intersections d'entité entre fichiers.

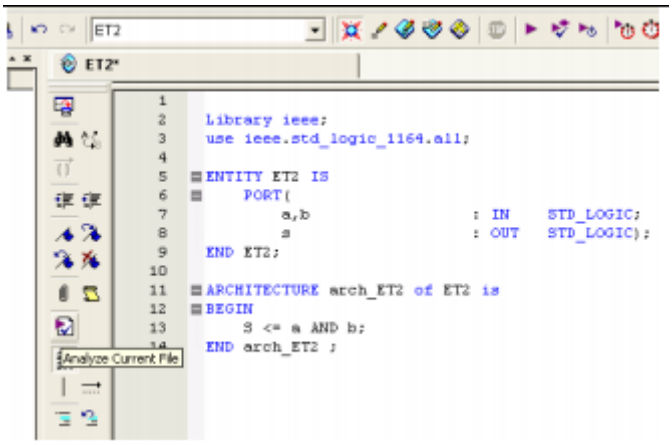


***Figure 3. 10*** Vérification de syntaxe de description.

Une fois que le fichier est **OK**, on peut alors créer un symbole graphique qui nous permettra d'insérer le composant dans la feuille graphique initiale. Pour cela, aller dans

**File →Create/Update →Create Symbol File from Current File**

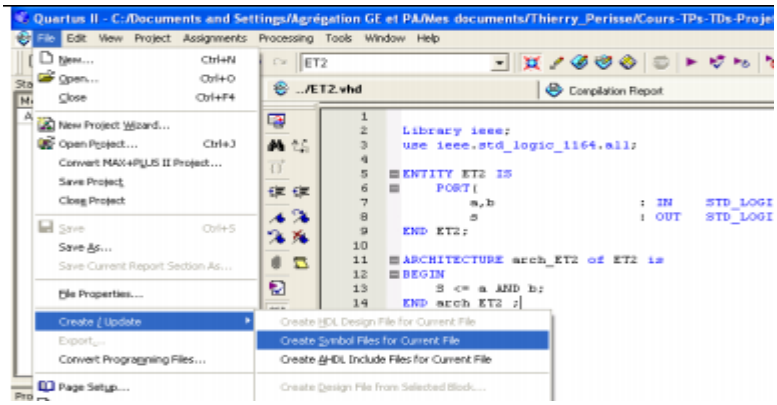


***Figure 3. 11*** Création d' un symbole graphique.

### 3.3.5 Compilation

Cette étape consiste maintenant à compiler le schéma précédemment réalisé.

Pour lancer la compilation, cliquer sur : **Processing →Compiler Tool**. la fenêtre suivante apparait:

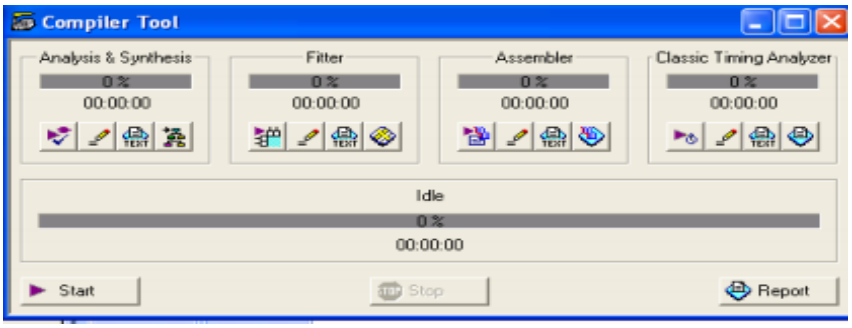


***Figure 3. 12*** Fenêtre de compiler Tool.

Cliquer sur **Start**. Une fois la compilation est terminée vous pouvez consulter le rapport en cliquant sur l'icône. Il faut à ce moment corriger les erreurs s'il y a lieu.

Appuyer sur **Tools →RTL Viewer** pour voir comment le schéma contenant le code VHDL a été transformé en portes et bascules. Cela permet de voir comment la synthèse logique s'est déroulée.

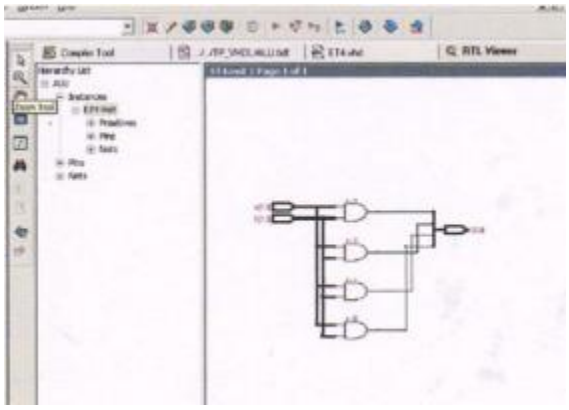

***Figure 3. 13*** Synthèse logique.

Utiliser la commande **Tools →Technology Map Viewer** pour visualiser la synthèse physique.

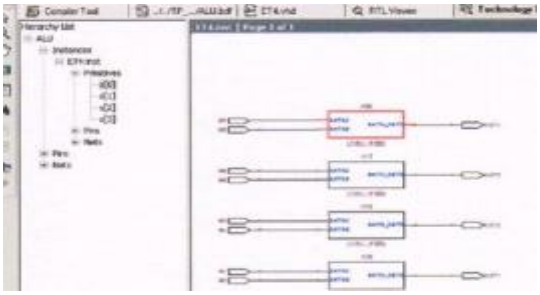

***Figure 3. 14*** Synthèse physique.

### 3.3.6 La simulation du projet

La simulation du projet permet de visualiser à l'écran le comportement de la fonction créée avant de programmer le composant. Cette visualisation se fait sous forme de chronogramme représentant l'évolution des entrées et sorties de la fonction logique simulée.

La simulation nécessite que l'utilisateur définisse l'allure des signaux d'entrée de la fonction, le simulateur se charge quant à lui de calculer l'évolution des signaux de sortie correspondants.

La première étape consiste à définir les signaux à appliquer sur les entrées du circuit

Cliquer sur **File** puis sur **New** Sélectionner l'onglet **"Other Files"** et cliquer sur **Vector Waveform File.**

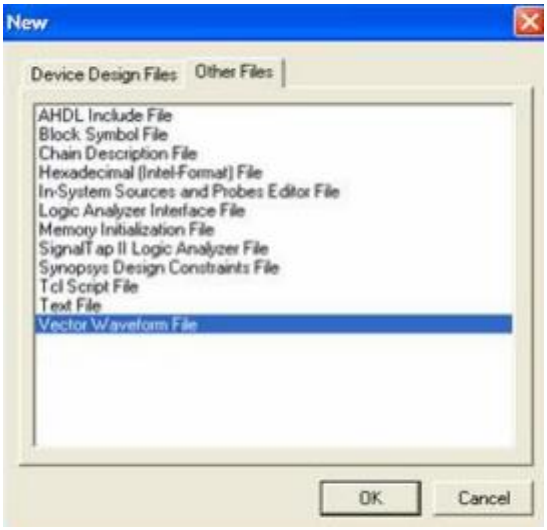


***Figure 3. 15*** Sélection du Vector Waveform File.

Pour modifier la durée de simulation, cliquer sur **Edit**, puis **End Time**. Une fois la durée modifiée, cliquer sur **OK**.


***Figure 3. 16*** Fenêtre de modification de la durée de simulation.

Sauvegarder le fichier sous son nom définitif avec son extension (l'extension. wvf «waveform vector file ») en cliquant sur **File** puis **Save As**.

Pour insérer les signaux d'entrée et les sorties, dans la fenêtre **"Name"**cliquer avec le bouton droit de la souris, puis sélectionner **Insert** et cliquer sur **Insert Node or Bus.** La boite de dialogue suivante qui s'ouvre:

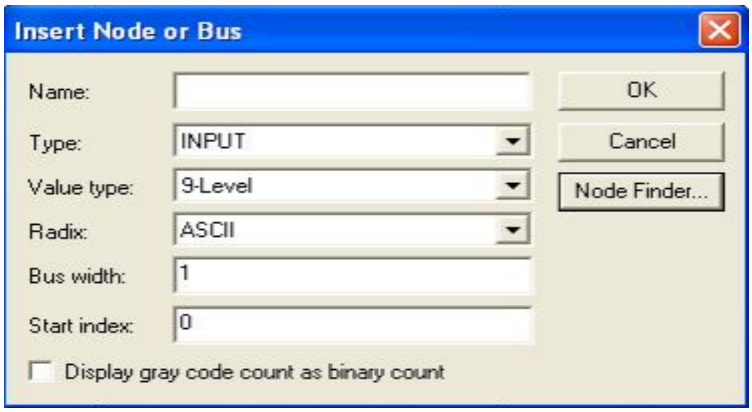


***Figure 3. 17*** Insertion des signaux d'entrées et les sorties.

Cliquer sur **Node Finder,** la fenêtre suivante apparait:

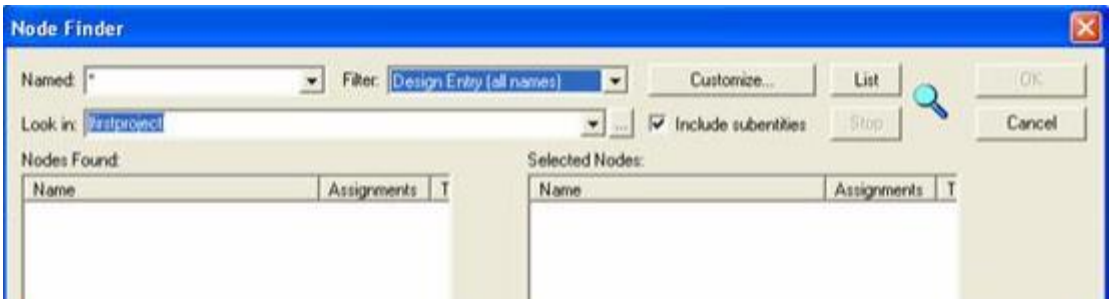


***Figure 3. 18*** Fenêtre de Node Finder.

Dans la catégorie « filter » choisir **all Name,** cliquer sur **List** afin apparaitre les signaux du design, dans la fenêtre **selected Nodes** sélectionner les signaux voulus puis cliquer sur la flèche correspondante [22].

Appuyer sur **OK pour** fermer les déférentes fenêtres et revenir à l'éditeur de signaux.

Afin de donner des valeurs de stimuli, On Clique avec le bouton droit de la souris sur le nom d'un signal, sélectionner **Value**, puis choisir la valeur du signal dans le menu.

Il est possible d'effectuer la même opération sur une partie seulement d'un signal en sélectionnant une zone dans la partie "chronogramme". Il faut pour cela maintenir le bouton gauche de la souris appuyé en déplaçant le curseur [22].

En dernier lieu, il ne reste plus qu'à lancer la simulation par la commande :

**Tools →Simulator Tool** dans la fenêtre de simulation qui s'ouvrira, spécifier le fichier de simulation et choisir le mode de simulation (Fonctionnelle ou Temporelle) puis cliquer sur le

bouton **Start** pour lancer la simulation, il est possible de voir le résultat en cliquant sur **Report**.

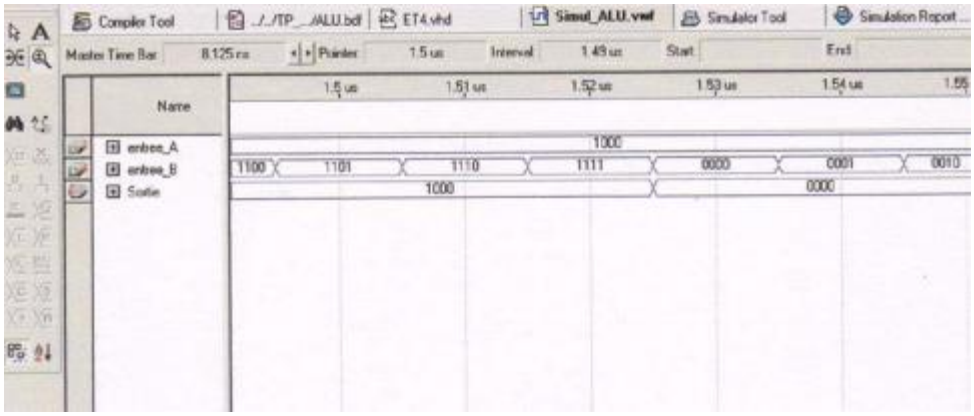


***Figure 3. 19*** Résultat de simulation.

Afin de permettre le rafraîchissement automatique du visualiseur de signal, il suffit de modifier une option en cliquant avec le bouton de droite sur **l'entité maître** dans le Project Navigator, puis en choisissant le menu **Settings**. Dans la nouvelle fenêtre aller dans l'onglet **Simulator Settings** et cocher la case **Overwrite simulation input file with simulation results**.

### 3.3.7 Programmation d'un circuit

- **Affectation des pins**

Afin de choisir quelle broche physique du circuit doit être connectée, lancer l'outil d'assignement de pins par : **Assignements →Pins**.

On double-clique sur la colonne **location** au niveau de la pin voulue de manière à faire apparaître un menu déroulant où sont répertoriées les broches disponibles du circuit.

- **Programmation du circuit**

Vérifier que les connections entre le PC (port parallèle) et la carte via le module ByteBlaster sont opérationnelles. Si tout est ok lancer le programmateur Cliquer sur **Tools** puis sur **Programmer.**

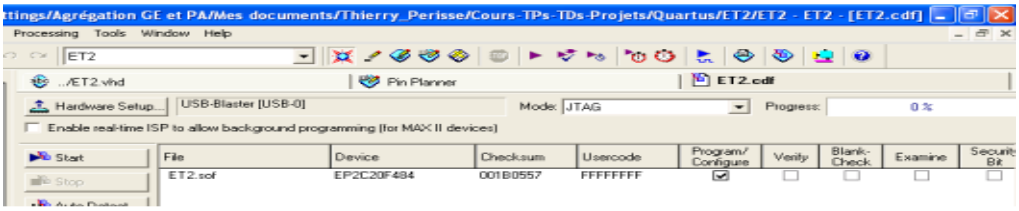


***Figure 3. 20*** lancement du programmateur du circuit.

Vérifier que le fichier avec l'extension **.sof** est bien là (sélectionner le) et que la case **Program/Configure** est cochée, puis cliquer sur **Start** [22].

## 3.4 ModelSim

Le ModelSim est un environnement de simulation de description de circuits numériques. Il intègre un gestionnaire de projet, un éditeur de texte et un visualiseur de chronogrammes qui permet de visualiser le résultat de simulation. Le cœur du logiciel, le simulateur logique, consomme une description VHDL multi-fichiers et un vecteur de test. Il produit un résultat de simulation qui est automatiquement affiché à l'écran par le visualiseur de chronogrammes [16]**.**

### 3.4.1 Lancement des outils

Après avoir lancé le programme de simulation (démarrer/programme) Il faut ensuite cliquer sur "Create a Project" dans la fenêtre Welcome. Ensuite choisir un répertoire de travail, un nom de projet et laisser le nom de bibliothèque work, avant d'effectuer une simulation, il est nécessaire de créer plusieurs fichiers de test (testbench).

### 3.4.2 Création des fichiers

Pour simuler un circuit, il faut 3 fichiers **.vhd**

- Un fichier de code.
- Un fichier de simulation (testbench).
- Un fichier de configuration.

### 3.4.3 Analyse des fichiers

Il faut toujours analyser avant de tester le circuit, donc il faut ajouter des fichier VHDL à compiler en suivant les étapes suivantes :

- File->Add to project-> « nom du module ».vhd
- Répéter l'opération pour les deux autres fichiers vhdl : sim_ « nom du module ».vhd et cfg_ « nom du module ».vhd
- Précisez l'ordre de compilation des fichiers grâce à la commande Compile->Compile Order
- Project->Compile All

On peut aussi compiler les fichiers d'une façon pas à pas,un éditeur de texte s'ouvre lorsque l'on double-clique sur les fichiers .vhd.

### 3.4.4 Lancement de la simulation

Pour lancer la simulation il faut choisir l'onglet Library, puis, en double-cliquant sur la configuration « nom du module » _cfg, la simulation se charge. Ceci est équivalent à :

Simulate->Start Simulation. Choisir ensuite work. « nom du module »_cfg.

Après dans le fenêtre "Objects", sélectionnez l'ensemble des signaux et « tirez » les dans la fenêtre « Wave » ensuite cliquer sur Simulate -> Run->Run 100ns permet de faire avancer la simulation de 100 ns (ou utilisez les icônes).

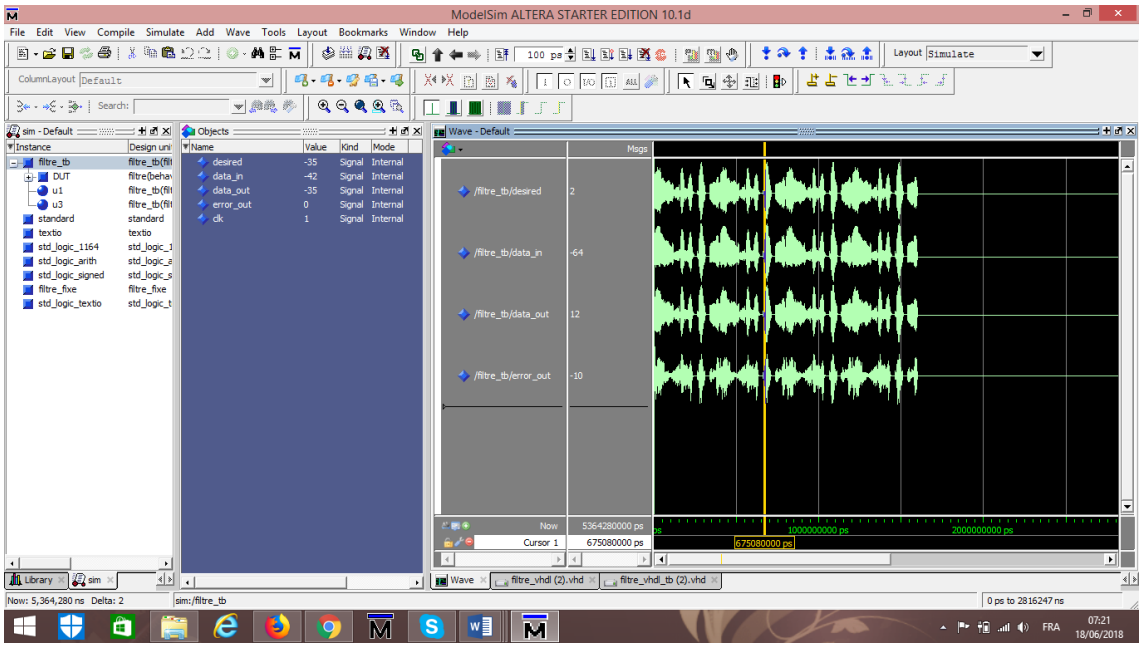


***Figure 3. 21*** Fenêtre de simulation en ModelSim avec plusieurs d'autres fenêtres.

L'écran est composé de plusieurs parties:

- View/Debug Window/All permet de faire apparaître l'ensemble des fenêtres de simulation.
- La fenêtre WORK SPACE (à gauche) avec ses onglets, qui permet d'accéder aux fichiers, aux entités compilées dans les biblitohèques, etc.
- Le code vhdl
- Les fenêtres "Active Process", "Locals", "Objects" et "Dataflow" permettent de sélectionner les signaux à visulaiser.
- La fenêtre "wave" dessine les chronogrammes

## 3.5 Circuit FPGA

### 3.5.1 Définition

FPGA (Field Programmable Gate Arrays) est un circuit intégré qui peut être personnalisé pour une application spécifique. Il est « programmable sur le terrain », ce qui signifie qu'il peut être configuré par l'utilisateur après la fabrication.

Les FPGA contiennent un réseau de cellules programmables. Chaque cellule est capable de réaliser une fonction, choisie parmi plusieurs possibles. Les interconnexions sont également programmables.

### 3.5.2 Architecture

L'architecture FPGA consiste en trois types d'éléments configurables:

- Un réseau de bloques logiques programmable qui peuvent être câblés dans différentes configurations. Ces blocs créent un tableau physique de portes logiques pouvant être utilisées pour effectuer différentes opérations et comportant des éléments à mémoire;
- Un réseau d'interconnexions programmables entre les bloques.
- Des bloques spéciaux d'entrée et de sortie avec le monde extérieur (*Input/Output Block – IOB*).

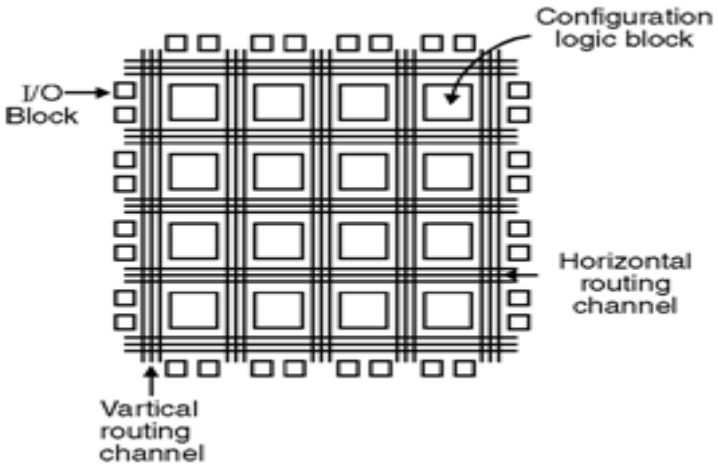


**Figure 3. 22** Modèle d'un FPGA.

## 3.6 Conclusion

Dans ce chapitre, nous avons présenté les étapes et les techniques d'implémentation d'une application sous le logiciel ModelSim/Quartus. Ce chapitre nous permettre de présenter et implémenter les filtres numériques sous VHDL.

Les résultats de cette implémentation seront présentés dans le chapitre suivant.

# Chapitre 4 Résultats des simulations

## 4.1 Introduction

Dans ce chapitre nous allons présenter les résultats de l'annulation d'écho acoustique, en utilisant l'algorithme du gradient stochastique normalisé (NLMS) et l'algorithme NLMS amélioré (NLMA). Deux types de résultats expérimentaux vont être exposés dans ce chapitre. Le premier type de résultats concerne les expérimentations faites sur le logiciel Matlab, le deuxième type concerne les résultats obtenus par l'implémentation de l'algorithme NLMS amélioré sous VHDL en utilisant le logiciel ModelSim.

Dans ce qui suit, nous allons tout d'abord présenté les signaux de test que nous avons utilisé dans les deux types de simulations décrites précédemment, puis les résultats obtenus avec les deux types d'implémentations seront détaillés.

## 4.2 Description des signaux de test

Les signaux utilisés dans les simulations que nous allons détaillées dans ce chapitre sont:

- Bruit USASI (United States of America Standards Institute) : C'est un bruit stationnaire qui a un spectre similaire au spectre moyen de la parole, et qui est donné par la figure (4.1), il est souvent utilisé comme signal de test dans les applications d'annulation d'écho acoustique pour évaluer la vitesse de convergence des algorithmes adaptatifs.
- Un signal de parole de 8kHz, qui est une phrase prononcée par un locuteur masculin qui dit « un loup s'est jeté immédiatement sur la petite chèvre » (voir la figure (4.2)).

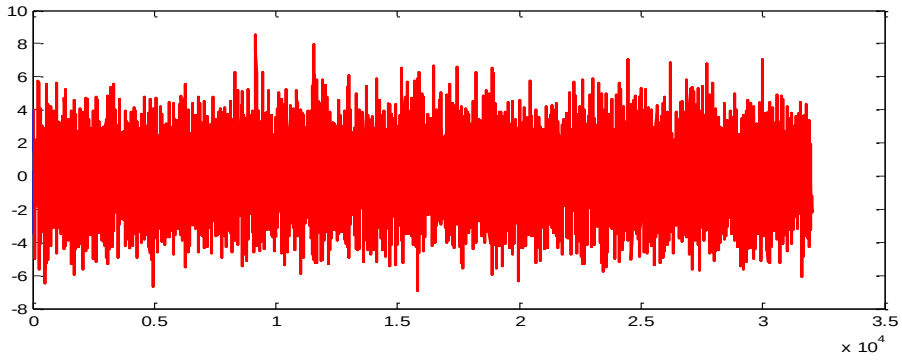


***Figure 4. 1*** Bruit BUSASI, Fréquence d'échantillonnage est de 8 kHz.

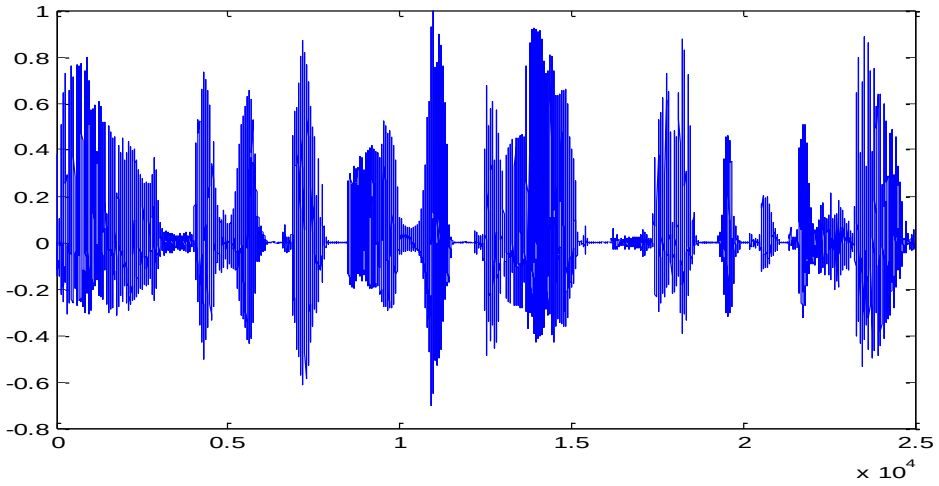


***Figure 4. 2*** Signal de parole, fréquence d'échantillonnage est de 8 kHz.

## 4.3 Description des critères de performances

Le critère de performance couramment utilisé en annulation d'écho est celui de l'évolution temporelle de l'MSE (mean square error) ce critère est donné par :

$$MSE = 10log\ (\sigma_e^2(n)) \tag{4.1}$$

Ou : $\sigma_e^2(n)$ l'énergie moyenne d'une suite temporelle de M échantillon consécutifs de l'erreur de filtrage$e(n)$donnée par la formule suivante :

$$\sigma_e^2(n)=\frac{1}{M}\sum_{i=1}^{M} e^2\ (n) \tag{4.2}$$

MSE est l'erreur quadratique moyenne exprimé en dB.

Le deuxième critère est l'ERLE (Echo Return Loss Enhancement) donnée par la relation suivante :

$$ERLE_{dB} = 10log_{10}(\frac{\sigma_y^2(n)}{\sigma_e^2(n)}) \quad (4.3)$$

Avec $\sigma_y^2(n) = \frac{1}{M}\sum_{i=1}^{M} y^2\,(n)$ (4.4)

Qui représente la variance de l'écho.

## 4.4 Résultat de simulation de l'algorithme NLMS amélioré obtenus en utilisant le logiciel Matlab

Dans ce contexte, nous allons présenter les résultats de simulation de l'algorithme NLMS et NLMSA sous Matlab appliqué. Nou avons utilisé les deux critères de performances à savoir MSE et ERLE et qui sont évalués avec le signal BUSASI et le signal de la parole comme signaux d'entrées.

### 4.4.1 Test avec le signal BUSASI sous Matlab

En utilisant le signal BUSASI, nous avons fait deux tests qui sont basés sur la taille du filtre *L* et le pas d'adaptation μ, ces derniers seront bien exposés par la suite:

#### *a Effet de la taille de filtre*

Les résultats de cette simulation sur l'effet de la taille du filtre sont obtenus en sélectionnant plusieurs tailles du filtre et en calculant l'erreur quadratique finale (MSE). Ces résultats obtenus sont donnés à la figure 4.3 suivante:

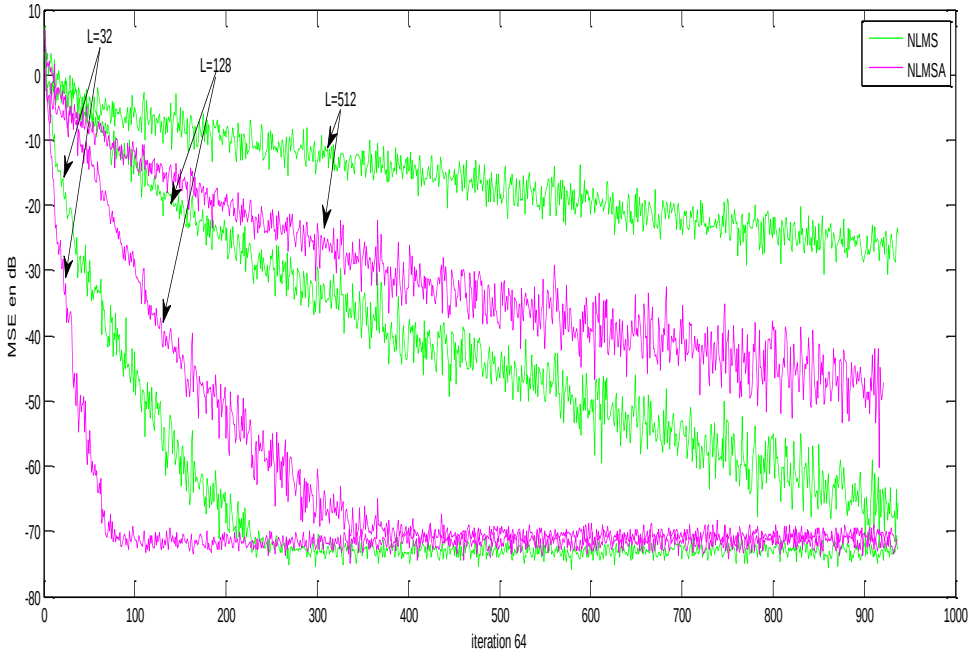


***Figure 4. 3*** Influence de la taille L sur le NLMS et NLMSA pour L= {32,128,512), µ=0.4, RSB=90 et un signal d'entrée BUSASI.

Selon la figure (4.3), nous remarquons bien que les deux algorithmes convergent mieux lorsque la taille du filtre adaptatif L est faible, ce qui montre que leur vitesse de convergence est inversement propostionnelle à la taille du filtre adaptatif, et nous observons une grande amélioration du convergence de notre algorithme proposé.

### *b Effet du pas d'adaptation*

Dans cette simulation, nous avons fixé la taille du filtre transverse L à 32 et nous avons fait varier le pas µ. Nous avons obtenu les résultats qui sont donnés àla figure(4.4) suivante.

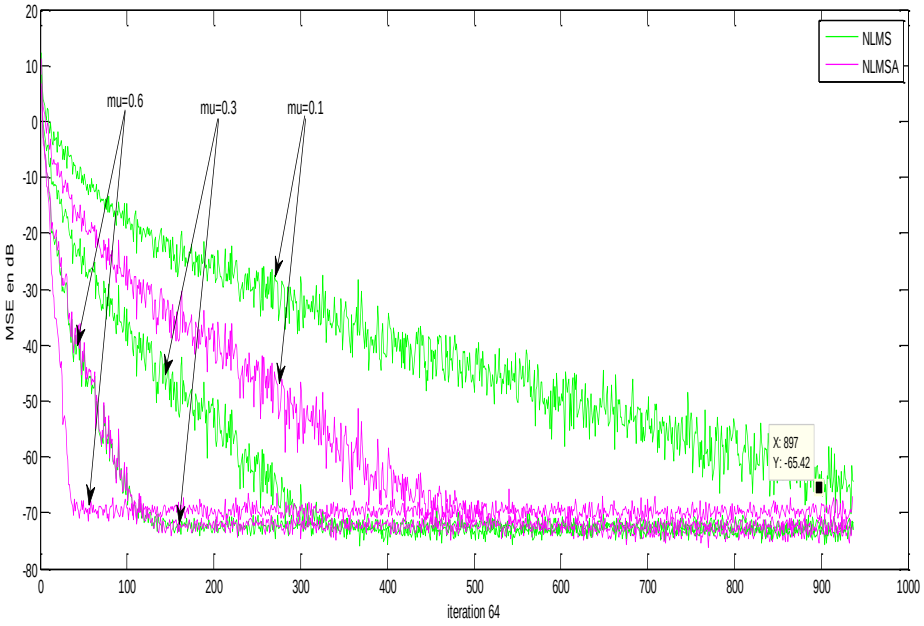


***Figure 4. 4*** Influence du pas d'adaptation pour NLMS et NLMSA μ={0.1,0.3,0.6), L=256, RSB d'entrée égale à 90 et un signal d'entrée BUSASI.

En se basant sur la figure (4.4), nous remarquons que les deux algorithmes convergent mieux lorsque le pas d'adaptation du filtre adaptatif μ est grand (μ = 0.6, 0.3 et 0.1 pour les deux algorithmes, et nous observons une grande amélioration de convergence de notre algorithme proposé en comparsion avec celle du NLMS.

### 4.4.2 Test dans un contexte d'annulation d'écho acoustique

**Signal d'entrée est de la parole**

Pour bien voir les performances de notre méthode dans un contexte réel d'AEA (annulation d'écho acoustique), nous avons réalisé une expérience en utilisant comme signal d'entrée de la parole réelle. Cette expérience dessous est réalisée avec une taille du filtre *L*=32, un pas d'adaptation μ=0.1 et un rapport signal à bruit d'entrée RSB=90 dB sur 160000 itérations. Les résultats obtenus sur le critère MSE et ERLE sont donnés respectivement par la figure (4.5) et (4.6) :

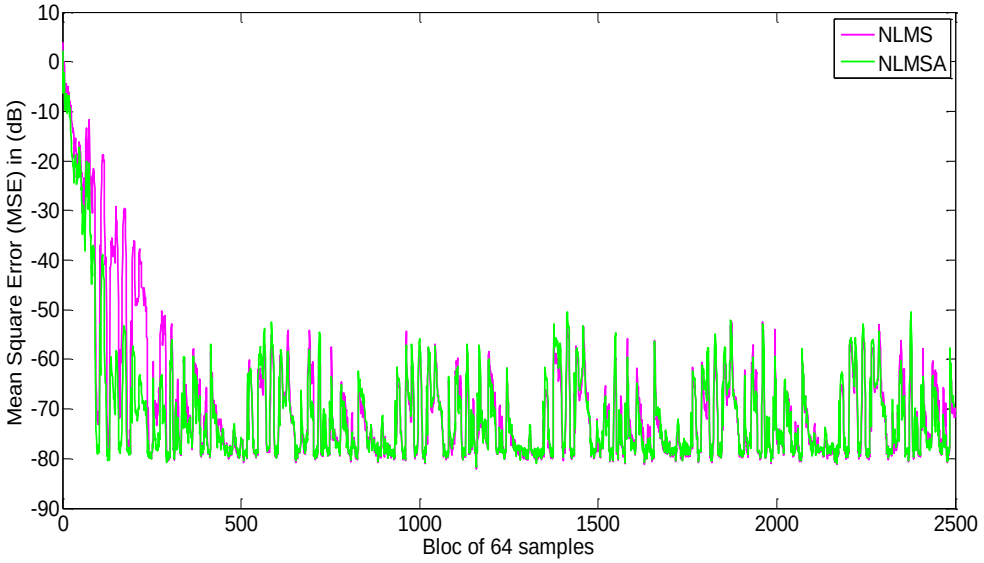


***Figure 4. 5*** Comparaison de la convergence de MSE pour l'algorithme NLMS et NLMSA. L=32. RSB d'entrée égale à 90dB. Signal d'entrée est de la parole.

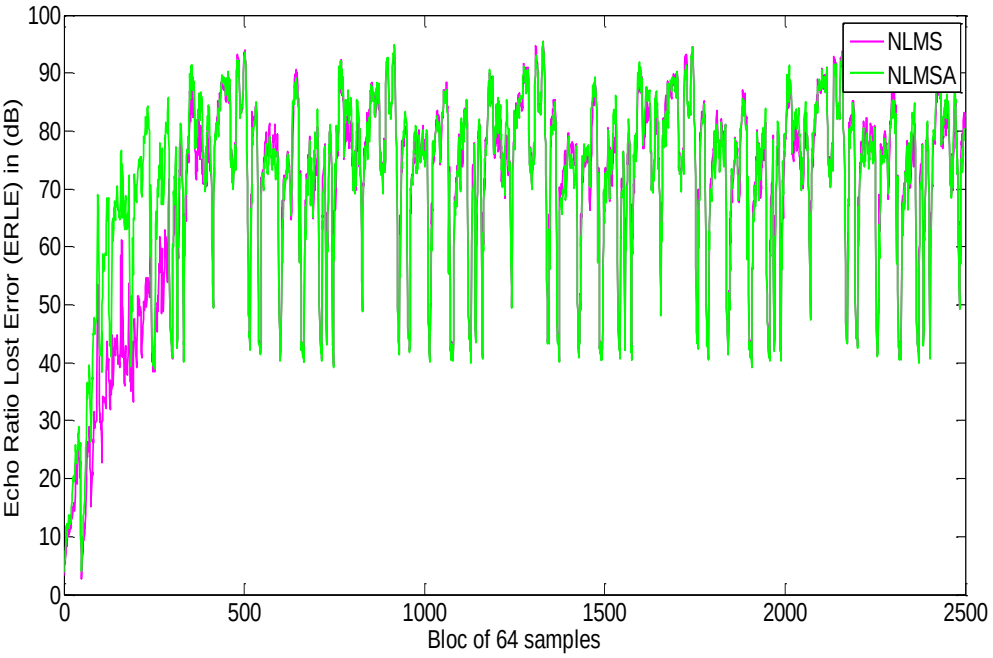


***Figure 4. 6*** Comparaison de la convergence de ERLE pour l'algorithme NLMS et NLMSA. L=32. RSB d'entrée égale à 90dB. Signal d'entrée est de la parole.

D'après les deux Figures (4.5) (4.6), nous remarquons clairement que la vitesse de convergence de l'algorithme NLMSA est meilleure que celle du NLMS. Cela montre bien l'efficacité de l'algorithme NLMSA dans les applications d'AEA.

## 4.5 Résultat de simulation de l'algorithme NLMS amélioré en utilisant le logiciel ModelSim

Dans cette partie, nous allons présenter les résultats expérimentaux de l'implémentation des deux algorithmes NLMS et NLMSA sous le logiciel ModelSim avec des signaux de test qui sont déjà utilisés précédemment, c'est-à-dire le bruit BUSASI et le signal de parole précédemment décrits.

### 4.5.1 Signal BUSASI sous ModelSim

Afin d'évaluer les performances de l'algorithme NLMSA en VHDL, nous l'avons implémenté pour fonctionner sous ModeSim. Nous avons fait une simulation avec le signal BUSASI, nous avons fixé la taille du filtre à 128 et le pas d'adaptation $\mu$ à 1 (max) pour voir le résultat de cette simulation qui nous montre le signal désiré, le signal d'entré et le signal de sortie ainsi que l'erreur, nous avons tracé tous ces signaux sous la figure (4.7) suivante en utilisant le logiciel ModelSim:

D'après cette figure, nous avons bien remarqué que le signal d'erreur de filtrage tend vers zéro après quelques itérations. Cela montre bien que notre algorithme a bien fonctionné sous le logiciel ModelSim. Aussi, cette tendance est bien montrée par le fait que le signal en sortie du filtre adaptatif de l'algorithme NLMSA soit égale au signal d'écho. Cela prouve que le système tel qu'il est implémenté sous ModelSim permet une de dire que notre implémentation a bien réussie sur ModelSim.

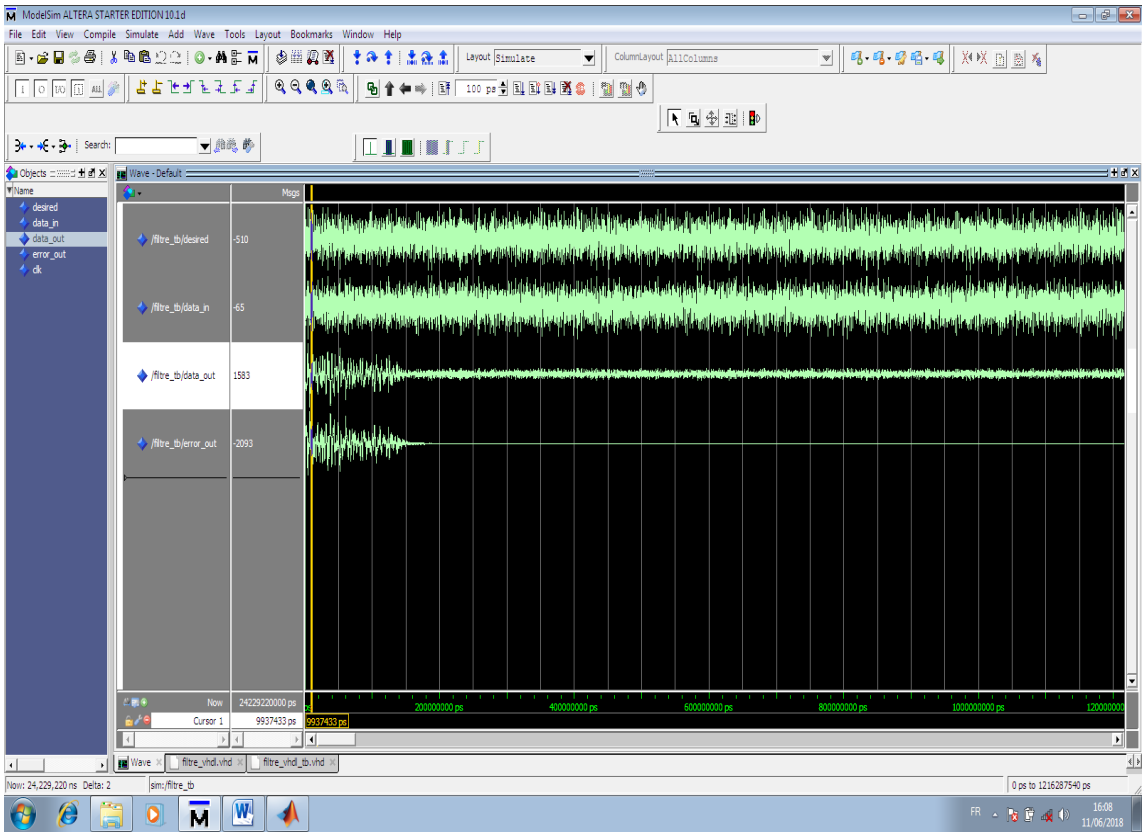


***Figure 4. 7*** Signaux d'entrées et sorties [signal de référence, signal désiré et signal d'erreur de filtrage] de l'implémentation de l'algorithme NLMSA sous ModelSim. Signal d'entrée est le BUSASI, la taille du filtre est L=128.

### 4.5.2 Signal de la parole sous ModelSim

Dans ce deuxième test sous le logiciel ModelSim, nous avons pris les mêmes paramètres qui sont utilisées pour le signal BUSASI dans l'expérience précédente et nous avons fait tourner l'algorithme NLMSA sous ModelSim en utilisant un signal de parole comme signal d'excitation à l'entrée. Ce choix est proche au cas réel car dans les applications de l'annulation d'écho acoustique et de la réduction du bruit acoustique, le signal d'excitation est toujours le signal de parole. Un échantillon des résultats obtenus pour une taille du filtre égale à 28 coefficients sont montré à la figure (4.8) suivante. Les mêmes remarques que précédemment ont été noté dan cette expérience.

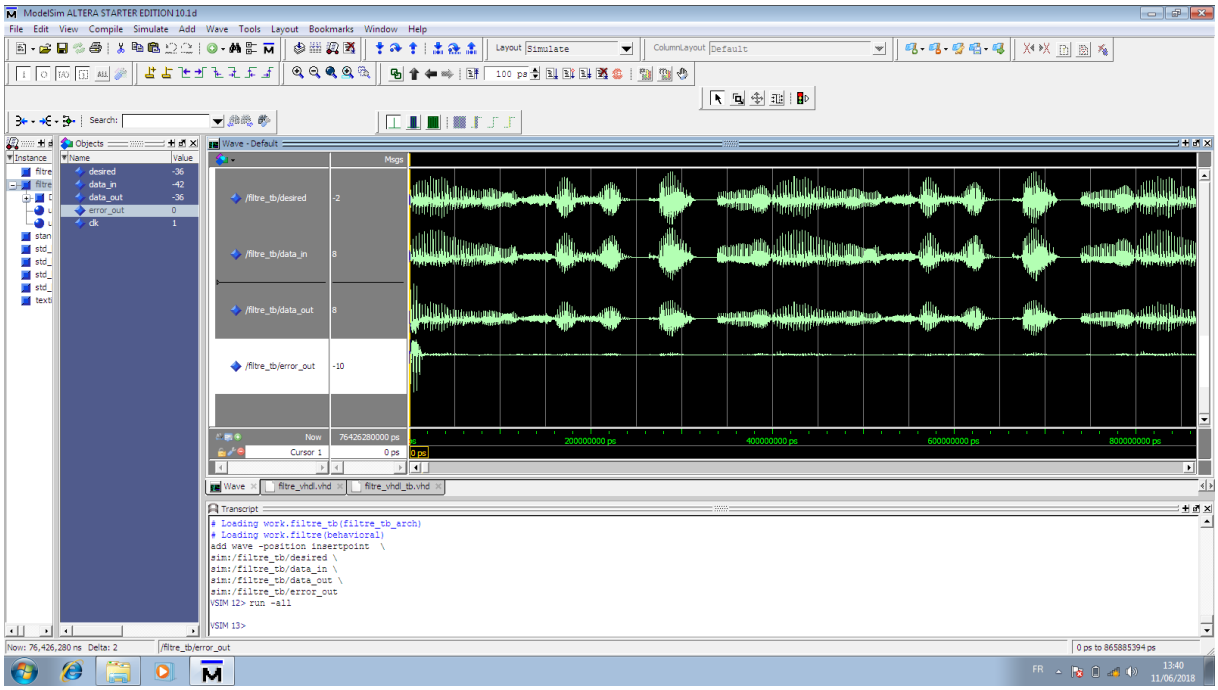


***Figure 4. 8*** 7 Signaux d'entrées et sorties [signal de référence, signal désiré et signal d'erreur de filtrage] de l'implémentation de l'algorithme NLMSA sous ModelSim. Signal d'entrée est de la parole, la taille du filtre est L=128.

### 4.5.3 Comparaison entre les résultats du NLMSA obtenus sous Matlab et VHDL

Dans cette section nous allons faire la comparaison des résultats de simulations de l'algorithme NLMSA sous ModelSim et Matlab en fixant le pas d'adaptation $\mu$ à 1 et variant la taille du filtre, ces résultats vont être exploité comme suit :

#### *a Test avec le signal BUSASI*

Les figures (4.9), (4.10), (4.11) et (4.12) représentent l'évolution temporelle de l'erreur de filtrage linéaire, le MSE, et l'ERLE en utilisant le signal BUSASI comme signal d'entrée des deux expériences faites sur les deux logiciels Matlab et ModelSim, le test d'implémentation a été réalisé avec deux tailles de filtre différentes à savoir L=32, L=128.

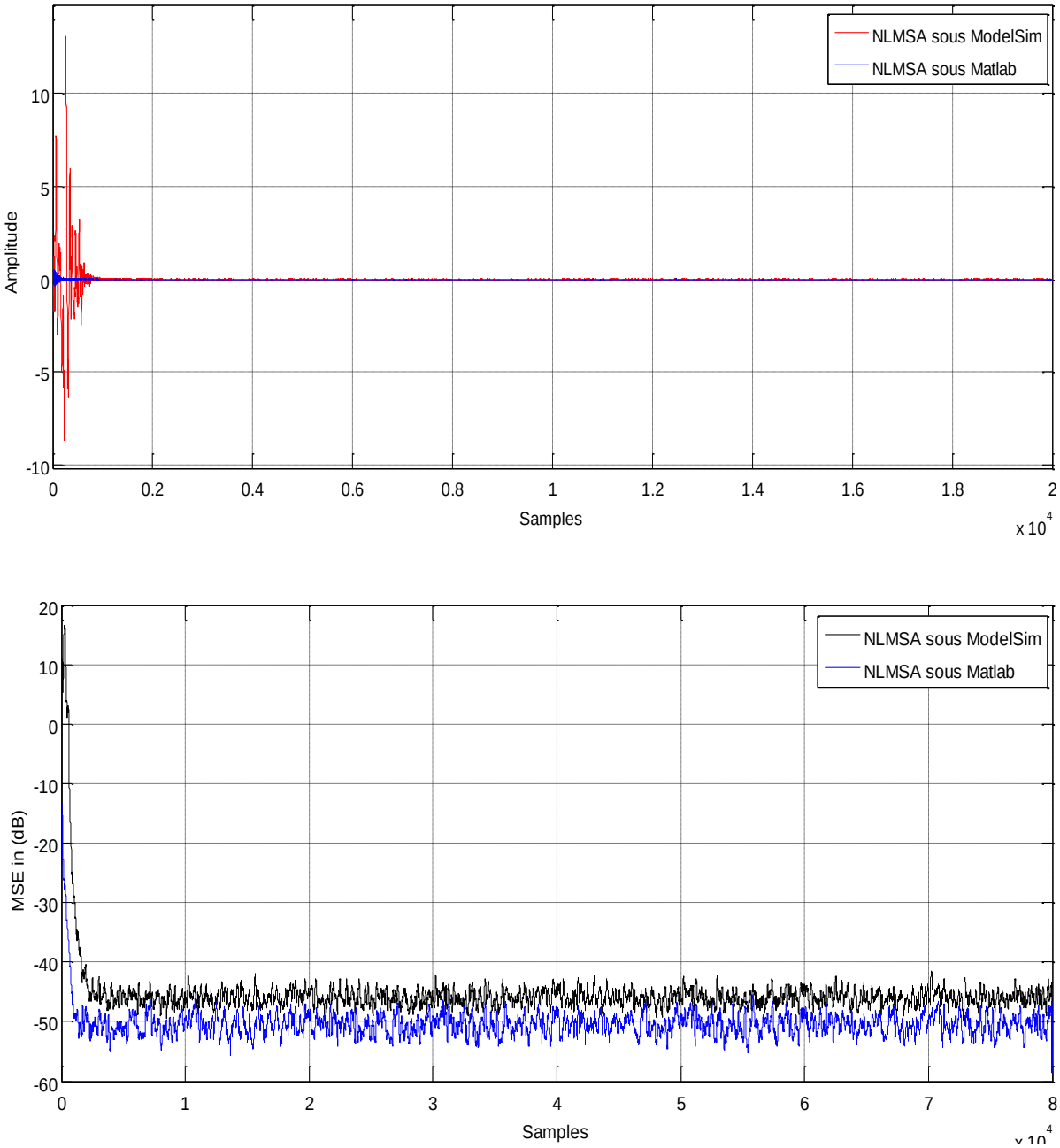


***Figure 4. 9*** Comparaison entre les erreurs de filtrage et les MSE obtenues par NLMSA implémenté sur Matlab et ModelSim avec BUSASI, L=32.

Dans ces figures, que ce soit sur l'erreur de filtrage linéaire ou bien MSE ou encore ERLE, nous remarquons que la vitesse de convergence de l'algorithme NLMSA est plus rapide sur matlab que sur ModelSim. En régime transitoire de ces réultts (début de convergence) nous observons que les évolutions commencent avec une grande amplitude avec ModelSim alor que sur Matlab c'est tout à fait le contraire. Cela revien au type de codage des valeurs ur les deux logiciels.

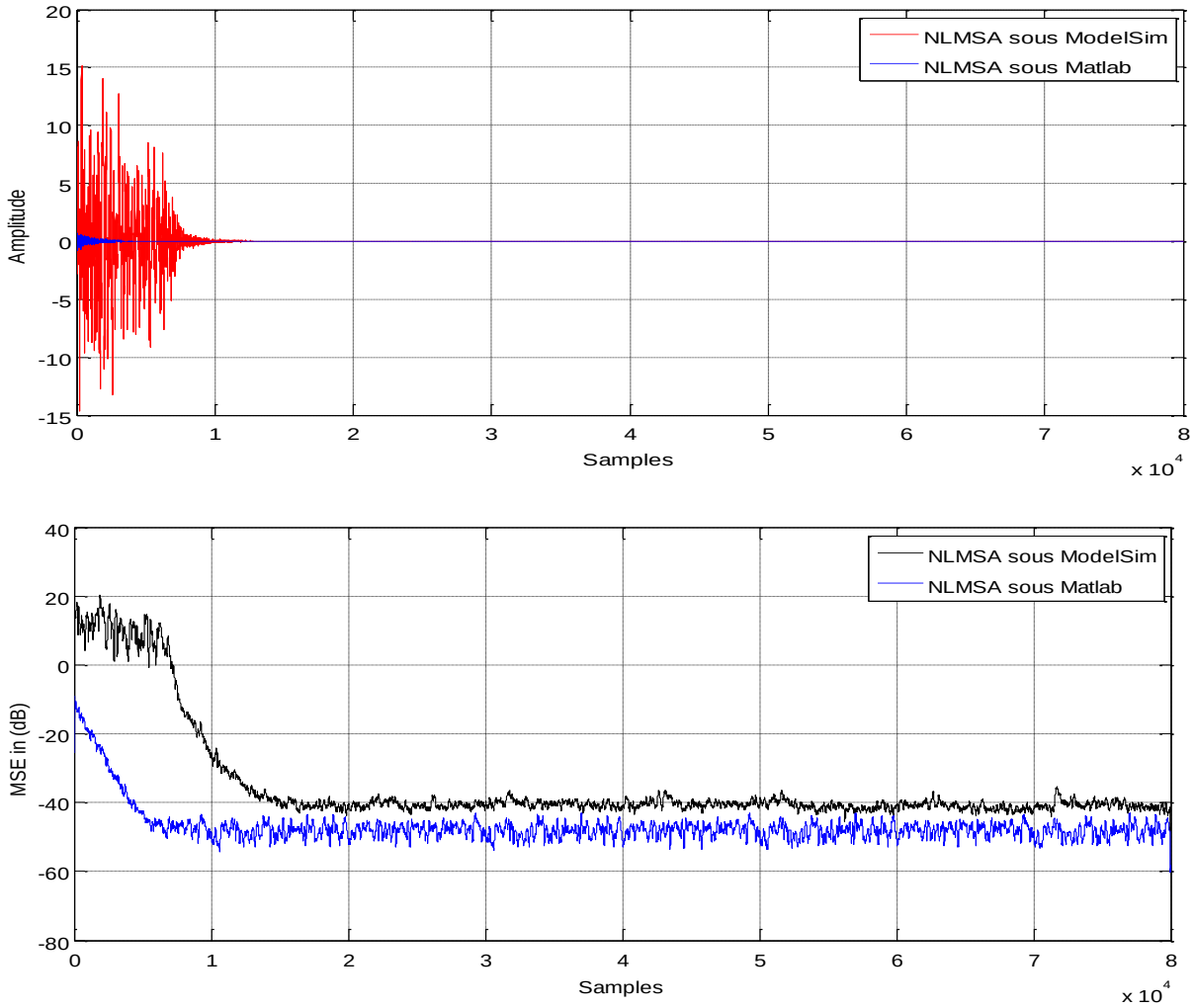


***Figure 4. 10*** Comparaison entre les erreurs de filtrage et les MSE obtenues par NLMSA implémenté sur Matlab et ModelSim avec le signal BUSASI comme signal d'entrée et une taille du filtre L=128.

Dans cette comparaison (voir les figures (4.9) et (4.10) ) l'erreur et le MSE prennent beaucoup de temps pour converger sur ModelSim par rapport au Matlab, de plus à chaque fois qu' on diminue la taille du filtre la vitesse de convergence devient plus rapide. Ce résultat confirme les résultats obtenus précédemment lorsque la taille du filtre varie.

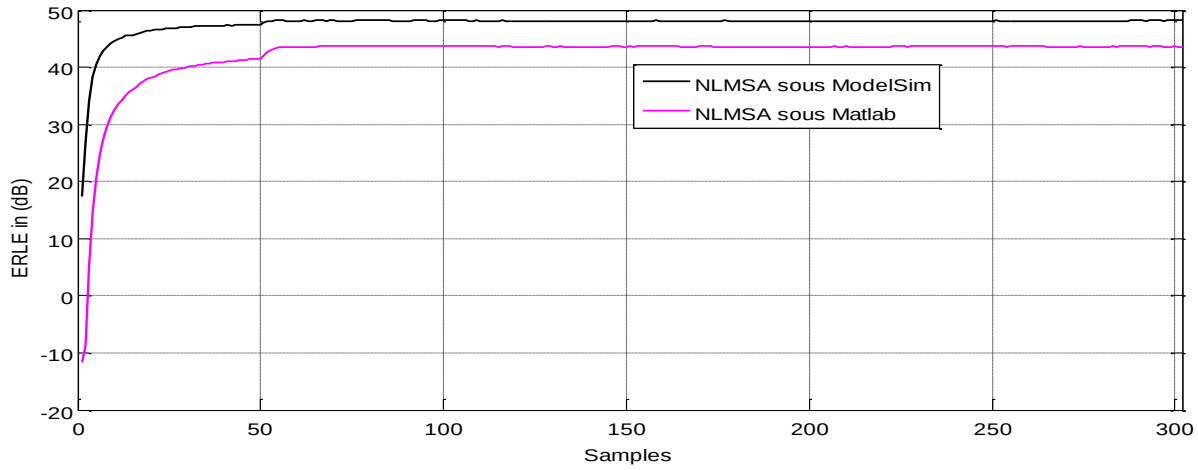


***Figure 4. 11*** *Comparaison de ERLE implémenté sur Matlab et ModelSim avec BUSASI, L=32.*

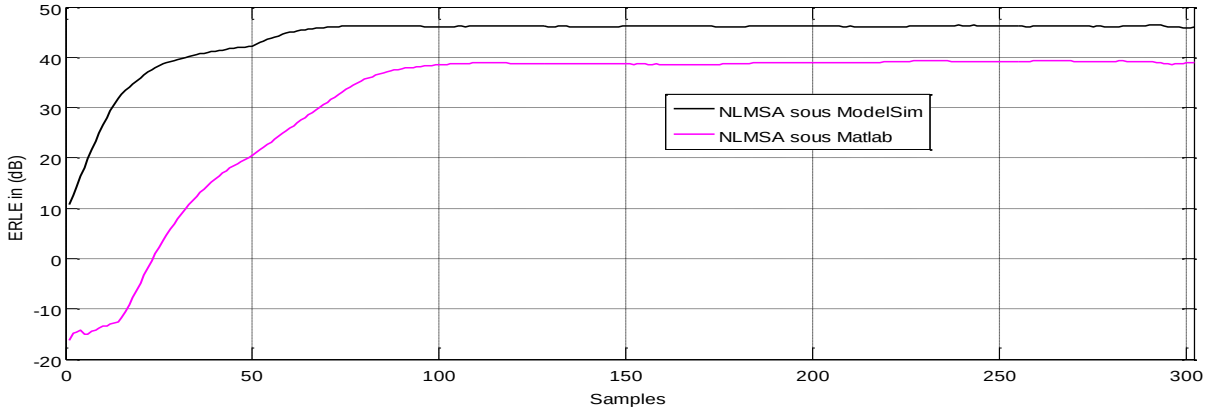


***Figure 4. 12*** *Comparaison de ERLE implémenté sur Matlab et ModelSim avec BUSASI, L=128.*

D'après les deux figures 4.11 et 4.12 du critère ERLA, nous avons remarqué que la vitesse de convergence de l'algorithme NLMSA est plus rapide sur Matlab que dans le cas du logiciel ModelSim. Ces résultats confirme encore une fois la différences de codage des chiffre qui fait la différence entre les deux logiciels.

### *b Test avec le signal de la parole*

Les figure (4.13), (4.14), (4.15) et (4.16) représentent l'évolution temporelle de l'erreur de filtrage linéaire, le MSE, et l'ERLE, en utilisant le signal de parole comme signal source pour l'algorithme NLMSA qui est implémenté sous implémenté sous Matlab et sous ModelSim, le test d'implémentation est réalisé avec deux taille de filtre adaptatif à savoir L=32, L=256.

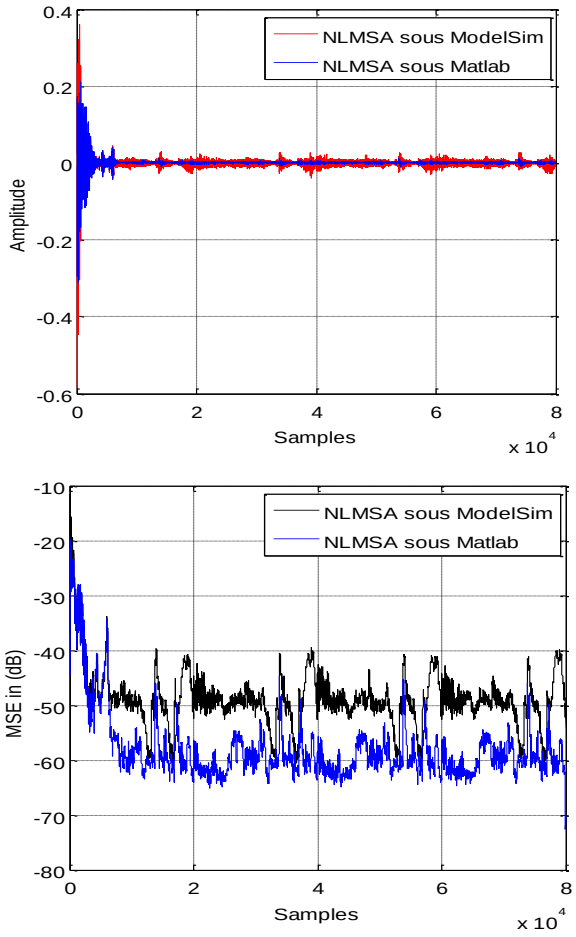


***Figure 4. 13*** Comparaison entre les erreurs de filtrage linéaire et les MSE obtenues par NLMSA implémenté sur Matlab et ModelSim avec le signal de parole comme signal d'entrée et une taille du filtre adaptatif égale à L=32.

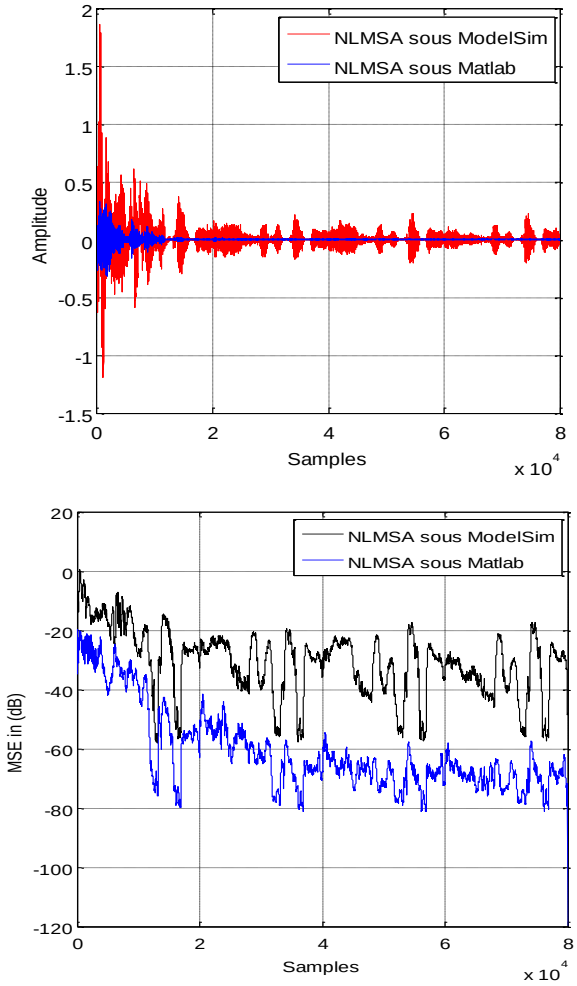


***Figure 4. 14*** Comparaison entre les erreurs de filtrage linéaire et les MSE obtenues par NLMSA implémenté sur Matlab et ModelSim avec le signal de parole comme signal d'entrée et une taille du filtre adaptatif égale à L=256.

D'après les deux figures (4.13) et (4.14) nous remarquons que les erreurs de filtrage et le MSE obtenues par NLMSA sont minimisé quel que soit le régime transitoire jusqu'elles prennent le régime permanent. Nous remarquons aussi que l'erreur de filtrage sous Matlab est plus petite que celle sous ModelSim et c'était le même cas pour le MSE.

Nous notons que, nous pouvons mieux voir la différence entre l'implémentation de NLMSA sous Matlab et l'implémentation de NLMSA sous ModelSim lorsque la taille du filtre adaptatif L est grande.

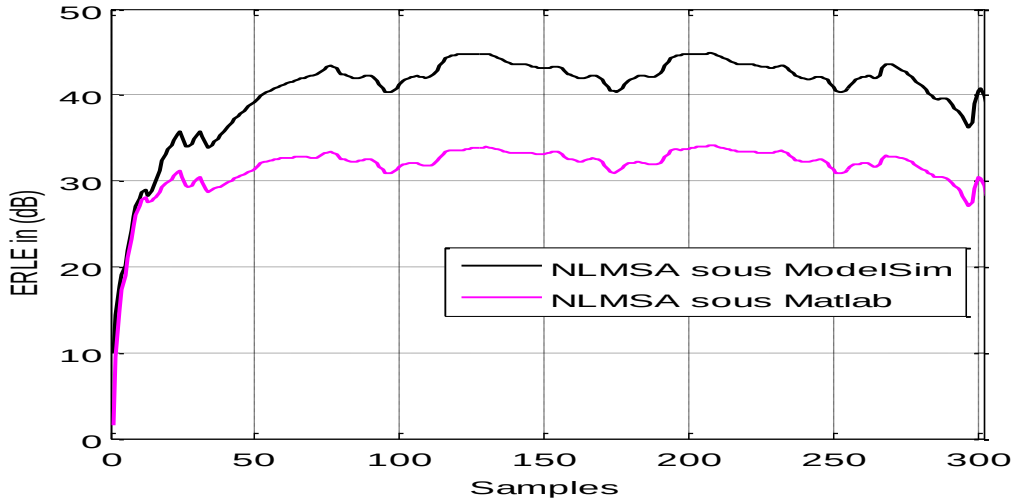


***Figure 4. 15*** Comparaison de ERLE implémenté sur Matlab et ModelSimavec le signal de la parole, L=32.

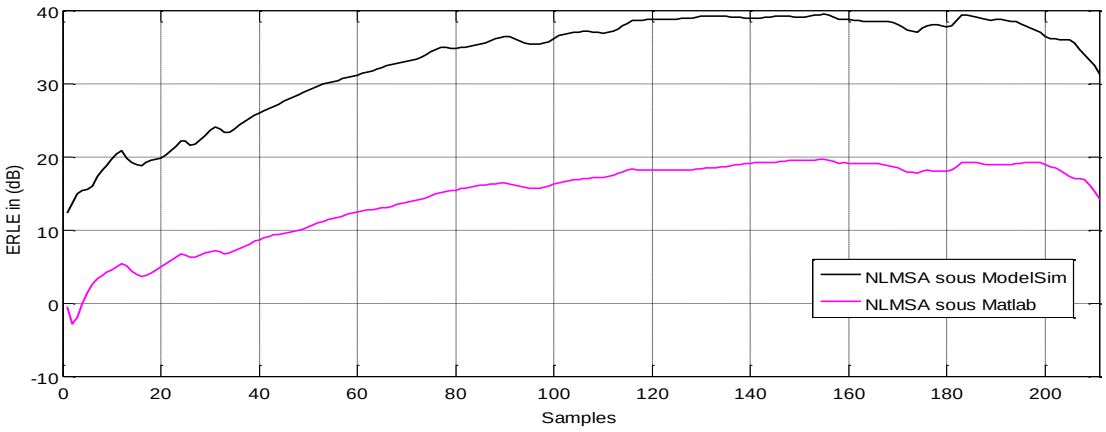


***Figure 4. 16*** Comparaison de ERLE implémenté sur Matlab et ModelSimavec le signal de la parole, L=256.

D'après les deux figures (4.15) et (4.16), nous remarquons que l'ERLE obtenus sous Matlab est plus petit que celui obtenus sous ModelSim. La différence entre ces deux derniers apparaît mieux lorsque la taille du filtre adaptatif est grande, de plus nous remarquons que le graphe de ERLE sous Matlab commence avant le graphe de ERLE sous ModelSim quel que soit la taille du filtre.

A la fin de cette comparaison des résultats de Matlab et ModelSim, nous concluons que les résultats sous Matlab sont plus précis que les résultats sous Modelsim à condition du mode de codage des valeurs sous les deux logiciels.

### 4.5.4 Etude comparative entre NLMS et NLMSA sous ModelSim

Dans cette section nous allons comparer les deux algorithmes NLMS et NLMSA sous ModelSim. Cette comparaison est faite dan un contexte d'annulation d'écho acoustique, c'est-à-dire le signal d'entré est un signal de parole. Les résultats obtenus de cette comparaison ont été exprimés en termes de l'erreur de filtrage linéaire, le MSE et l'ERLE et sont donnés aux figures 4.17, 4.18, et 4.18, respectivement.

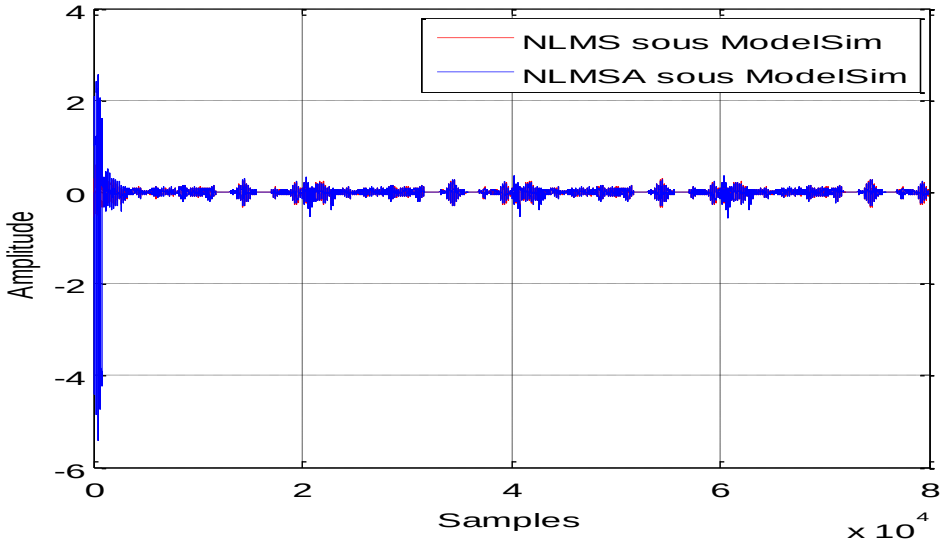


***Figure 4. 17*** Comparaison de la convergence de l'erreur du filtre pour l'algorithme de NLMS et NLMSA. Le signal de parole est pris comme signal d'entrée et la taille du filtre adaptatif égale à L=32.

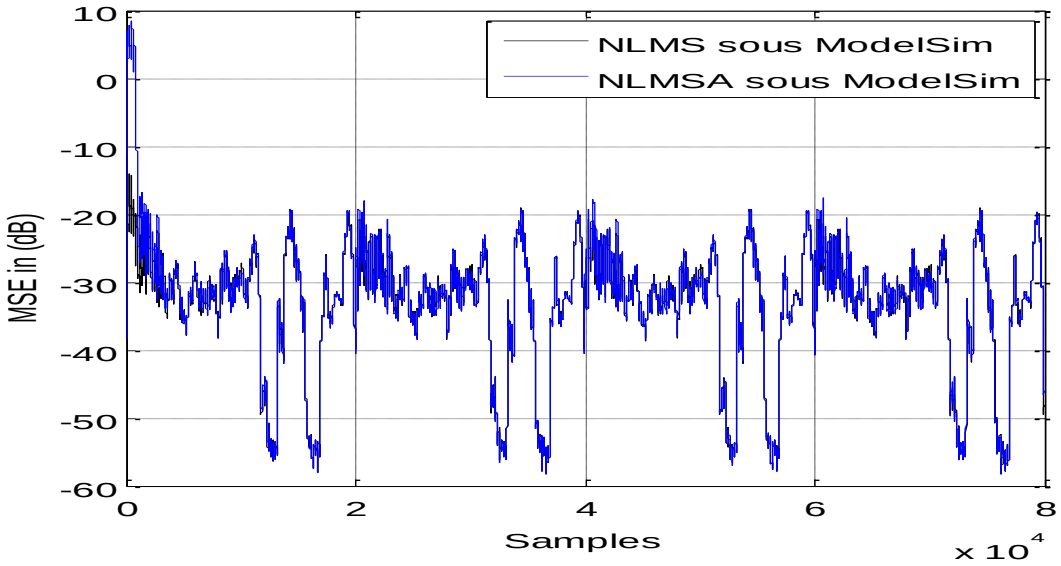


***Figure 4. 18*** Comparaison de la convergence de MSE pour l'algorithme de NLMS et NLMSA. Le signal de parole est pris comme signal d'entrée et la taille du filtre adaptatif égale à L=32.

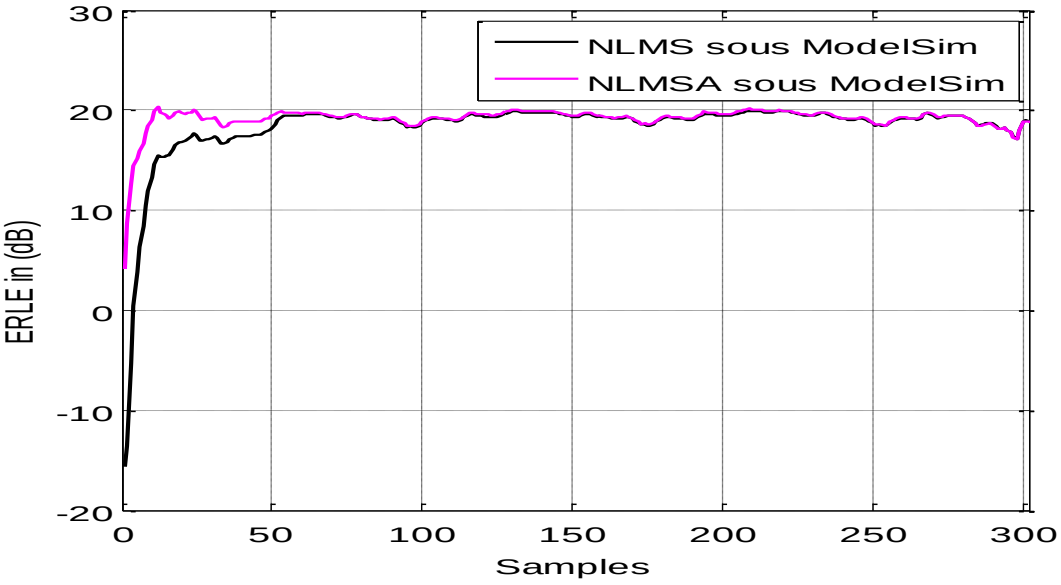


***Figure 4. 19*** Comparaison de la convergence de ERLE pour l'algorithme de NLMS et NLMSA. Le signal de parole est pris comme signal d'entrée et la taille du filtre adaptatif égale à L=32.

D'après les résultats des trois critères (l'erreur de filtrage linéaire, le MSE, et l'ERLE), données aux Figures (4.17), (4.18) et (4.19), nous avons constaté que l'algorithme NLMSA est bien meilleure en termes de vitesse de convergence que l'algorithme NLMS et cela dans les deux experiences faites sous Matlab ou bien sous ModelSim. Cela nou permet de

dire que l'algotithme NLMSA peut faire une bonne alternative pour les appluications d'AEA en temps réels et sur les systèmes électronique embarqués.

## 4.6 Conclusion

Dans ce chapitre, nous avons présenté deux types d'algorithmes (NLMS et NLMSA) pour l'annulation d'écho acoustique (AEA) et qui sont implémenté sur Matlab et ModelSim. les résultats de simulations ont été obtenus en utiliant le signal BUSASI et le signal de la parole afin de teste et comparer les performances de ces deux algorithmes dans un contexte académqie (BUSAI) et aussi dans un cotexte réel d'annulation d'écho acoustique. Tous les résulats ontenus ont été présentés puis analysés dans ce chapitre.

A l'issu de ces résulats, nous avons montré que la méthode d'annulation d'écho acoustique en employant l'algorithme NLMSA dans son fonctionnement est plus efficace que celle basée sur l'algorithme clasique NLMS. Aussi, nous avons remarqué que les résultats obtenus avec l'algorithme NLMSA en utiliant le logiciel Matlab ont donné des résultats plus précis que lorque le logiciel ModelSim est utilisé.

# Conclusion générale

L'utilisation d'un dispositif de communication main-libre est accompagnée de plusieurs problèmes critiques affectant la prise de son tels que la réverbération, l'écho acoustique et le bruit ambiant qui dégradent la qualité et l'intelligibilité de communication, et posent des problèmes très gênants et très difficiles à résoudre.

L'étude que nous avons présentée dans ce mémoire concerne l'annulation du phénomène d'écho acoustique par les algorithmes de filtrage adaptatif.

Dans un premier temps, nous avons étudié théoriquement les deux algorithmes du gradient stochastique normalisée NLMS (Normalised Least Mean Square) et NLMSA (Normalise Least Mean Square Ameliorate), ensuite nous avons implémenté les deux algorithmes NLMS et NLMSA sous Matlab et sous ModelSim pour bien évaluer les performances des deux algorithmes en temps différé et en temps réel.

A l'issu de cette implémentation et des résultats expérimentaux, nous avons remarqué que les performances de l'algorithme NLMSA sont nettement meilleures que celles du NLMS, ceci montre clairement que l'algorithme NLMSA est meilleure que l'algorithme NLMS sur tous les plans. Nous avons également remarqué, que les résultats du logiciel Matlab étaient plus précis que ceux obtenus avec le logiciel ModelSim. D'un autre point de vue, nous avons remarqué que les résultats obtenus sous Matlab sont plus performants que ceux obtenus sous le logiciel ModelSim, cela peut être interprété par le codage des variables et la précision de calcul sur les deux logiciels.

Comme perspective à ce travail, nous envisageons à faire une implémentation de ces deux algorithmes NLMS et NLMSA sur le circuit FPGA.

# Bibliographie


[1] Francis Cottet « aide-mémoire traitement du signal » Dunod, Paris, 2005.

[2] O. Sentiey « Traitement numérique du signal » ENSSAT-Université de Rennes 1, IRISA-équipe de recherche $R^2D^2$, 2003.

[3] O. Sentiey « Analyse et synthese des filtres numeriques » ENSSAT-Université de Rennes 1, 28 mai 2008.

[4] Assia Kourgli «Analyse et Filtrage des Signaux Numériques» University of Science and Technology Houari Boumediene, 2015.

[5] M. BELLANGER « Traitement numérique du signal Théorie et pratique » Science sup, 8 ème édition, Dunop, 2006.

[6] Gilloire A., Julien J.P « L'acoustique des salles dans les télécommunications » L'écho des recherches, N°127, pp 43-54, 1987.

[7] Kuttruff H « Rooms acoustics » Elsevier Applied Science, 1991.

[8] Gilloire A., Veterli M «Performance evalution of acoustic echo controls: Required values and measurement procedures» Annales de Télécommunications, vol 49, 1994.

[9] Vijay k. Madisetti, Dougls B. Williams «Digital signal Processing» CRC Press, 1999.

[10] Dimitris G. Manolakis, Vinay K. Ingle, Stephen M. Kogon «Satistical and adaptative signal Processing» ARTECH HOUSE, 2005.

[11] H.Alaeddine « Application de transformée en nombres entiers a la conception d'algorithmes à faible complexité pour l'annulation d'écho acoustique » Thése de Doctorat, Université de Bretagne Occidentale,2010.

[12] J.Benesty « traitement de signal numérique II, Filtrage de Winer» http://externe.inrs-emt.uquebec.ca/users/benesty/, 2005.

[13] M. Bellenger « traitement numérique du signal » MASSON, 1987.

[14] Greet Rombouts « Adaptive filtering algorithms for acoustic echo and noise cancellation dissertation» KATHOLIEKE UNIVERSITE ITLEUVEN, 2003.

[15] Hugues Benoit-Cattin « Traitement du signal » Département télécommunications, Services & Usages 3TC/INSA-Lyon.

[16] Mohamed Bendada « Implantation d'algorithme de filtrage numérique sur FPGA (réseau de portes programmables) » Université Farhat Abbas de Sétif Algérie-Master électronique.

[17] J.WEBER, M.MEAUDRE « circuits numériques et synthèse logique un outil: VHDL », Collection technologies de l'université à l'industrie.

[18] Peter J. Ashenden « The designer's guide to VHDL» (3rd edition) Morgan Kaufmann, 2008.

[19] © Alexis Polti « Cour en ligne: Type de données, expressions » Journal, TELECOM ParisTech COMELEC, 2005.

[20] E. Messerli « Le langage VHDL, notions de base & synthèse » pp28, (HES-SO / HEIG-VD / REDS), 2017.

[21] Fabrice CAIGNET « Les circuits logiques programmables – FPGA: Introduction au langage VHDL Sémantique » LAAS – CNRS.

[22] T.Perisse « Manuel_QuartusII: VHDL » 2009-2010.

[23] M.Djendi,S.Berkani , H.Dhifallah<<.....>> internationale conférence eu signal processing SIVA 2016 Guelma Algérie.