République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Ecole Nationale Polytechnique
Département d'Electronique

Projet de Fin d'Etudes

Thème :

Contribution à l'implémentation d'un annuleur d'écho sur circuit FPGA

Etudié et soutenu par : M. TAÏLEB El Mehdi.

Devant le jury composé de :

Dr. L.HAMMAMI	Présidente
M. A.K OUDJIDA	Examinateur
Dr. A.BELOUCHRANI	Promoteur
M. IDIRENE	Invité

Promotion 2003

E.N.P 10, Avenue Hassen Badi, B.P 182 El-Harrach, Alger, Algérie.

ملخص

هذا العمل مساهمة في غرس حاذف للصدى الصوتي بخوارزمية GMDFα على دارة مبرمجة من النوع FPGA.
ادخلنا في هذا الاطار تقنيات VLSI التي تسمح توسع الخوارزمية قبل البدء في عملية الغرس. شرحنا كذلك لزوم تبنى منهجية دقيقة و كذا حبكات اعلامية متطورة قصد اتمام هذا العمل.

Abstract

This work is a contribution to the implementation of an echo canceller based on the GMDFα algorithm on FPGAs. We introduced the VLSI digital signal processing techniques in order to optimize the algorithm before moving to the implementation leading to increase the performances of the final circuit in term of speed and area. We also explained why the use of a strict design methodology and powerful software tools is so important to achieve this work.

Résumé

Ce travail est une contribution à l'implémentation sur circuit FPGA d'un annuleur d'écho acoustique basé sur l'algorithme GMDFα. Nous avons introduits les techniques VLSI afin d'optimiser l'algorithme avant même de passer à l'implémentation ce qui permet un gain important en terme de taille et de vitesse. Nous avons aussi expliqué la nécessité d'adopter une méthodologie stricte ainsi que l'utilisation d'outils logiciels performants afin de mener à bien ce travail.

Mots clé

Annulation d'écho, implémentation, VLSI, GMDFα, méthodologie de conception.

Table des matières

Liste des figures

Introduction

Lors de la conception d'un système de communication offrant des possibilités de mains libres, l'ingénieur est confronté au problème de l'écho acoustique qui peut devenir très pénalisant pour la qualité de la communication. Ce problème s'accentue encore plus en téléphonie mobile où le milieu est en constant changement.

En effet, le milieu environnant réfléchit les sons émis par un haut parleur. Le signal résultant de la superposition des différentes réflexions se superposera au signal utile capté par le microphone et sera retransmis.

La solution retenue afin d'annuler l'écho acoustique consiste à estimer la réponse impulsionnelle du milieu environnant. Puis, à partir de celle-ci, estimer le signal de l'écho et le soustraire du signal de l'écho réel.

Le filtrage adaptatif a été retenu afin de pouvoir suivre les changements en fonction du temps de la réponse impulsionnelle du milieu.

Le phénomène de l'écho acoustique présente en outre une difficulté supplémentaire qui réside dans le fait que la réponse impulsionnelle est généralement très longue. D'où la difficulté d'implémentation de solutions temps réel satisfaisantes. Le traitement dans le domaine fréquentiel a donc été préféré car [2] il présente l'avantage de réduire considérablement la charge de calculs.

Différentes études et comparaisons entre les algorithmes opérant dans le domaine temporel (LMS : Least Mean Square, NLMS : Normalized LMS, BLMS : Block LMS) et fréquentiel (FBLMS : Frequency-Domain BLMS, MDF : Multidelay Frequency-Domain Adaptive filter, GMDFα : Generalized MDF α) ont mené à choisir l'algorithme GMDFα. Deux projets [2] et [3] de fin d'études, réalisés au niveau du département d'électronique de l'Ecole Nationale Polytechnique, ont déjà traités les aspects théoriques inhérents aux avantages du GMDFα. Nous avons aussi à notre disposition un programme Matlab complet [2] permettant de tester les performances du GMDFα sur des signaux réels.

Reste ensuite le problème d'implémentation de l'algorithme. Trois solutions sont en fait envisageables : Software, Hardware ou mixte.

La solution software (déjà explorée dans [2]) n'est pas satisfaisante en raison des contraintes temps réel mais pourra être exploitée comme plateforme de validation d'une solution Hardware par exemple.

La solution Hardware, plus appropriée, présente l'avantage que l'on peut personnaliser à volonté l'architecture de l'implémentation. On peut en effet exploiter par exemple le parallélisme présent dans l'algorithme et choisir la longueur des mots.

La solution mixte consiste à effectuer tout d'abord un partitionnement entre le Hardware et le Software afin de choisir les parties de l'algorithme à implémenter en Hardware où en Software. Toutefois, ce partitionnement est très difficile à effectuer. De plus, ce type de conception nécessite des compétences multiples (e.g. DSP, ASIC, FPGA), ainsi que la disponibilité de logiciels pouvant gérer ce type de cycles de conception (e.g. partitionnement H/S, conception en parallèle de l'architecture Hardware et Software, simulation mixte).

Nous avons donc choisi l'approche hardware où nous avons le choix entre :

- Un cycle full custom (ASICs) où il est nécessaire d'effectuer une conception dont le niveau d'abstraction descend du niveau algorithmique jusqu'au niveau transistor. De plus, il faut soumettre le design à toute une batterie de test allant de la simulation fonctionnelle jusqu'aux tests relatifs à la validité de la fabrication.
- Un cycle FPGA (les autres types de circuits logiques programmables sont à exclure car n'offrant pas les ressources suffisantes pour ce type d'applications).

Le cycle FPGA offre l'avantage d'affranchir le concepteur de la phase de fabrication et des tests post-fabrication, le niveau d'abstraction s'arrêtant au niveau des primitives. De plus, il offre l'avantage d'un coût (financier et de temps) de développement réduit dans le contexte académique où nous nous situons. Enfin, comme nous nous situons dans un contexte de conception VLSI, qui est indépendante de la technologie choisie, le design réalisé peut être facilement porté d'une plateforme à une autre.

I. OBJECTIFS :

L'objectif de ce projet de fin d'études sera donc d'effectuer une étude sur l'algorithme GMDFα et d'optimiser son exécution en vue de son implémentation temps réel sur circuit FPGA.

L'implémentation devant se faire suivant une méthode de conception scrupuleusement respectée afin d'assurer la consistance du circuit final avec l'algorithme initial et de gérer les aspects inhérents à la complexité de l'algorithme à implémenter.

La consistance de l'implémentation doit être en permanence vérifiée en effectuant des simulations fonctionnelles et temporelles. Une analyse temporelle statique est aussi nécessaire afin de déterminer les délais les plus longs pouvant rendre l'implémentation en temps réel impossible et de trouver éventuellement des solutions.

La validation du circuit final devra être automatique, i.e. qu'elle sera effectuée par un programme qui comparera les résultats obtenus par simulation du circuit avec ceux obtenus par l'implémentation Matlab de l'algorithme GMDFα.

II. ORGANISATION DU RAPPORT :

Nous commencerons ce rapport par exposer le problème de l'écho acoustique ainsi que l'algorithme GMDFα.

Ensuite, nous exposerons les techniques VLSI (Very Large Scale Integration) permettant d'optimiser l'implémentation d'algorithmes de traitement numérique du signal [1]. Nous illustrerons l'utilisation de certaines de ces techniques sur l'algorithme GMDFα.

Dans le troisième chapitre, nous présenterons la méthodologie de conception top-down et présenterons les avantages de suivre une méthode stricte lors de l'implémentation. Nous parlerons aussi des outils logiciels utilisés pour mettre en œuvre la méthode choisie.

Dans le quatrième chapitre, nous passerons à l'implémentation de l'algorithme GMDFα sur FPGA, nous exploiterons au maximum les notions exposées dans les deux chapitres sur les techniques VLSI et sur la méthodologie de conception afin d'aboutir à une implémentation efficace en un minimum de temps. Nous aborderons également dans ce chapitre la phase des tests que nous avons fait subir au circuit à chaque étape de la conception. Nous présenterons aussi les résultats obtenus.

Le dernier chapitre conclura ce travail en mettant en exergue les objectifs atteints, ceux que nous n'avons pas pu réaliser et enfin les propositions d'enrichissement et de perfectionnement de ce travail.

Chapitre I

L'ALGORITHME GMDFα

I. INTRODUCTION A L'ANNULATION D'ECHO ACOUSTIQUE:

L'écho acoustique est un phénomène physique qui résulte de la superposition des différentes réflexions (par le milieu environnant) d'un son émis par un haut parleur.

Le signal de l'écho va donc être retransmis vers le locuteur lointain (celui-ci va s'entendre parler avec un retard). De plus, le milieu où se trouve le locuteur lointain pourra aussi réfléchir l'écho. Dans ce cas, un sifflement très désagréable rendra la communication impossible.

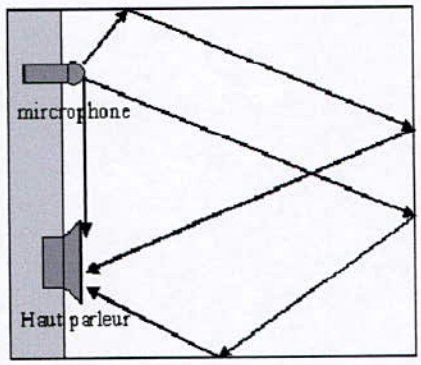


Figure 1 illustration du phénomène de l'écho acoustique.

Dans le contexte d'une téléconférence par exemple, le locuteur proche écoutera le message envoyé par le locuteur lointain. Ce message, émis par le/les haut-parleur(s) se trouvant dans la salle sera réverbéré par la salle où se trouve le locuteur proche, puis sera capté par le microphone et réémis vers le locuteur lointain.

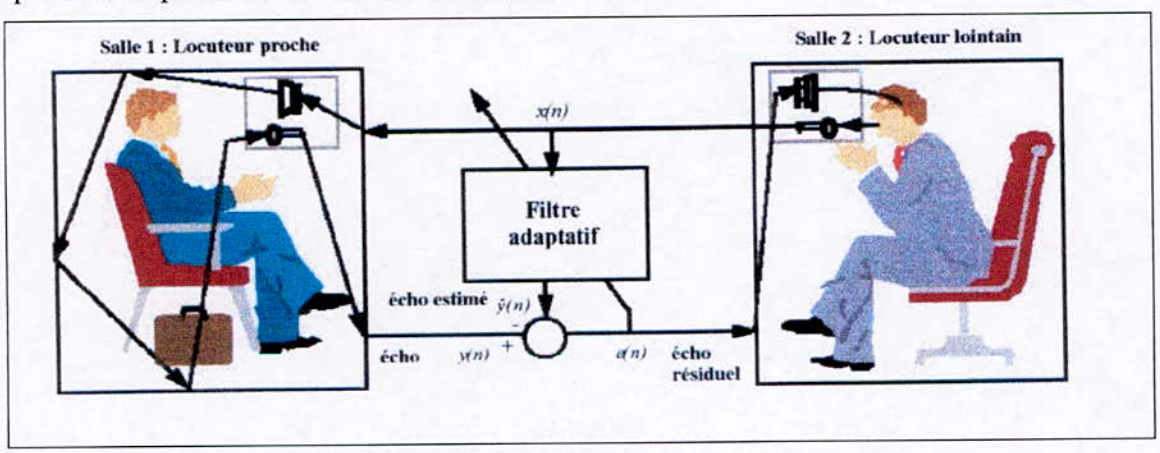


Figure 2 Génération et annulation de l'écho acoustique dans un contexte de téléconférence.

Il est donc nécessaire de connaître la réponse impulsionnelle du milieu où se propage le son. Et ce afin de le prédire. Cet écho prédit sera alors soustrait de l'écho réel (capté par le microphone) puis transmis vers le locuteur lointain.

L'annulation de l'écho acoustique consiste donc à trouver les coefficients du filtre modélisant l'environnement entourant le locuteur proche. Mais puisque cet environnement n'est pas connus a priori, et surtout, du fait qu'il change en fonction du temps (déplacement du locuteur dans le cas de la téléphonie mobile), il est indispensable que ces coefficients soient calculés dynamiquement, d'où la nécessité d'effectuer un filtrage adaptatif.

La solution du problème peut être alors schématisée par la figure suivante où x(n) est le signal provenant du locuteur lointain, y(n) est l'écho estimé, d(n) est l'écho réel désiré et e(n) est le signal d'erreur qui représente l'écho résiduel :

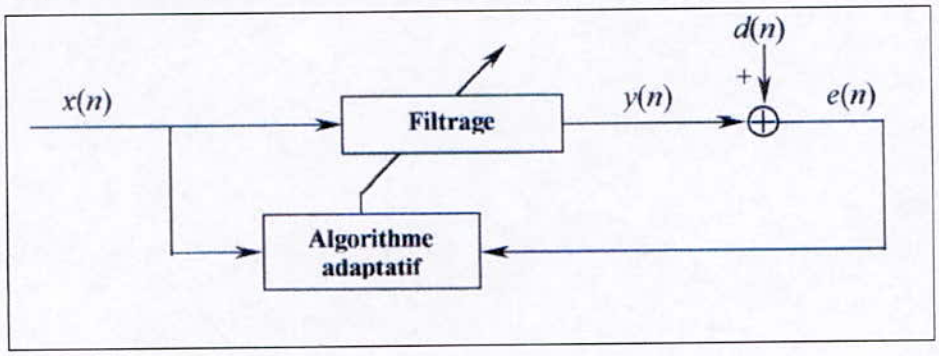


Figure 3 Le filtrage adaptatif.

Notre choix s'est naturellement opéré sur l'algorithme GMDFα qui est un algorithme de filtrage adaptatif dans le domaine fréquentiel. En effet, les études précédemment menées [2], ont montré un gain considérable, par rapport aux algorithmes opérant dans le domaine temporel tels que le LMS ; BLMS, en complexité arithmétique pour les réponses impulsionnelle longues.

II.CONVOLUTIONS RAPIDES DANS LE DOMAINE FREQUENTIEL :

II.A.Matrices circulantes :

Une matrice ψ de dimensions M x M est dite circulante si et seulement si :

$\Psi_{ij} = \Psi_{1<i-j>M} = \Psi_{<i-j>M}$, où <n>M est le résidu de n modulo M.

Les lignes et les colonnes d'une matrice circulante se déduisent les unes des autres par circulation (d'où la dénomination de circulante) :

$$\Psi = \begin{bmatrix} \Psi_0 & \Psi_{M-1} & \bullet & \bullet & \bullet & \Psi_2 & \Psi_1 \\ \Psi_1 & \Psi_0 & \bullet & \bullet & \bullet & \Psi_3 & \Psi_2 \\ \bullet & \bullet & \bullet & & & \bullet & \bullet \\ \bullet & \bullet & & \bullet & & \bullet & \bullet \\ \bullet & \bullet & & & \bullet & \bullet & \bullet \\ \Psi_{M-2} & \Psi_{M-3} & \bullet & \bullet & \bullet & \Psi_0 & \Psi_{M-1} \\ \Psi_{M-1} & \Psi_{M-2} & \bullet & \bullet & \bullet & \Psi_1 & \Psi_0 \end{bmatrix}.$$

Les propriétés les plus importantes des matrices circulantes sont :

- Une matrice circulante Ψ peut s'écrire sous la forme : $\Psi = W_M^{-1} X W_M$ où X est la matrice diagonale dont les éléments sont la transformée de Fourier discrète de la première colonne de Ψ et W_M et W_M^{-1} sont respectivement les coefficients de la TFD et de la TFD inverse.
- Le produit de deux matrices circulantes est commutatif.
- Le produit de deux matrices circulantes est une matrice circulante.

II.B.Convolution par OLS :

Dans le domaine temporel, l'évaluation d'un bloc de N sorties successives dans l'intervalle [n,n+N+1] d'un filtre RIF modélisé par les coefficients $\{h_n\}$ est donnée par :

$$\begin{bmatrix} y_n \\ y_{n+1} \\ \\ \\ y_{n+N-1} \end{bmatrix} = \begin{bmatrix} x_n & x_{n-1} & \bullet & \bullet & \bullet & x_{n-L+1} \\ x_{n+1} & x_n & \bullet & \bullet & \bullet & x_{n-L+2} \\ \bullet & \bullet & \bullet & & & \bullet \\ \bullet & \bullet & & \bullet & & \bullet \\ \bullet & \bullet & & & \bullet & \bullet \\ x_{n+N-1} & x_{n+N-2} & \bullet & \bullet & \bullet & x_{n+N-L} \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ \\ \\ h_L \end{bmatrix}.$$

En effectuant une extension dans le passé de la première colonne de la matrice des entrées, on obtient :

$$\begin{bmatrix} \widetilde{y}_{n-L+1} \\ \widetilde{y}_{n-L+2} \\ \bullet \\ \bullet \\ \bullet \\ \widetilde{y}_{n-1} \\ \widetilde{y}_{n} \\ \widetilde{y}_{n+1} \\ \bullet \\ \bullet \\ \bullet \\ \widetilde{y}_{n+N-2} \\ \widetilde{y}_{n+N-1} \end{bmatrix} = \begin{bmatrix} x_{n-L+1} & x_{n+N-1} & \bullet & \bullet & \bullet & x_{n+N-L+1} & x_{n+N-L} & x_{n+N-L-1} & \bullet & \bullet & \bullet & x_{n-L+2} \\ x_{n-L+2} & x_{n-L+1} & \bullet & \bullet & \bullet & x_{n+N-L+2} & x_{n+N-L+1} & x_{n+N-L} & \bullet & \bullet & \bullet & x_{n-L+3} \\ \bullet & \bullet & \bullet & & \bullet & \bullet & \bullet & \bullet & & \bullet & \bullet & \\ \bullet & \bullet & \bullet & & \bullet & \bullet & \bullet & \bullet & & \bullet & \bullet & \\ \bullet & \bullet & \bullet & & \bullet & \bullet & \bullet & \bullet & & \bullet & \bullet & \\ x_{n-1} & x_{n-2} & \bullet & \bullet & \bullet & x_{n+N-1} & x_{n+N-2} & x_{n+N-3} & \bullet & \bullet & \bullet & x_{n} \\ x_{n} & x_{n-1} & \bullet & \bullet & \bullet & x_{n-L+1} & x_{n+N-1} & x_{n+N-2} & \bullet & \bullet & \bullet & x_{n+1} \\ x_{n+1} & x_{n} & \bullet & \bullet & \bullet & x_{n-L+2} & x_{n-L+1} & x_{n+N-1} & \bullet & \bullet & \bullet & x_{n+2} \\ \bullet & \bullet & \bullet & & \bullet & \bullet & \bullet & \bullet & & \bullet & \bullet & \\ \bullet & \bullet & \bullet & & \bullet & \bullet & \bullet & \bullet & & \bullet & \bullet & \\ \bullet & \bullet & \bullet & & \bullet & \bullet & \bullet & \bullet & & \bullet & \bullet & \\ x_{n+N-2} & x_{n+N-3} & \bullet & \bullet & \bullet & x_{n+N-L-1} & x_{n+N-L-2} & x_{n+N-2} & \bullet & \bullet & \bullet & x_{n+N-1} \\ x_{n+N-1} & x_{n+N-2} & \bullet & \bullet & \bullet & x_{n+N-L} & x_{n+N-L-1} & x_{n+N-1} & \bullet & \bullet & \bullet & x_{n-L+1} \end{bmatrix} \cdot \begin{bmatrix} h_0 \\ h_1 \\ \bullet \\ \bullet \\ \bullet \\ h_{L-2} \\ h_{L-1} \\ 0 \\ \bullet \\ \bullet \\ \bullet \\ 0 \\ 0 \end{bmatrix}.$$

Cette dernière relation peut être écrite de façon plus compacte :

$\widetilde{y}_n = \phi_n \widetilde{h}$ ....(1)

En développant, on montre que les N derniers termes correspondent à la convolution linéaire désirée, les *L-1* premiers termes quand à eux correspondent à une convolution circulaire.

On voit bien que la matrice ϕ_n est circulante, on peut alors utiliser la propriété de la décomposition des matrices circulantes dans le domaine fréquentiel :

$\phi_n = W_M^{-1} \Omega_n W_M$. Ω_n étant la matrice diagonale dont les éléments sont la TFD Φ_n de la première colonne de la matrice ϕ_n.

$\Phi_n = W_M [x_{n-L+1} \quad \bullet \quad \bullet \quad \bullet \quad x_{n-1} \quad x_n \quad x_{n+1} \quad \bullet \quad \bullet \quad \bullet \quad x_{n+N-1}]^T$.

L'équation (1) devient alors :

$\widetilde{y}_n = W_M^{-1} \Phi_n \circ H$. L'opérateur $\circ$ représente le produit de Schur des vecteurs complexes (produit terme à terme). Et M la plus petite puissance de 2 supérieure à L+N-1.

Le vecteur H étant la transformée de Fourier discrète des coefficients du filtre :

$$H = W_M \begin{bmatrix} h \\ 0_{(N-1)\times 1} \end{bmatrix}.$$

L'isolement des N derniers termes de $\widetilde{y}_n$ correspondant à le convolution linéaire désirée se fait avec le fenêtre rectangulaire f :

$$f = \begin{bmatrix} 0_{(L-1)\times(L-1)} & 0_{(L-1)\times(N)} \\ 0_{(N)\times(L)} & I_{N\times N} \end{bmatrix}.$$

Finalement :

$$y_n = fW_M^{-1}\Phi_n \circ H .$$

Cette dernière relation décrit l'algorithme de convolution rapide OLS (overlap-save ou recouvrement sauvegarde).

Nous déduisons de l'utilisation de cette technique les avantages suivants :

- Le traitement du signal d'entrée par blocs permet de garder les coefficients du filtre ***h*** inchangés pendant toute la durée de traitement du bloc.
- La convolution, opération complexe lorsqu'elle est effectuée dans le domaine temporel, se réduit à une simple multiplication dans le domaine fréquentiel.

La figure suivante montre le schéma simplifié décrivant l'algorithme OLS :

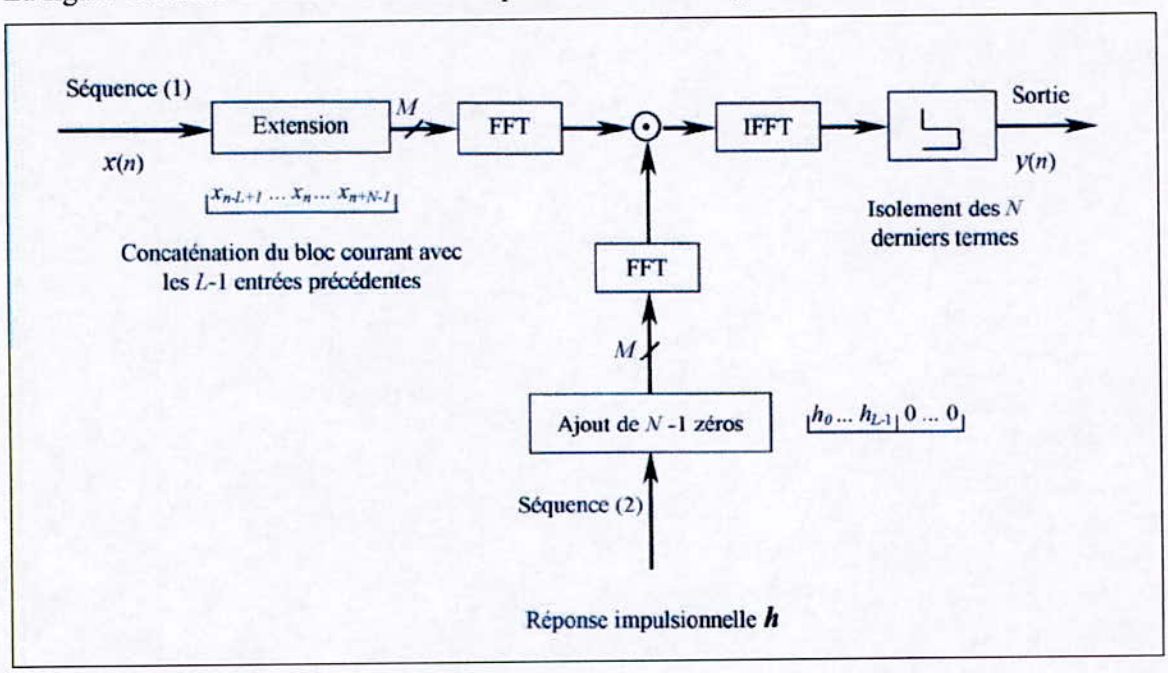


Figure 4 Convolution rapide par la méthode OLS

II.C.Convolution par WOLA :

La méthode WOLA (weighted overlap add) est similaire à la méthode OLS sauf que chaque bloc d'entrée est segmenté en α sous blocs de taille R=[N/α] ([x] étant la partie entière de x).

A chaque nouvelle itération s, un sous bloc de taille R est pris en considération. Ce dernier est complété par les (2α-1) blocs précédents afin d'effectuer une convolution rapide sur le bloc $\tilde{y}_s$ suivant la méthode OLS.

Soit $\boldsymbol{f}$ la fenêtre de troncature permettant d'isoler les N dernières composantes du vecteur de sortie : $f = \left[0_{1\times(L-1)} \quad f_0 \quad \bullet \quad \bullet \quad \bullet \quad f_{N-1}\right]^T$.

Cette fenêtre doit satisfaire une condition de normalisation exprimée par :

$\sum_{i=0}^{\alpha-1} f_{n-iR} = 1$ pour n=1,…,R.

On définit z_s comme étant le vecteur de sorties pondérées de taille N contenant les composantes isolées par la fenêtre de reconstitution $\boldsymbol{f}$:

$z_s = \left[0_{N\times N} \quad I_{N\times N}\right](f \circ \tilde{y}_s)$.

Chaque échantillon du signal de sortie pourra être synthétisé à partir de α blocs z_s. cela se fait par la relation :

$y_s(n) = \sum_{k=0}^{\alpha-1} z_{s-k}(n+kR)$. Pour n=1,…,R.

Le principal avantage de cette méthode est la possibilité de modifier la vitesse de rafraîchissement des coefficients du filtre $\boldsymbol{h}$.

La figure suivante résume la convolution par la méthode WOLA :

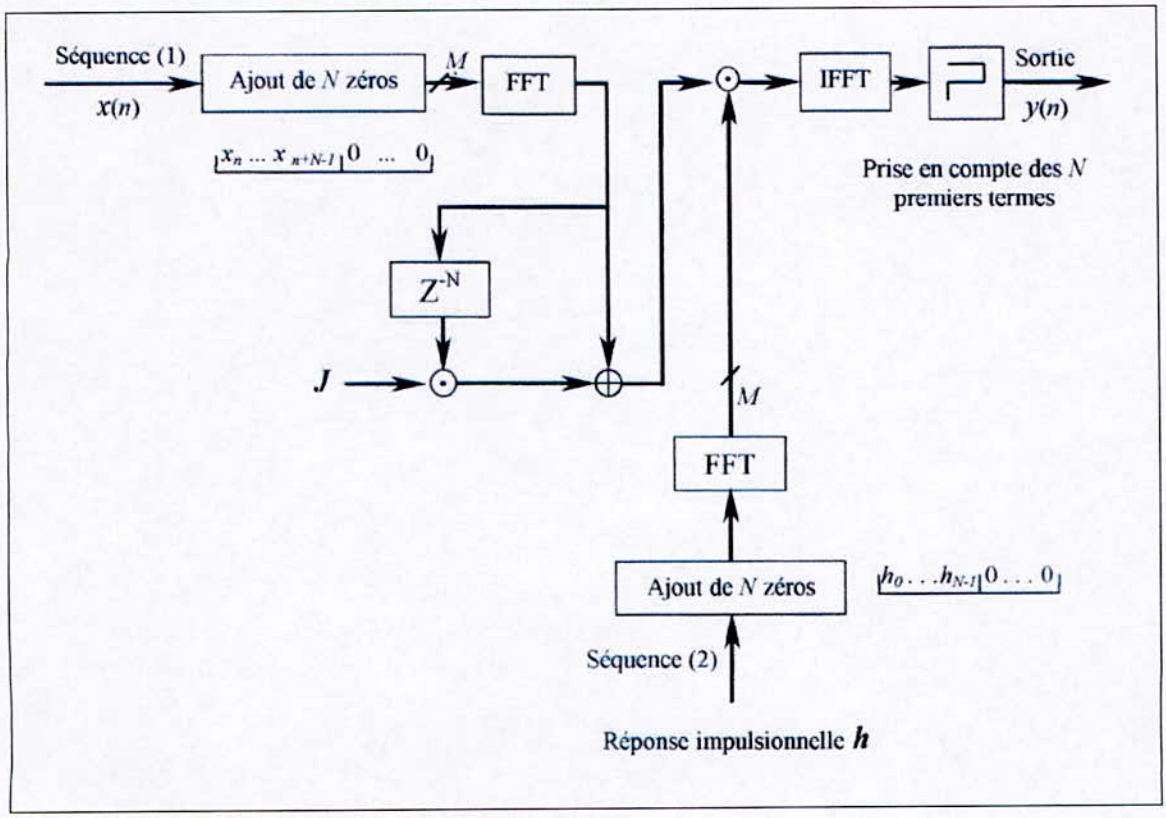


Figure 5 Convolution rapide par la méthode WOLA.

III. L'ALGORITHME GMDFα :

Soit $\mathbf{H_L}$ le modèle (filtre FIR) estimé du système, avec L : nombre de coefficients.

L'algorithme GMDFα a pour principe de segmenter la réponse impulsionnelle en K segments de longueur P, on aura alors $L = K \cdot P$, ce qui permet de réduire la taille des FFTs à utiliser.

Une autre particularité est qu'il traite le signal d'entrée par blocs en utilisant la convolution rapide WOLA. Cela a pour avantage que les coefficients du filtre restent inchangés pendant toute la durée de traitement du bloc. De plus, on introduit un recouvrement entre les blocs successifs afin d'accélérer la mise à jour des coefficients du filtre ainsi que la convergence de l'algorithme.

L'algorithme traite donc des blocs de N échantillons, dont R sont mis à jour à chaque itération tandis que les N-R proviennent du signal de l'itération précédente. Le rapport N/R=α

est appelé facteur de recouvrement. On préfère choisir α comme entier naturel par souci de simplicité.

Afin de minimiser le nombre de FFTs à calculer, on choisis N=P.

Les étapes d'exécution de l'algorithme sont les suivantes :

- Formation du bloc courant d'entrées fréquentielles: à cette étape, on commence par l'acquisition des R nouveaux échantillons (on utilise s comme indice du bloc courant) : $x_{sR} = [x(sR),...,x(sR+R-1)]^T$. Puis on complète le bloc par les 2N-R échantillons précédents : $x_s = [x(sR-2N+R),...,x(sR),...,x(sR+R-1)]^T$.

  Vient ensuite l'évaluation du bloc fréquentiel courant: $X_s^{(0)} = FFT_{2N}(x_s)$. La FFT étant calculée sur 2N points. Ce bloc sera injecté dans le premier segment du filtre.
- Evaluation des coefficients de normalisation : ceux-ci seront utilisés dans la prochaine itération.

  Tout d'abord, il faut évaluer la puissance du signal d'entrée : $P_s(i) = \gamma P_{s-1}(i) + (1-\gamma)\left|X_s^{(0)}(i)\right|^2$, $i = 1,...,2N$. γ étant le facteur d'oubli dont la valeur se situe dans l'intervalle [0,1]. Les coefficients de normalisation auront alors pour expression : $Ts = \left[P_s^{-1}(1),...,P_s^{-1}(2N)\right]$.
- Rafraîchissement des entrées fréquentielles retardées des K-1 segments restants : $X_s^{(k)} = X_{s-k\alpha}^{(0)}$.
- Evaluation du bloc des sorties temporelles court terme : $\overline{y}_s = \left[0_{N\times N}\ I_{N\times N}\right]\left[g \cdot FFT_{2N}^{-1}\left(\sum_{k=0}^{K-1} X_s^{(k)} \circ H_s^{(k)}\right)\right]$. Le symbole $\circ$ représentant le produit de deux vecteurs terme à terme et g étant une fenêtre de reconstitution de longueur 2N : $g = [0_{1\times N}, g(1),...,g(N)]^T$ et satisfaisant la condition de normalisation : $\sum_{i=0}^{\alpha-1} g(n-iR) = 1$, pour $n = P,...,N+P-1$.

- Reconstitution par WOLA des R échantillons de sortie : $y_s(n) = \sum_{k=0}^{\alpha-1} z_{s-k}(n+kR)$.
- Formation du bloc fréquentiel des sorties désirées (se fait de la même manière que pour le bloc courant des entrées fréquentielles) : $D_s = FFT_{2N}\left[d(sR-2N+R),...,d(sR),...,d(sN+R-1)\right]$.
- Evaluation du signal d'erreur en fréquence : $E_s = FFT_{2N}\left(\begin{bmatrix} 0_{N\times N} & 0_{N\times N} \\ 0_{N\times N} & I_{N\times N} \end{bmatrix} FFT_{2N}^{-1}\left(D_s - \sum_{k=0}^{K-1} X_s^{(k)} \circ H_s^{(k)}\right)\right)$.
- Evaluation des termes correctifs du k^ième segment du filtre :

  Pour le cas contraint : $\nabla_s^{(k)} = FFT_{2N}\left(\begin{bmatrix} I_{N\times N} & 0_{N\times N} \\ 0_{N\times N} & 0_{N\times N} \end{bmatrix} FFT_{2N}^{-1}\left(T_s \circ \left(X_s^{(k)}\right)^* \circ E_s\right)\right)$.

  Pour le cas non contraint : $\nabla_s^{(k)} = T_s \circ \left(X_s^{(k)}\right)^* \circ E_s$.
- Mise à jour des coefficients du k^ième segment du filtre : $H_s^{(k+1)} = H_s^{(k)} + \mu\nabla_s^{(k)}$. μ étant le pas d'adaptation.

La structure générale de l'algorithme peut être décrite par la figure suivante:

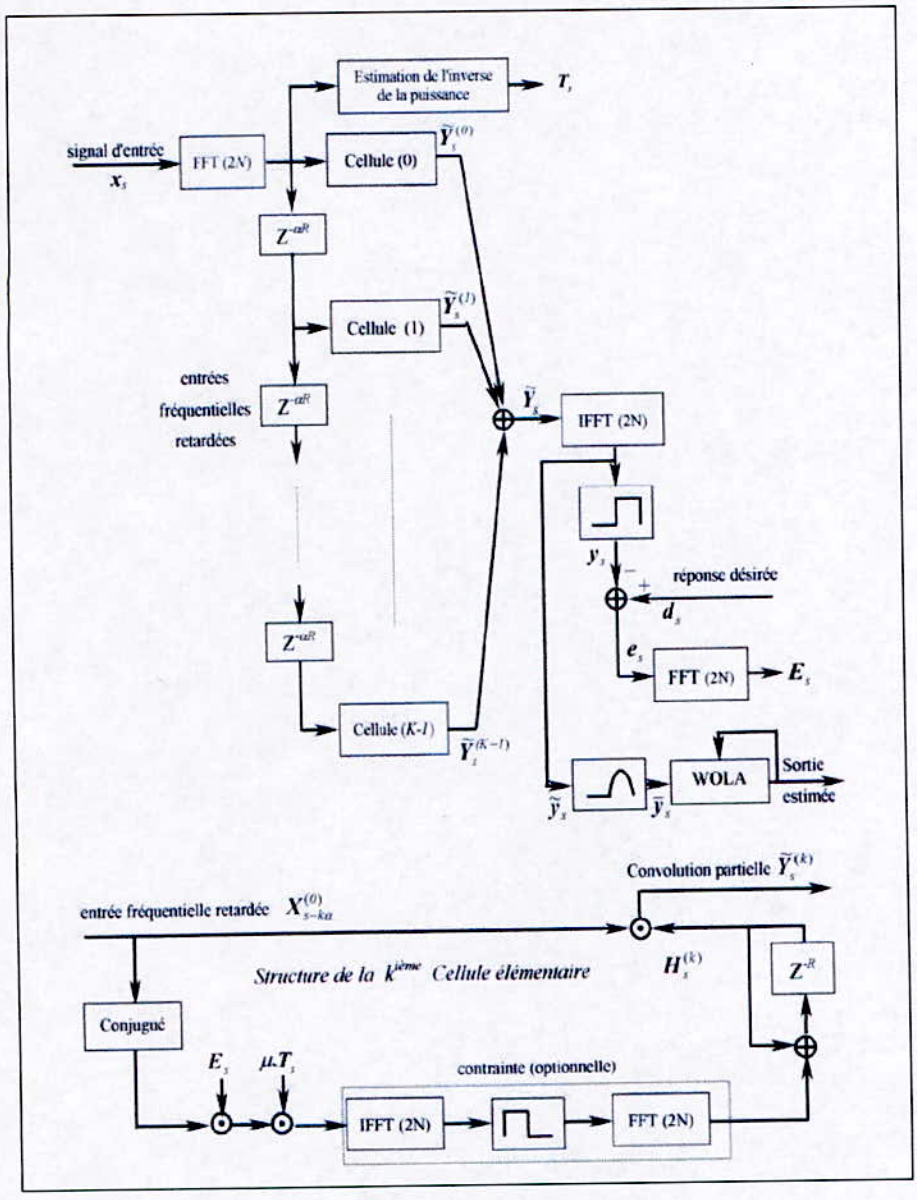


Figure 6 Schéma bloc simplifié de l'algorithme GMDFα

Avant de passer à l'implémentation de l'algorithme GMDFα proprement dite, il est impératif de maîtriser les concepts de conception VLSI, ce qui fera l'objet du chapitre suivant.

Chapitre II

TECHNIQUES D'IMPLEMENTATION VLSI

I.INTRODUCTION

Nous aborderons dans ce chapitre les techniques que l'on utilise afin de modifier la structure d'un algorithme sans en modifier la fonctionnalité.

Ces modifications peuvent avoir pour but la réduction du temps d'exécution de l'algorithme, la réduction des éléments de mémorisation ou réduction de la consommation électrique.

Les techniques citées dans ce chapitre ne seront pas forcément toutes utilisées dans la suite de ce travail, ou seront utilisées de manière implicite afin d'éviter la redondance. Elles ont été exposées ici afin de mettre en exergue l'utilité, sinon la nécessité d'utiliser les techniques VLSI avant de passer à l'implémentation, qu'elle soit hardware ou software, d'un algorithme.

II.REPRESENTATIONS GRAPHIQUES D'UN ALGORITHME

La plupart des opérations d'optimisation s'effectuent sur les graphes représentant l'algorithme ou l'exécution de celui-ci. C'est pourquoi, il faut connaître les représentations utilisées pour décrire un algorithme.

II.A.Block-diagram

Ceci est la représentation usuelle des algorithmes.

L'algorithme est représenté par des blocs fonctionnels reliés par des flèches représentant le flux de données du bloc d'entrée jusqu'au bloc de sortie, ces flèches peuvent ou non contenir des délais.

Par exemple, un filtre FIR à 3 coefficients décrit par l'expression : $y(n) = b_0 x(n) + b_1 x(n-1) + b_2 x(n-2)$, peut être représenté par :

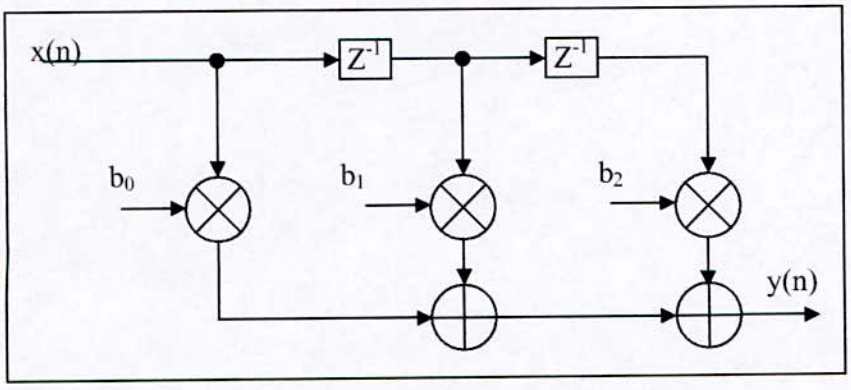


Figure 7 Bloc diagramme d'un filtre FIR à trois coefficients.

Le bloc-diagramme représente explicitement tous les blocs fonctionnels et tous les signaux d'un algorithme, plusieurs bloc-diagrammes distincts peuvent représenter différentes réalisations du même algorithme.

II.B.Signal-Flow graph

Un signal-flow graph (SFG) est un ensemble de nœuds représentant les calculs et de flèches représentant une transformation linéaire du signal à l'entrée de la flèche vers le signal à la sortie de celle-ci.

Les SFGs sont utilisés pour l'analyse, la représentation et l'évaluation des réseaux numériques linéaires.

La représentation du filtre précédent sera :

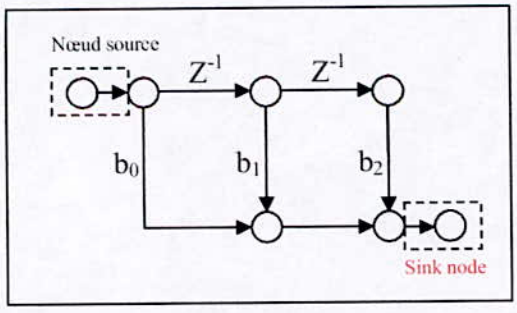


Figure 8 SFG d'un filtre FIR à trois coefficients.

II.C.Data-Flow Graph

Dans ce type de graphes, les nœuds représentent les taches ou calculs, les flèches représentent quand à elles le chemin des données. A chaque flèche est affecté un nombre positif ou nul de délais. On associe aussi à chaque nœud son temps d'exécution en terme d'unités de temps normalisé (u.t).

Un nœud dans un DFG ne peut effectuer les opérations qui lui sont associées que si toutes les données à son entrée sont disponibles. Ce qui nous mène à définir deux types de contraintes de "précédence" entre deux nœuds.

Les contraintes de "précédence" entre deux nœuds dépend du type de flèches liant ces deux nœuds. Si celle-ci ne contient pas de délais, la contrainte sera une contrainte intra itération. Par contre si elle contient des délais, elle sera une contrainte inter itération.

Les contraintes inter itération et intra itération définissent l'ordre d'exécution des nœuds.

Par exemple, dans le SFG décrivant le filtre FIR à trois coefficients, la contrainte de "précédence" intra itération impose l'exécution du nœud b_2 (les nœuds b_i correspondent à une multiplication par b_i) avant l'exécution du nœud Add_2 pour une itération k. La contrainte de "précédence" inter itération impose quand à elle l'exécution de la $k^{ième}$ itération du nœud Add_1 avant la $(k+1)^{ième}$ itération du nœud Add_2.

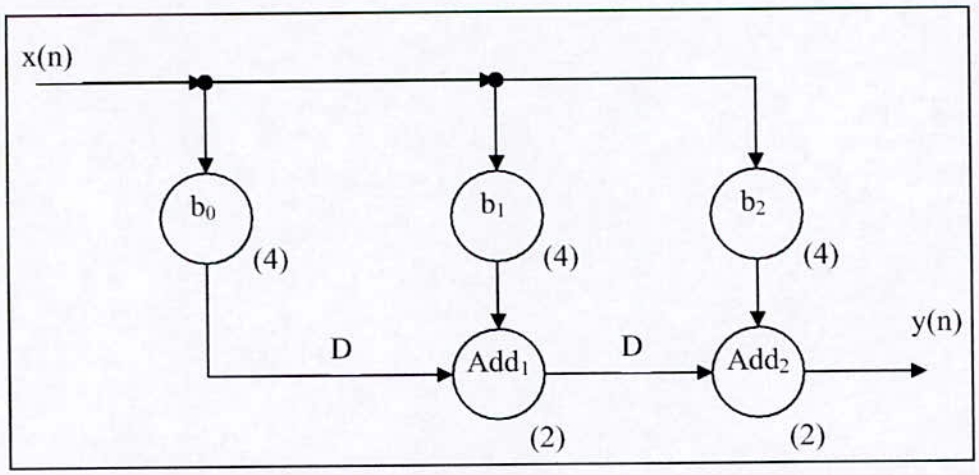


Figure 9 DFG d'un filtre FIR à trois coefficients.

Un DFG peut être décrit à différents niveaux de *granularité*, il peut être grossier si les nœuds représentent des opérations complexes (FFT, filtre,...) ou fin si les nœuds représentent des opération élémentaires (e.g. addition, multiplication, opérations logiques élémentaires).

II.D.Dependence graph (DG)

Le DFG met en exergue l'exécution d'une seule itération, les nœuds y sont ré exécutés à chaque itération.

Dans un DG, par contre, un nœud est créé à chaque exécution d'une tâche.

Un DG ne contient pas d'éléments de mémorisation (délais).

La figure suivante montre le DG d'un filtre FIR à trois coefficients :

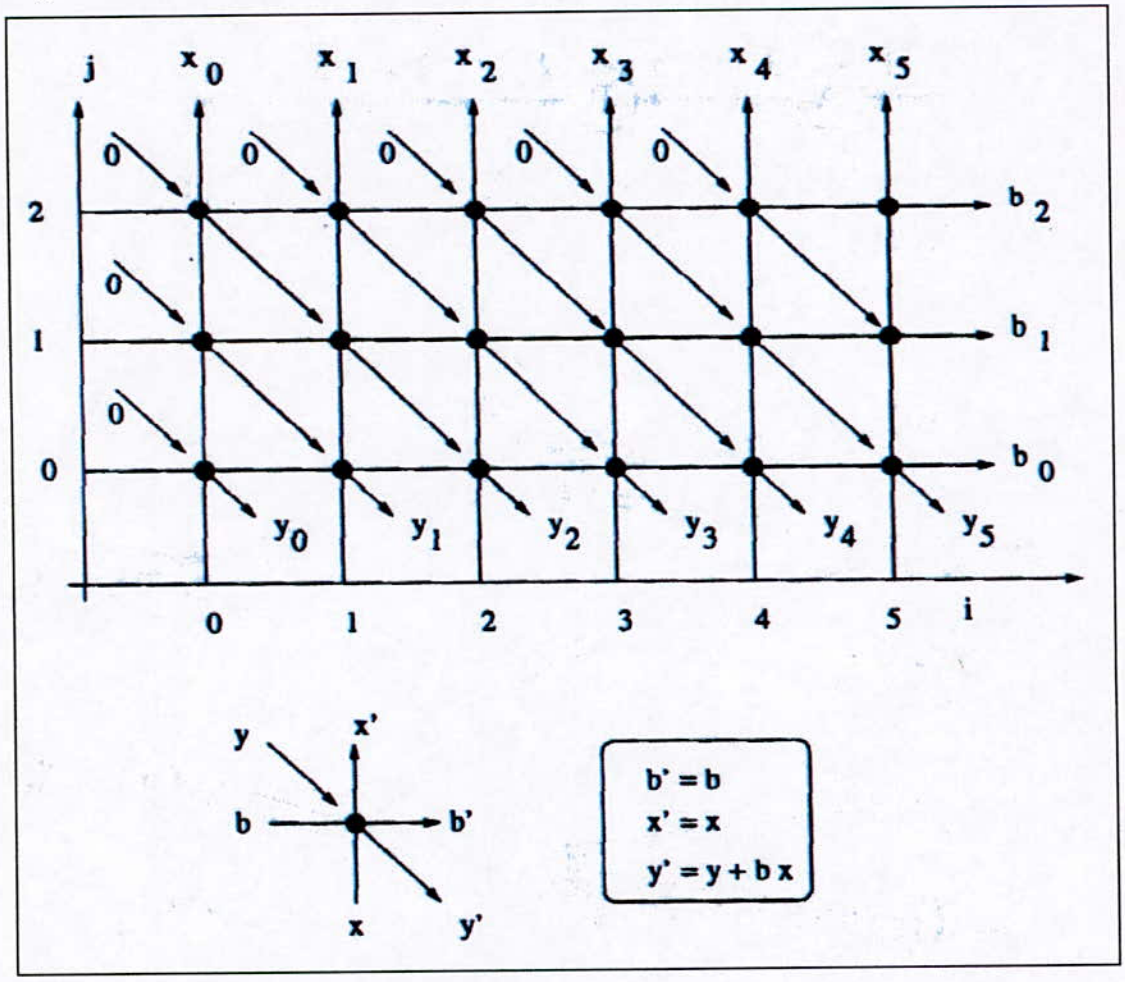


Figure 10 DG d'un filtre FIR à trois coefficients.

III. NOTIONS FONDAMENTALES

III.A. Latence

La latence (latency) est le temps nécessaire au traitement d'une donnée. C'est donc le temps qui sépare l'entrée d'une donnée et la sortie du résultat du traitement de celle-ci.

III.B. Loop bound

Une boucle (loop) est un chemin dirigé démarrant et finissant au même point.

Le loop bound est la borne inférieure de l'exécution de cette boucle. Cette limite ne peut être outrepassée même en la présence d'un nombre infini de processeurs.

Cette borne est donnée par la formule $\frac{t_l}{w_l}$ (u.t) avec t_l : durée de l'exécution de la boucle, w_l : nombre de délais dans la boucle.

Par exemple avec le DFG suivant :

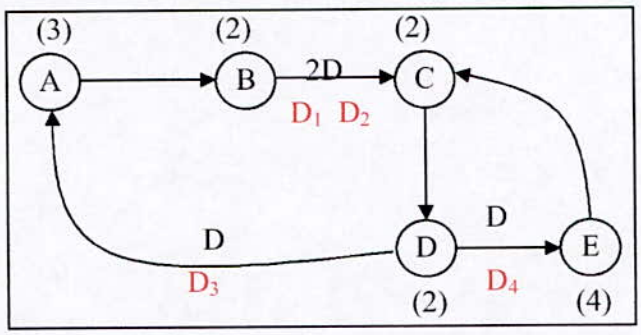


Figure 11 Exemple de DFG avec deux bloucles.

Ce DFG contient deux boucles : A=>B=>C=>D=>A et C=>D=>E=>A.

Les bornes de boucles (loop bound) seront respectivement : $\frac{t_1}{w_1} = \frac{9}{3} = 3$ et $\frac{t_2}{w_2} = \frac{8}{1} = 8$.

III.C. Chemin critique

Le chemin critique (critical path) est un chemin ne contenant aucun délai et le temps de calcul est le plus long.

Le chemin critique pour le DFG précédent est E=>C=>D, son temps d'exécution est de 8 u.t.

III.D.Boucle critique

La boucle critique (critical loop) est celle dont la borne inférieure d'exécution (loop bound) est la plus grande parmi toutes les boucles d'un DFG.

Cette boucle est critique dans le sens où elle est la plus lente à être exécutée. C'est elle qui va donc fixer la limite inférieure de la durée d'une itération (et donc de la fréquence d'échantillonnage). Cette borne est dite "iteration bound" et est notée T_∞.

D'après les définitions précédentes T_∞ sera donnée par la formule :

$T_\infty = \max_{l \in L}\left\{\frac{t_l}{w_l}\right\}$, l étant une boucle appartenant à l'ensemble des boucles d'un DFG.

Pour l'exemple précédant, la plus petite période d'échantillonnage possible sera : $T_\infty = \max\left\{\frac{9}{3}, \frac{8}{1}\right\} = \max\{3, 8\} = 8$.

III.D.1.Méthode de calcul de T_∞

Lorsque le DFG d'un algorithme contient peu de boucles, le calcul de T_∞ est relativement simple. Mais lorsque le nombre de boucles est important, il devient difficile de les énumérer toutes. On doit recourir dans ce cas à une méthode qui évite ce problème.

Nous utilisons à cet effet l'algorithme LPM (Longest Path Matrix).

III.D.1.a)L'algorithme LPM

Cet algorithme consiste en la construction de d (d : nombre de délais dans le DFG) matrices $L^{(m)}$ (m=1,…d). L'élément $l_{i,j}^{(m)}$ correspond au temps d'exécution du chemin allant du délai d_i vers le délai d_j passant par *m-1* délais (les délais *di* et *dj* non inclus), lorsqu'un tel chemin n'existe pas alors $l_{i,j}^{(m)} = -1$.

Plutôt que de construire les matrices $L^{(m)}$ (m=2…d) à partir du DFG, on utilise la relation récurrente suivante :

$l_{i,j}^{(m+1)} = \max_{k \in K}\left\{-1, l_{i,k}^{(1)} + l_{k,j}^{(m)}\right\}$. K étant un sous ensemble de {1,…,m} tel que : $l_{i,k}^{(1)} \neq -1$ et $l_{m,j}^{(m)} \neq -1$.

Après avoir calculé les matrices $L^{(m)}$, T_∞ sera donné par la relation :

$$T_\infty = \max_{i,m \in \{1,\ldots,d\}}\left\{\frac{l_{i,i}^{(m)}}{m}\right\}.$$

Un programme Matlab de l'algorithme LPM est fournit dans l'annexe III.

III.D.1.b)Exemple

Pour le DFG précédent, la matrice $L^{(1)}$ est donnée par:

$$L^{(1)} = \begin{bmatrix} -1 & 0 & 0 & -1 \\ -1 & -1 & 4 & 4 \\ 5 & -1 & -1 & 2 \\ -1 & 6 & 8 & 8 \end{bmatrix}.$$

Après exécution de l'algorithme LPM, nous obtenons les matrices $L^{(m=2,3,4)}$ suivantes :

$$L^{(2)} = \begin{bmatrix} 5 & -1 & 4 & 4 \\ 9 & 10 & 12 & 12 \\ -1 & 8 & 10 & 10 \\ 13 & 14 & 16 & 16 \end{bmatrix}, L^{(3)} = \begin{bmatrix} 9 & 10 & 12 & 12 \\ 17 & 18 & 20 & 20 \\ 15 & 16 & 18 & 18 \\ 21 & 22 & 24 & 24 \end{bmatrix}, L^{(4)} = \begin{bmatrix} 17 & 18 & 20 & 20 \\ 25 & 26 & 28 & 28 \\ 23 & 24 & 26 & 26 \\ 29 & 30 & 32 & 32 \end{bmatrix}.$$

T_∞ sera alors donnée par la relation :

$$T_\infty = \max_{i,m \in \{1,2,3,4\}}\left\{\frac{l_{i,i}^{(m)}}{m}\right\} = \max\left\{\frac{8}{1}, \frac{5}{2}, \frac{10}{2}, \frac{10}{2}, \frac{16}{2}, \frac{9}{3}, \frac{18}{3}, \frac{18}{3}, \frac{24}{3}, \frac{17}{4}, \frac{26}{4}, \frac{26}{4}, \frac{32}{4}\right\} = 8.$$

III.E.Feed-forward Cutset

III.E.1.Cutset

Un "cutset" est un ensemble de flèches d'un DFG qui si elles venaient à être enlevées, celui-ci sera disjoint.

III.E.2.Feed-Forward Cutset

C'est un "cutset" dont toutes les flèches ont la même direction.

IV.TECHNIQUES VLSI

IV.A.Pipelining et traitement parallèle

IV.A.1.Pipelining

Le pipelining est une transformation qui consiste à ajouter un latch (délai) à chaque flèche d'un "feed-forward cutset".

L'intérêt de cette méthode est de réduire le chemin critique. Mais en contrepartie, la latence ainsi que le nombre d'éléments de mémorisation augmentent.

Par exemple, pour le DFG suivant :

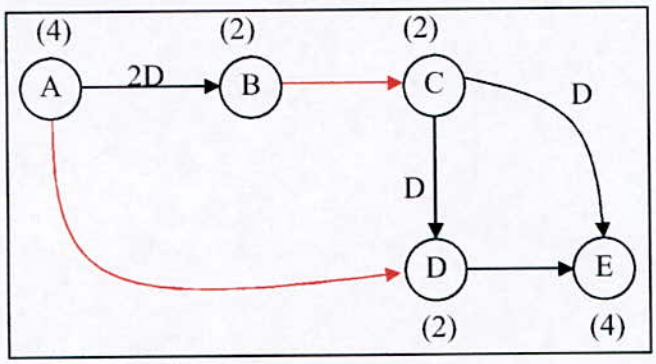


Figure 12 Exemple de feed-forward cutset.

Le chemin critique de ce DFG est A=>D=>E dont le temps d'exécution est 4+2+4=10u.t.

Les deux flèches rouges représentent un "feed-forward cutset" car si elles sont enlevées, on obtiendra deux DFG disjoints.

On ajoute donc un délais à chaque flèche afin d'obtenir une architecture pipeline à deux étages qui est illustrée par la figure suivante :

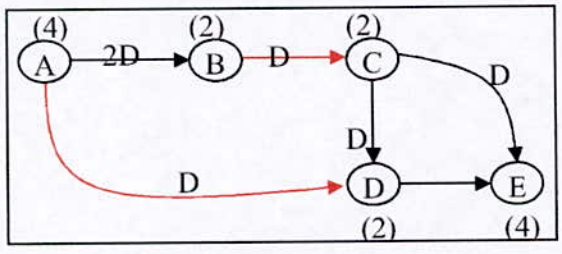


Figure 13 exemple de pipelining.

Le chemin critique sera maintenant D=>E et aura pour temps d'exécution 4+2=6u.t.

IV.A.2.Traitement parallèle

Le traitement parallèle consiste en la transformation d'un système SISO (single input single output) en un système MIMO (multiple input multiple output).

Cette opération s'effectue en dupliquant la structure initiale du système L fois et en insérant un convertisseur série parallèle en entrée et parallèle série en sortie.

Il est à noter que lors de l'insertion d'un délai dans une architecture ainsi dupliquée, revient à insérer L délais.

Cette opération ne change en rien le chemin critique de l'algorithme. Par contre, un tel système pourra traiter un bloc de L entrées à chaque cycle d'horloge et donc de produire L sorties à chaque période d'horloge. La fréquence d'échantillonnage pourra donc être multipliée par L.

IV.B.Retiming

C'est une technique basée sur le changement de l'emplacement et du nombre des délais sur un DFG sans affecter les propriétés d'entrée/sortie de l'algorithme.

Cette technique a pour but de réduire la période d'horloge du circuit, de réduire le nombre d'éléments de mémorisation ou bien de réduire la consommation électrique. Cela s'effectue essentiellement en modifiant le chemin critique du DFG de l'algorithme.

Le "retiming" transforme un graphe G en un graphe G_r.

On notera w(e) le poids (i.e le nombre de délais) de la flèche e, après "retiming", celle-ci aura pour poids $w_r(e)$.

La solution de "retiming" sera donnée par une valeur r(V) pour chaque nœud du graphe.

Le poids de la flèche $U \xrightarrow{e} V$ dans le graphe G_r sera calculé à partir du poids de celle-ci dans le graphe G par la relation :

$$w_r(e) = w(e) + r(V) - r(U).$$

Pour qu'une solution de "retiming" soit faisable, il faut que : $\forall e \in G_r : w_r(e) \geq 0$. qui représente la contrainte de faisabilité de toute solution de retiming.

Il est à noter que le "retiming" :

- Ne change pas le nombre de délais dans une boucle.
- Ne change pas l'"iteration bound".
- Le rajout d'une constante à la solution de "retiming" ne change pas la transformation du graphe G vers le graphe G_r.

IV.B.1.Techniques de retiming

IV.B.1.a)Cutset retiming

Rappelons qu'un cutset est un ensemble de flèches dans un DFG G qui, si elles sont enlevées, séparent ce dernier en deux DFGs disjoints G_1 et G_2.

Le cutset retiming consiste à ajouter k délais à chaque flèche allant de G_1 vers G_2 et à supprimer k délais de chaque flèche allant de G_2 vers G_1.

Pour que ce retiming soit faisable, il faut que les deux conditions : $\begin{cases} w_r(e_{1,2}) \geq 0 \\ w_r(e_{1,2}) \geq 0 \end{cases} \Rightarrow \begin{cases} w(e_{1,2}) + k \geq 0 \\ w(e_{1,2}) - k \geq 0 \end{cases}$ soient satisfaites pour toute flèche $e_{1,2}$ (allant de G_1 vers G_2) et $e_{2,1}$.

Ces deux conditions permettent de définir l'intervalle des valeurs possibles de k :

$$-\min_{G_1 \xrightarrow{e} G_2} \{w(e)\} \leq k \leq \min_{G_2 \xrightarrow{e} G_1} \{w(e)\}.$$

Une remarque importante s'impose dans le cas où le "cutset" est du type "feed-forward". En effet, dans ce cas, l'ajout de k délais est en fait une opération de "pipelining". Le "pipelining" est donc un cas particulier du "retiming".

IV.B.1.b)Retiming pour la minimisation de la période d'horloge

Nous présenterons dans cette section un algorithme qui permet d'effectuer un retiming en vue de la minimisation de la période d'horloge d'un circuit.

La période d'horloge minimale d'un circuit G (la notion de circuit ou de graphe peuvent être utilisées indifféremment) correspond au temps d'exécution du chemin critique. Elle peut être formulée par: $\Phi(G) = \max\{t(p) : w(p) = 0\}$.

Nous devons donc trouver une solution de retiming r_0 qui vérifie: $\Phi(G_{r0}) \leq \Phi(G_r)$ pour toute autre solution de retiming r.

Soit $W(U,V) = \min\{w(p): U \xrightarrow{p} V\}$ qui représente le chemin de U vers V contenant le moins de délais. Et $D(U,V) = \max\{t(p): U \xrightarrow{p} V \text{ et } w(p) = W(U,V)\}$ le temps de traitement le plus long parmi tous les chemins possibles de U vers V ayant le poids *W(U,V)*.

Un algorithme permettant le calcul des matrices D et W est donné par:

1. Soit $M = t_{max} n$. t_{max} étant le temps de traitement maximal de tous les nœuds du graphe G et n le nombre de nœuds.
2. Former un nouveau graphe G' qui est le même que le graphe G sauf que les poids sont remplacés par: $w'(e) = Mw(e) - t(U)$ pour toutes les flèches $U \xrightarrow{e} V$.
3. Trouver le chemin le plus court S'_{UV} (contenant le moins de délais) entre toutes les paires (U,V) de nœuds du graphe G'. La solution à ce problème peut être trouvée par l'algorithme de Floyd-Warshall présenté dans l'annexe II.
4. Si U≠V alors $W(U,V) = \left\lceil \frac{S'_{UV}}{M} \right\rceil$ et $D(U,V) = M \cdot W(U,V) - S'_{UV} + t(V)$.

   Si U=V alors $W(U,V) = 0$ et $D(U,V) = t(U)$. $\lceil x \rceil$ étant le plus petit entier supérieur à x.

L'algorithme permettant le calcul de W et de D a été implémenté sous Matlab et est fournit dans l'annexe IV.

Les valeurs de W(U,V) et D(U,V) seront utilisées pour déterminer la faisabilité d'une solution de "retiming" r permettant d'obtenir une période d'horloge désirée c vérifiant : $\Phi(G_r) \leq c$. Cette solution ne pourra exister que si:

1. La contrainte de faisabilité est vérifiée: $r(U) - r(V) \leq w(e) \forall (e = U \xrightarrow{e} V) \in G$.
2. La contrainte du chemin critique est vérifiée : $r(U) - r(V) \leq W(U,V) - 1$ pour tout nœuds U et V vérifiant : D(U,V)>c.

Ces deux contraintes nous mèneront à établir un système de N inégalités à N inconnues du type: $r_i - r_j \leq k_l$. Avec $i, j = \overline{1,N}$ et $l = \overline{1,M}$.

Ce type de système d'inégalités peut être résolue par l'algorithme décrit dans l'annexe I.

L'existence d'une solution à ce système d'inégalités indique que la période d'horloge minimale du circuit est inférieure ou égale à c. De plus, il a été démontré [1] que la valeur période d'horloge minimale est contenue dans la matrice D(U,V) et qu'il faut exclure les valeurs inférieures à t_{max}.

Les considérations précédentes nous mènent à essayer chaque valeur de D(U,V) comprises dans l'intervalle $[t_{max}, c]$. La valeur minimale de l'horloge sera soit t_{max} soit la valeur de D(U,V) directement supérieure à celle donnant un système d'inégalités insolvable.

IV.C.Unfolding

L'"unfolding" est une transformation d'un algorithme en un nouvel algorithme décrivant J itérations de l'algorithme original. On dira qu'on a effectué un J-unfolding.

Cette technique peut être appliquée pour minimiser la période d'horloge d'un circuit. Car, comme cela a été montré plus haut, la plus petite période d'horloge est égale à la borne d'itération T_∞. Toutefois, si un des nœuds de l'algorithme a un temps de calcul supérieur à T_∞, aucune solution de retiming ne peut réduire la temps de calcul du chemin critique en deçà de T_∞.

Un autre cas où l'unfolding est nécessaire dans le cas où T_∞ n'est pas un entier car le retiming seul ne permet pas de trouver de solutions fractionnelles.

Une autre application est de concevoir une architecture fonctionnant au niveau mot à partir d'une architecture fonctionnant au niveau bit.

Pour effectuer cette opération, on peut utiliser l'algorithme suivant:

1. pour chaque nœud U dans le graphe original, dessiner J nœuds $U_0, U_1,\ldots,U_{J-1}$.
2. pour chaque flèche U→V ayant w délais dans le graphe original, dessiner les J flèches $U_i \rightarrow V_{(i+w)\%J}$ avec $\left\lfloor \frac{i+w}{J} \right\rfloor$ délais pour $i = \overline{0, J-1}$. x%y étant le reste de la division de x par y et $\lfloor a \rfloor$ le plus grand entier inférieur ou égal à a.

Nous allons appliquer cet algorithme à un filtre FIR à 3 coefficients dont le DFG est le suivant:

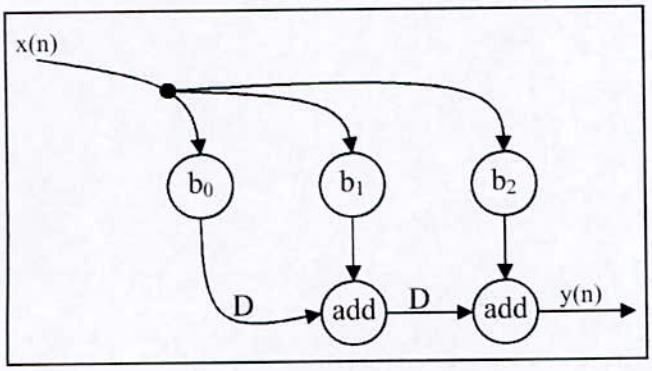


Figure 14-DFG d'un filtre FIR à trois coefficients.

Pour J=2, nous obtiendrons le DFG suivant:

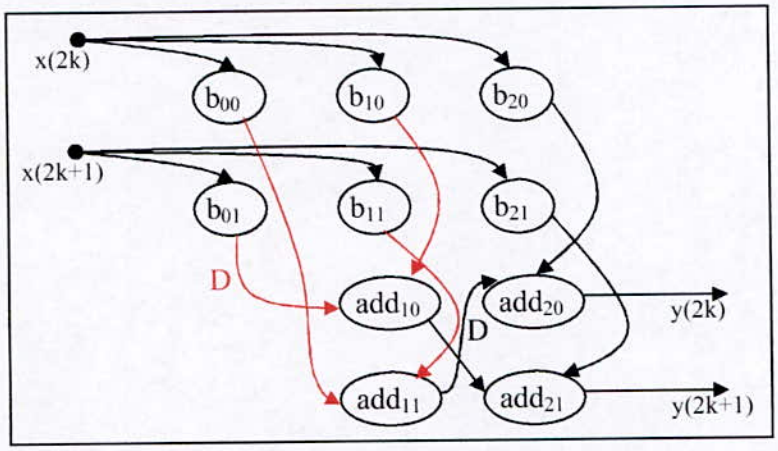


Figure 15-Filtre FIR à trois coefficients après 2-Unfolding

IV.C.1.Propriétés:

Certains points importants sont à noter en ce qui concerne l'unfolding :

- Un délai sera J fois plus lent car si l'entrée de celui-ci est x(kJ+m), sa sortie sera x((k-1)J+m)=x(kJ+m-J).
- Le nombre de délais est préservé après unfolding.
- Après unfolding, une boucle l contenant w_l délais est dupliquée gcd(w_l,J) fois et chacunes de ces boucles contiendra $\frac{w_l}{\gcd(w_l, J)}$ délais et $\frac{J}{\gcd(w_l, J)}$ copies de chaque nœud qui apparaît dans l. gcd(a,b) étant le plus grand commun diviseur de a et b.

L'exemple suivant indique la différence entre un unfolding avec J=3 (b) et un unfolding avec J=4 (c) où le graphe original (a) contient une seule boucle avec w_I=6 :

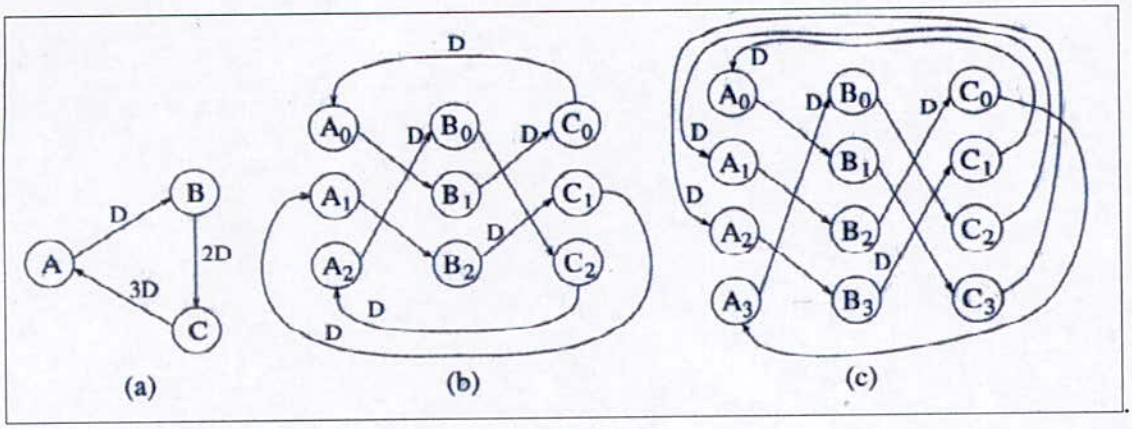


Figure 16-Illustration des propriétés de l'unfolding.

- De l'unfolding d'un DFG ayant une borne d'itération (iteration bound) T_∞ résultera un DFG ayant une borne d'itération $J \cdot T_\infty$.

IV.C.2.Relation entre Unfolding, Retiming et chemin critique:

IV.C.2.a)Propriété:

Soit un chemin ayant w délais avec $w<J$. Du J-unfolding de ce chemin résultera J-w chemins sans délais et w chemins avec 1 délai.

IV.C.2.b)Conséquence :

Tout chemin contenant J délais ou plus dans le DFG original produira, suite au J-unfolding au moins J chemins avec au moins un délai dans chaque chemin. En conséquence, un chemin ayant J délais ou plus dans le DFG original ne pourra créer de chemin critique après J-unfolding.

IV.C.2.c)Lemme :

Toute période d'horloge que l'on peut obtenir en effectuant un retiming du DFG obtenu après un J-unfolding, peut être obtenue en effectuant un retiming du DFG original puis en effectuant un J-unfolding de ce dernier.

IV.D.Folding:

Cette technique est utilisée pour réduire la surface des circuits. Elle a pour principe de minimiser le nombre d'unités fonctionnelles (e.g. additionneurs, multiplieurs, registres, multiplexeurs et lignes d'interconnexion) d'un algorithme. Cette réduction se fait en opérant un multiplexage temporel entre les opérations similaires d'un algorithme est opéré afin que celles-ci soient effectuées par la même unité fonctionnelle.

L'exemple suivant montre comment une architecture qui comprenait deux additionneurs est transformée en une architecture ayant un additionneur et un étage de pipelining.

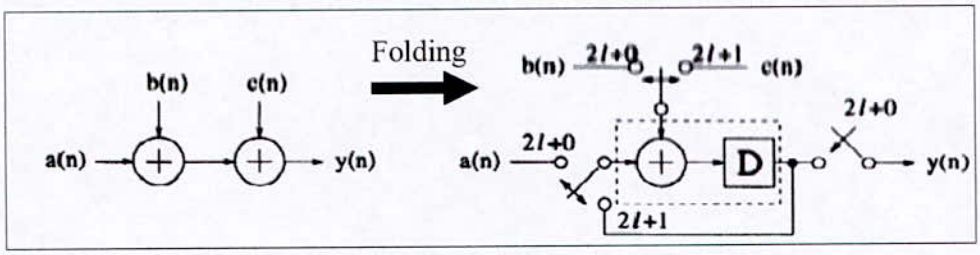


Figure 17-Exemple de folding.

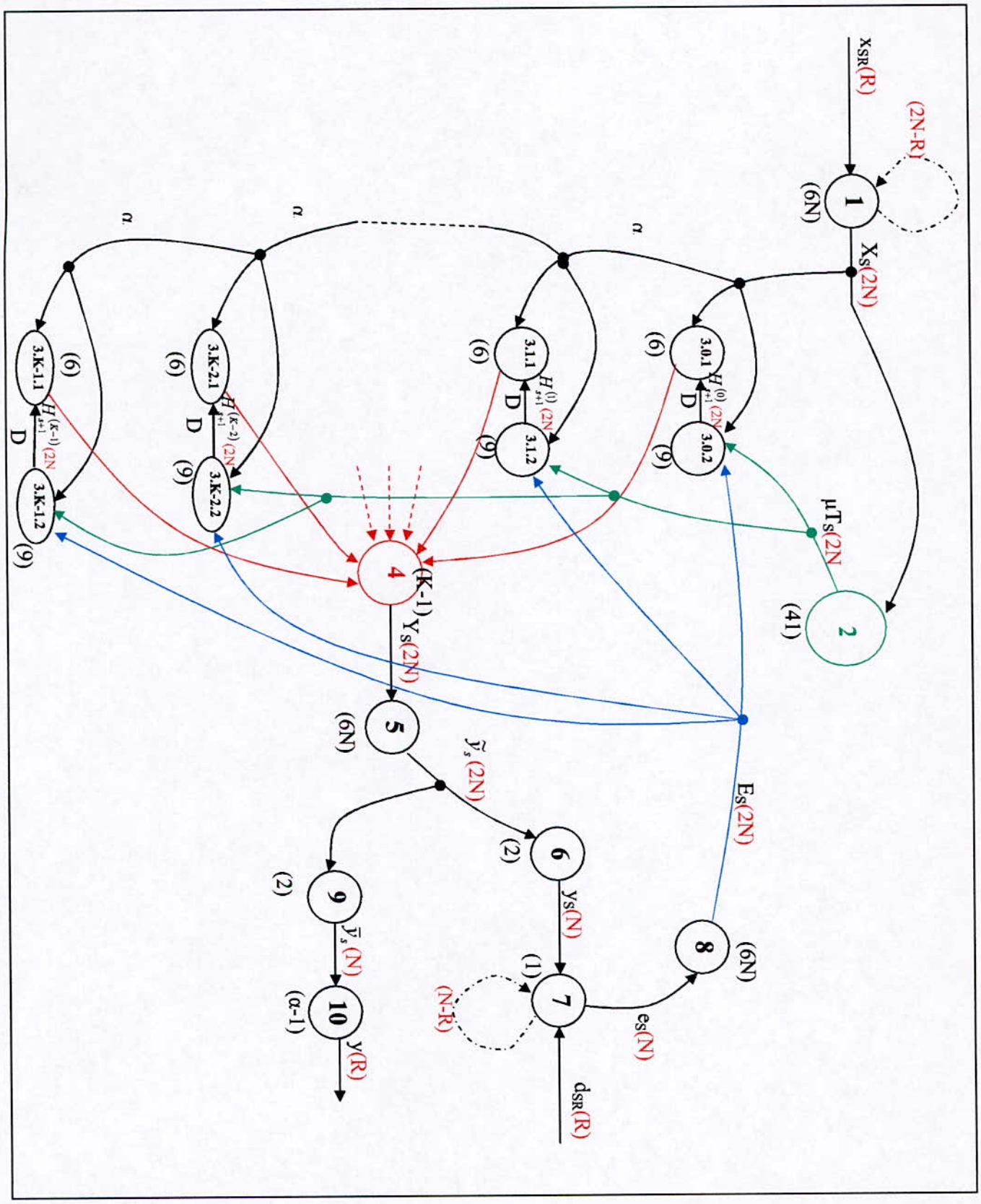
Le tableau suivant montre le fonctionnement du circuit obtenu durant les six premières périodes d'horloge.

Cycle	Entrée gauche de l'additionneur	Entrée haute de l'additionneur	Sortie du système.
0	$a(0)$	$b(0)$	–
1	$a(0)+b(0)$	$c(0)$	–
2	$a(1)$	$b(1)$	$a(0)+b(0)+c(0)$
3	$a(1)+b(1)$	$c(1)$	–
4	$a(2)$	$b(2)$	$a(1)+b(1)+c(1)$
5	$a(2)+b(2)$	$c(2)$	–

Tableau 1-Fonctionnement après Folding

V.APPLICATION A L'ALGORITHME GMDFα :

Tout d'abord, nous devons "dessiner" le DFG de l'algorithme GMDFα.


xSR(R)
(2N-R)
1
(6N)
Xs(2N)
α
3.0.1
3.0.2
3.1.1
3.1.2
3.K-2.1
3.K-2.2
3.K-1.1
3.K-1.2
(6)
(9)
D
μTs(2N
2
(41)
4
(K-1)
Ys(2N)
5
(6N)
Es(2N)
8
(6N)
6
(2)
ys(N)
7
(1)
(N-R)
es(N)
dSR(R)
9
(2)
10
(α-1)
y(R)

La nomenclature des nœuds est la suivante:

- 1 et 8 correspondent à une FFT à 2N points.
- 2 correspond au bloc de calcul des coefficients de normalisation.
- 3.k.1 correspond au produit du signal d'entrée par les coefficients du $k^{ième}$ segment du filtre.
- 3.k.2 correspond au bloc de mise à jour des coefficients du $k^{ième}$ segment du filtre.
- 4 correspond à l'addition des k produits partiels.
- 5 correspond à une FFT inverse sur 2N point.
- 6 correspond à un fenêtrage permettent d'isoler les N derniers termes d'un bloc de 2N échatillons.
- 7 est un soustracteur utilisé pour l'évaluation du signal d'erreur.
- 9 est une fenêtre pondérée respectant la condition de normalisation.
- 10 est un bloc d'addition recouvrement.

Il est à noter qu'à chaque itération, chaque nœud devra traiter un bloc de données, la taille du bloc d'entrée ainsi que la taille du bloc généré par chaque nœud est indiquée en rouge entre parenthèses.

V.A. Calcul du chemin critique:

Le chemin critique du DFG représentant l'algorithme GMDFα est celui passant successivement par les nœuds 1→3.0.1→4→5→6→7→8→3.0.2.

Le temps d'exécution du chemin critique sera donc: T=6N+6+K-1+6N+2+1+6N+9 donc: T=18N+K+17.

Ce temps d'exécution correspond à un échantillon parmi les 2N échantillons formant le bloc d'entrée, le temps d'exécution correspondant au traitement d'un bloc sera donc :

$T = \left(2N\left(6N+6+K-1+6N+6N+9\right)+N\left(2+1\right)\right)T_{clk} = \left(32N^2+2NK+31N\right)T_{clk}$ qui doit être inférieure où égale au temps necessaire pour l'acquisition d'un nouveau bloc qui est égal à $T_{acq}=RT_{sample}$, T_{sample} étant la période d'échantillonnage.

Ce qui nous mène à : $\left(32N^2+2NK+31N\right)T_{clk} \le RT_{sample}$. En prenant en compte les valeurs choisies : $\left(32\cdot 128^2+2\cdot 128\cdot 8+31\cdot 128\right)T_{clk} \le 32T_{sample} \Rightarrow T_{clk} \le 60.28\cdot 10^{-6} T_{sample}$.

Pour un période d'échantillonnage T_{sample}=62.5µs (qui correspond à une fréquence d'échantillonnage de 16kHz) il faudra que $T_{clk} \le 3.77\text{x}10^{-9}$. ce qui correspond à un période d'horloge de 265.4Mhz.

Cette période d'horloge est relativement élevée pour les circuits FPGA cible de notre implémentation.

Nous devons donc utiliser les techniques VLSI précédemment acquises afin de minimiser le chemin critique et donc de réduire la fréquence de fonctionnement du circuit.

Toutefois, vu la complexité, l'application des techniques sus citées nécessite un traitement automatisé par ordinateur. Nous nous sommes donc lancés dans le développement d'une toolbox sous Matlab qui servira une fois finalisée à appliquer les techniques d'optimisation VLSI.

La seule proposition raisonnable d'optimisation qui nous ait venu à l'esprit était d'exploiter le fait que pendant le calcul de la FFT du signal d'entrée, le reste du circuit est "au repos", nous pouvons donc effectuer par exemple le calcul des 6N dernière opérations du chemin de calcul de l'erreur dans le domaine fréquentiel et de la mise à jour des coefficients du filtre en parallèle avec le calcul de la FFT du prochain bloc.

Cette opération, qui correspond à un retiming, permettra un gain de $6NxT_{clk}$ par échantillon traité, ce qui correspond à un gain de $12N^2 xT_{clk}$ par bloc d'échantillons traités.

Dans ces conditions, nous pourrons effectuer le traitement d'un bloc dans un temps de : $T_B = (20N^2 + 2NK + 31N)T_{clk}$. Pour une fréquence d'échantillonnage de 16kHz, nous aurons alors une fréquence de fonctionnement de 166.8Mhz. Ce qui représente déjà un gain de 40%.

Pour notre part, nous utiliserons une fréquence d'échantillonnage de 8kHz qui imposera une fréquence interne d'au moins 83.4Mhz.

Nous utiliserons en plus dans notre architecture plusieurs horloges de périodes différentes, ce qui nous permettra d'effectuer certaines opérations plus vite que d'autres, le choix se faisant en effectuant une analyse des temps de traitement de chaque bloc en examinant le datasheet du circuit cible et les informations fournies par l'auteur des "cores" utilisés.

Chapitre III

METHODOLOGIE ET OUTILS DE CONCEPTION

I.INTRODUCTION :

Une fois que l'algorithme à implémenter est « mis en forme » en vue d'une implémentation hardware, on passe à l'étape de l'implémentation proprement dite.

Ce travail d'implémentation nécessite un effort considérable de la part du concepteur, car il doit satisfaire en même temps les contraintes du cahier des charges et celles du marché (temps et coût de développement réduits). Il doit donc faire vite et bien. Cela implique qu'il doit adopter une méthodologie de design lui permettant de passer rapidement de l'idée à l'implémentation tout en assurant la détection/correction rapide des éventuels bugs. De plus, afin de pouvoir mettre en œuvre cette méthodologie, il doit posséder des outils logiciels suffisamment complets pour pouvoir gérer un design du début jusqu'à la fin.

Nous aborderons donc dans ce chapitre, la méthodologie dite top-down, ainsi que les outils de conception que nous avons utilisés.

II.METHODOLOGIE TOP-DOWN :

Traditionnellement, les ingénieurs chargés de l'implémentation hardware commencent par écrire la spécification du projet, ils utilisent ensuite cette spécification pour partitionner le projet en blocs fonctionnels. Ils continuent ensuite à implémenter de façon itérative le projet mais ne vérifient que très rarement où pas du tout la consistance avec la spécification initiale.

La méthodologie top-down (figure), par contre, consiste à démarrer la conception à partir d'un haut niveau d'abstraction. Puis, une fois que la consistance de cette description est vérifiée avec la spécification issue de la validation du niveau qui lui est supérieur, on écrit la spécification du niveau suivant, on l'implémente, vérifie sa consistance avec la spécification précédemment écrite, puis on écrit la spécification du niveau suivant et ainsi de suite.

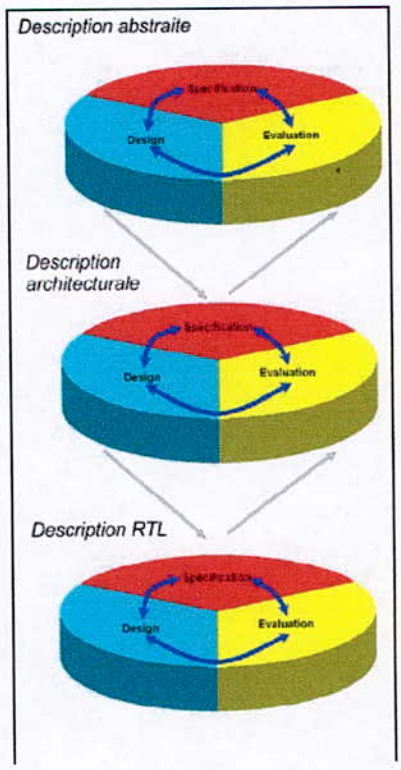


Figure 18- Principe de la méthodologie Top-Down

Les étapes à suivre sont les suivantes :

- Extraction à partir du cahier des charges (requirement specification) d'un modèle fonctionnel de haut niveau d'abstraction. Ce modèle subdivise le projet en sous

blocs fonctionnels et décrit les interconnexions entre eux. La consistance de ce modèle fonctionnel avec le cahier des charges sera vérifiée et les éventuelles omissions ou ambiguïtés corrigées. En même temps, doit être définie la spécification des tests ; cette dernière permet de définir la manière dont sera testé le design. Puis, à partir de cette spécification, on écrit un test bench (ou banc d'essais) qui est le programme de test proprement dit. Ce test bench sera utilisé afin de valider chaque étape du design.

- Le modèle fonctionnel sera utilisé comme spécification pour définir le modèle comportemental du système. Le modèle comportemental devra alors être validé par le test bench écrit précédemment. Cette validation assure la consistance avec le modèle fonctionnel et donc avec le cahier des charges initial. Une fois ce modèle validé, on écrit la spécification comportementale du projet.
- L'étape suivante consiste à "épurer" le code HDL (Hardware Description Language) comportemental généré à partir du modèle comportemental afin de le rendre synthétisable. La description peut être maintenant un mélange de code comportemental et RTL (niveau registre). Ce code est ensuite synthétisé[1] et optimisé pour la technologie cible tout en prenant en compte les contraintes de taille et les informations sur l'horloge. Ensuite, le code est simulé pour assurer la consistance de ce dernier avec le modèle comportemental. De plus les résultats de l'optimisation sont comparés aux contraintes physiques (e.g. espace, horloge) définies dans la spécification initiale du projet.
- Enfin, vient la simulation de la description niveau portes ; cette simulation prend en compte les délais de propagation des signaux électrique. En cas d'inconsistance des résultats de cette simulation avec le modèle RTL, il est nécessaire de modifier ce dernier et de refaire la synthèse. Une fois cette étape validée, les données de fabrications sont générées.

La figure suivante détaille le cycle de conception basé sur la méthodologie top-down.

[1] La synthèse, consiste à écrire une liste des interconnexions (netlist) entre les primitives de la technologie cible et ce dans un format standard

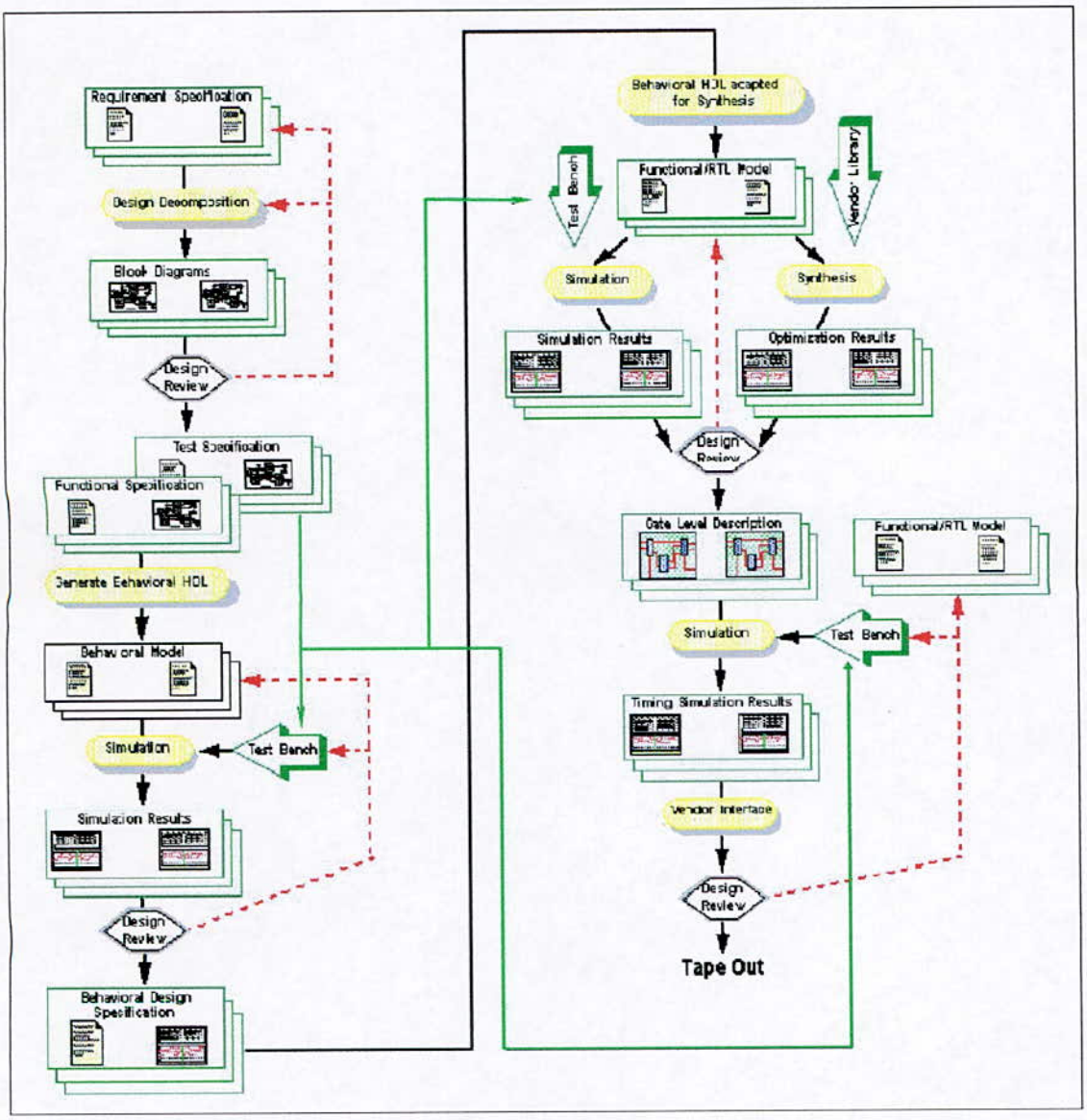


Figure 19-Méthodologie Top-Down.

En plus de ce cycle de conception, il est à noter que lors de l'écriture du modèle comportemental, il est recommandé de prendre en compte la technologie cible, cela a pour but de simplifier le passage du modèle comportemental vers le modèle RTL.

De plus, il faut essayer d'écrire un test bench le plus détaillé possible, et ce, avant de commencer l'implémentation du bloc à valider. Cela a pour objectif de ne pas influencer l'écriture du test bench par des détails concernant l'implémentation du bloc à valider afin de ne pas répercuter des erreurs ou des incompréhensions dans le test bench. Il est en plus fortement recommandé de confier l'écriture du test bench à une personne différente de celle qui implémente le bloc à valider.

Un test bench doit en outre couvrir un maximum de cas possibles pour le bloc à tester et surtout en ce qui concerne les cas limites, mais sans pour autant consommer un trop grand temps de simulation. Une analyse supplémentaire appelée *code coverage* permet de tester le degré de couverture d'un test bench des états que peut prendre le bloc à tester. Un autre outil nommé Performance analyser permet d'effectuer des statistiques sur les portions de code les plus souvent exécutées, cela permet d'identifier les goulots d'étranglement et ainsi de modifier le test bench en conséquence. Cette analyse permet d'optimiser les temps de simulation et d'obtenir des gains pouvant atteindre 75%.

III.ENVIRONNEMENT DE DEVELOPPEMENT :

Pour mettre en œuvre notre projet nous avons besoin:

- D'outils permettant d'exprimer les idées facilement dans l'environnement, d'un logiciel de simulation pour la validation et d'un logiciel de synthèse.
- D'outils fournis par le constructeur permettant de générer les données de fabrication (traduction de la netlist, mapping, placement et routage), de faire une analyse temporelle statique (calcul des délais de propagation, détermination du chemin critique (critical path)), d'analyser la consommation énergétique du circuit, génération du modèle de simulation temporelle (écrit en HDL) du circuit final.
- D'une bibliothèque IP (intellectual property) qui contient des fonctions déjà implémentées, fournies en général sous forme de netlist que l'on intègre directement dans le design.

Nous avons choisis les logiciels des leaders mondiaux dans les deux premières catégories, il s'agit dans l'ordre du logiciel FPGA Advantage de chez Mentor graphics et de Xilinx Foundation/Alliance series de chez Xilinx. En ce qui concerne la bibliothèque IP, nous utilisons celle fournie par Xilinx par l'intermédiaire du logiciel Xilinx CoreGenerator.

III.A.FPGA Advantage :

FPGA Advantage est un ensemble de logiciels orientés vers l'implémentation sur circuits FPGA avec une option ASIC pour la version la plus évoluée.

Trois logiciels sont fournis :

III.A.1.HDL Designer series :

HDL Designer series est un logiciel permettant la gestion automatisée de projets de systèmes électroniques.

Plusieurs versions de HDL Designer series existent (HDL pilot, HDL Detective, HDL Author ou HDL Designer). Pour notre part, nous utilisons HDL Designer-Pro qui est la version la plus complète.

Chaque projet est symbolisé par une librairie.

Le niveau d'abstraction le plus haut peut être représenté par un block diagram, une IBD[2] view ou alors du texte HDL.

Cette vue peut être décomposée en composants ou blocks décrits en utilisant une machine d'état, une table de vérité, un flow chart ou du code HDL comportemental.

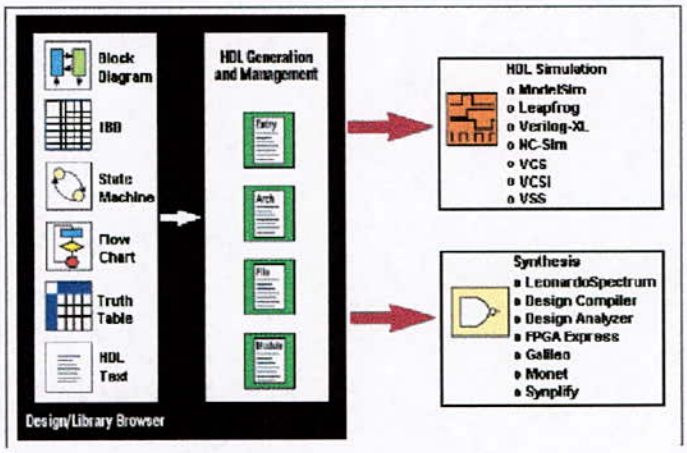


Figure 20 Représentation simplifiée de l'interaction d'HDL Designer Series avec les autres logiciels.

Il est à noter que plusieurs vues[3] peuvent définir un composant/block, le logiciel prenant en compte la vue par défaut. Par exemple, un composant peut avoir une vue sous forme graphique et une autre le représentant après implémentation.

Une fois les différentes vues décrites sous format graphique ou textuel, le code HDL peut être automatiquement généré puis compilé en vue de la synthèse ou de la simulation.

HDL designer series contient une interface avec les outils de gestion des versions (qui permettent de sauvegarder plusieurs versions de la même vue).

HDL Designer series permet une grande intégration avec les outils externes (downstream tools) de compilation, de synthèse ou de simulation. Permettant par exemple de suivre l'exécution de la simulation d'une vue graphique directement dans la fenêtre décrivant celle-ci.

---

[2] Interface Based Design : c'est une représentation sous forme tabulaire des interconnections entre les blocs d'un design

[3] Une vue est la description d'une unité de conception, plusieurs vues du même composant ou bloc peuvent coexister, celles-ci décrivent des implémentations alternatives du bloc en question.

III.A.2.Leonardo Spectrum :

Leonardo Spectrum est un logiciel de synthèse. Il permet en plus de définir les contraintes temporelles (de l'horloge par exemple) et offre de nombreuses fonctions d'optimisation. Il permet aussi de visualiser le schéma de niveau RTL et aussi le schéma final (celui qui décrit le projet sous forme de primitives de la technologie cible).

III.A.3.ModelSim :

ModelSim est un logiciel de simulation de code HDL. Il permet en outre d'afficher les résultats sous forme de chronogrammes ou de listes, d'effectuer des analyses de code coverage (qui permet d'évaluer le degrés de couverture d'un test bench), d'effectuer une comparaison entre différentes versions de chronogrammes (par exemple, de comparer les signaux issus de la simulation d'un code comportemental et ceux issus de la simulation post-implémentation).

III.B.Xilinx ISE Alliance/Foundation :

Xilinx ISE fournit un environnement de développement complet orienté vers l'implémentation sur les circuits programmables de chez Xilinx.

Il permet de :

- Définir le projet sous forme de code HDL, de block diagram, de machines d'état.
- Intégrer directement un Core issu du Xilinx Core Generator.
- Définir les contraintes temporelles, de placement, assignation des broches.
- Xilinx ISE contient aussi une interface vers les logiciels de synthèse et de simulation.
- Effectuer les opérations d'implémentation.
- Effectuer une analyse temporelle statique.
- Analyser la consommation énergétique du circuit final.
- Générer le modèle de simulation du circuit final afin d'effectuer la simulation temporelle.

Pour notre part, nous allons utiliser Xilinx ISE juste pour l'implémentation et les opérations s'y rapportant (e.g. analyse temporelle statique, génération du modèle de simulation temporelle).

La figure suivante montre la manière dont sont utilisés les logiciels sus-cités :

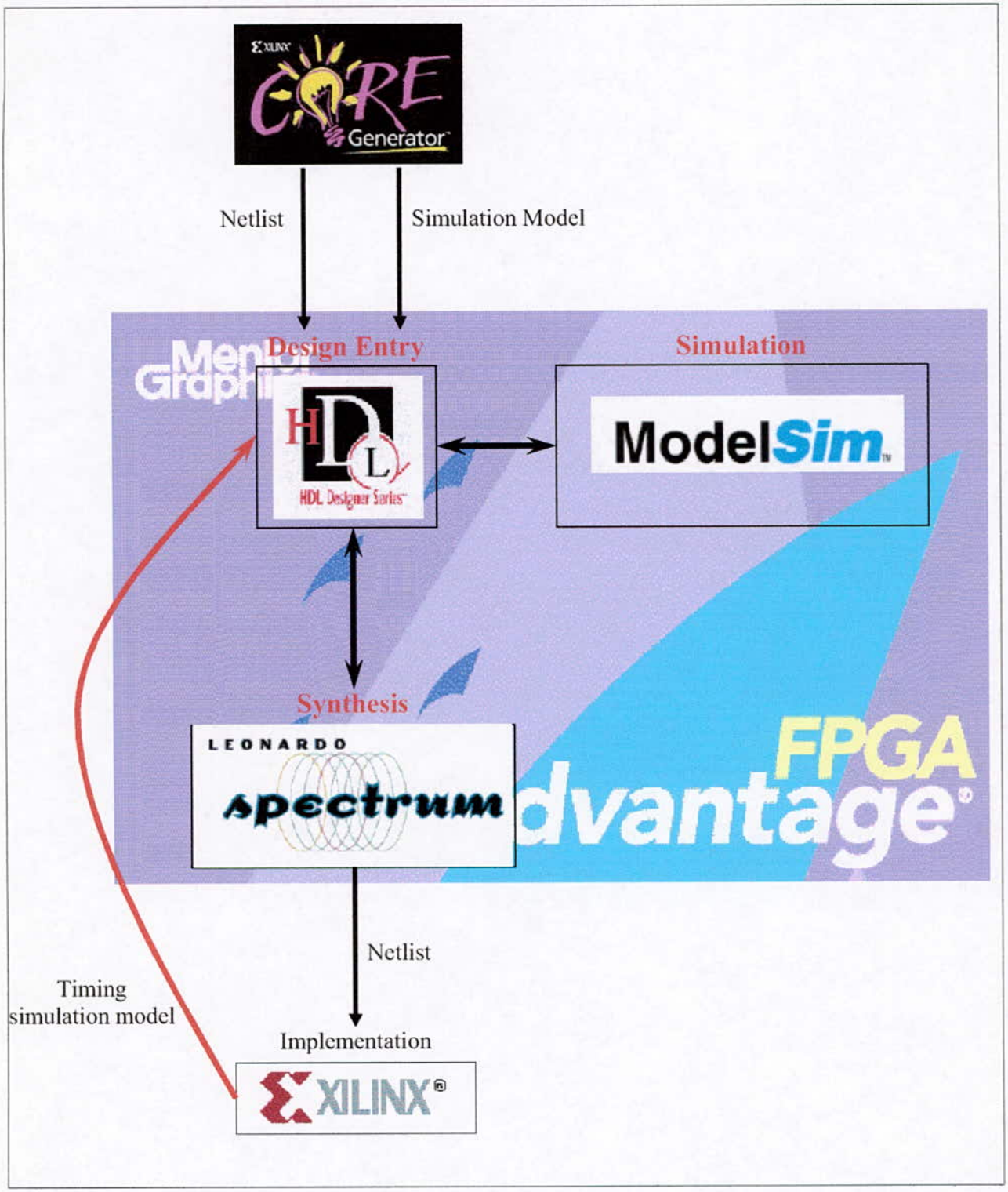


Figure 21-Infrastructure logicielle utilisée.

IV. LE LANGAGE VHDL :

IV.A. Bref Historique :

Le VHDL (VHSIC (Very High Speed Integrated Circuit) Hardware Description Language) a été développé par le DOD (Département de la Défense Américaine) pour décrire les circuits complexes de manière à établir un langage commun entre les fournisseurs. C'est un standard IEEE 1076 depuis 1987, qui aurait du assurer la portabilité du code entre les différents outils de travail (simulation, synthèse pour tous les circuits et tous les fabricants).

Une mise à jours du langage VHDL s'est faite en 1993 ((IEEE 1164) et en 1996, la norme 1076.3 a permis de standardiser la synthèse VHDL.

IV.B. Utilité du VHDL :

Le VHDL est un langage de spécification, de simulation et également de conception.

Mais contrairement à d'autres langages, la normalisation s'est faite d'abord pour la spécification et la simulation (1987) et ensuite pour la synthèse.

Le VHDL comporte de nombreux avantages par rapport à d'autres HDLs :

- Le langage peut être utilisé comme intermédiaire entre différents outils CAD et CAE.
- Le langage accepte les descriptions hiérarchiques.
- Le langage accepte plusieurs méthodologies de design : Top-Down, Bottom-Up ou mixte.
- Il accepte les models temporels synchrones et asynchrones.
- Différentes techniques de modélisation des systèmes digitaux sont supportées comme les machines d'états finis, description algorithmique et équations booléennes.
- Le langage est public, intelligible et non propriétaire.
- Le langage comporte trois styles de description : Structurelle, comportementale et par flot de données.
- Il n'y a pas de limites en ce qui concerne la taille des systèmes développés.
- Le modèle peut contenir aussi des informations sur le design lui même (notion d'attributs) comme la taille et la vitesse.

- La possibilité de définir de nouveaux types de données, ce qui permet la conception des systèmes avec un haut niveau d'abstraction sans se soucier des détails d'implémentation du circuit final.

IV.C.Représentation d'un système en VHDL :

L'interface d'un système est décrite par son ENTITE.

La fonction réalisée par le système est donnée par son ARCHITECTURE.

On peut enrichir le langage par des concepts non inclus dans le standard VHDL, on peut aussi inclure des PACKAGES (ensemble de fonctions prêtes à l'emploi).

IV.C.1.Entité :

La déclaration d'une entité se fait comme suit :

```
Entity <name> is
  Generic (…);
  Port (…);
End [entity] [name] ;
```

Le mot clé `generic` est utilisé pour déclarer des paramètres constants qui seront fournis par l'environnement au système, par exemple : la taille d'un bus ; le nombre de boucles qu'un compteur interne doit effectuer.

```
Generic (<gen1> :< type> [:=value] ;
        <gen2>: :< type> [:=value];
        …) ;
```

Le mot clé `port` est utilisé pour déclarer les ports d'entrée/sortie du système.

```
Port ([signal] <name11>, <name12>, : <in|out|inout|buffer>
<type&size> [:=value] ;
      [Signal]<name21>,  <name22>,  :  <in|out|inout|buffer>
<type&size> [:=value];
      …);
```

IV.C.2.Architecture :

```
Architecture <name> of <entity_name> is
…
 end [architecture] [name];
```

Il faut mettre en évidence le fait que le VHDL est un langage destiné à la conception de systèmes électroniques et non pas un langage informatique. En effet une des notions les plus importantes ici est la notion de concurrence (i.e. la possibilité d'exécution de plusieurs blocs de code simultanément), car une porte logique par exemple change d'état à chaque fois que l'une de ses entrées change d'état.

La notion de séquence existe toutefois en VHDL à travers les processus.

On peut aussi décrire une architecture d'un point de vue structurel, comportemental ou par flots de données.

Maintenant que nous maîtrisons les aspects inhérents aux techniques d'optimisation des algorithmes et de l'implémentation, nous pouvons passer à l'implémentation de l'algorithme GMDFα proprement dite.

Chapitre IV

IMPLEMENTATION ET SIMULATIONS

I.CONSIDERATIONS SUR L'IMPLEMENTATION :

Pour l'implémentation de l'algorithme GMDFα non contraint, nous avons effectués les choix suivants:

- Le signal d'entrée sera codé suivant le format 10bit virgule fixe signé (notation complément à deux) dont la partie fractionnelle comprend 9bits. Les opérations seront effectuées sur 16bits en interne.
- La taille de la FFT est de 256 points. Ce qui impose que N=128.
- α ainsi que R sont des puissances de 2. Ce qui fait que: $N = \alpha^{\alpha\_w} \cdot R^{r\_w}$, α_w et r_w étant deux paramètres qui définiront la taille des bus de α et R respectivement. On choisira α=4 ce qui impose R=32.
- La taille de la réponse impulsionnelle à estimer est de 1024 coefficients. Ce qui impose K=1024/128=8.

Malgré ces choix, nous avons tachés de rendre le design le plus configurable possible en définissant les paramètres génériques α_w, R_w et K.

II.PARTITIONNEMENT DU DESIGN:

Une vue d'ensemble du circuit final peut être décrite par la figure suivante:

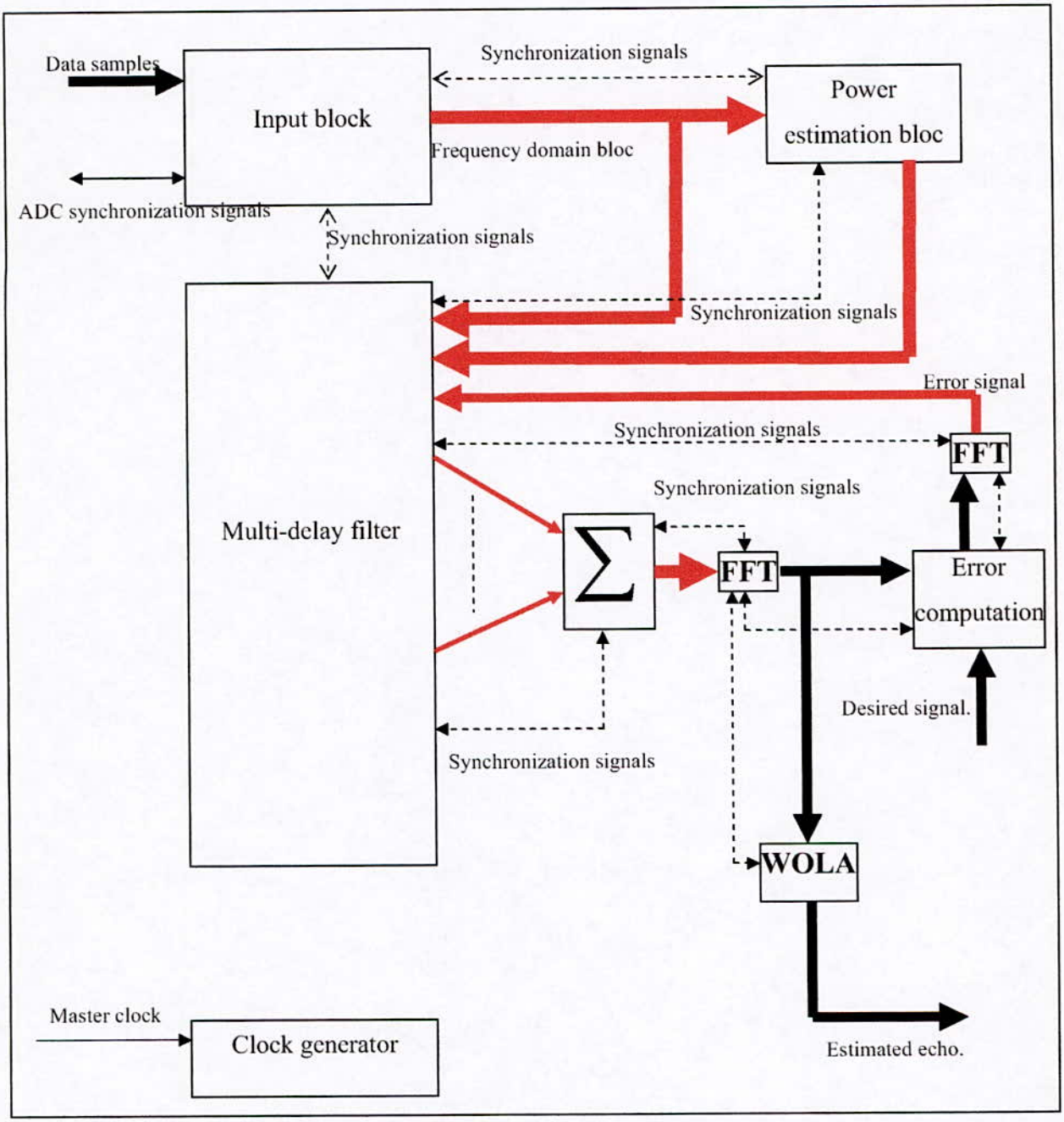


Figure 22-Partitionnement de l'algorithme GMDFα en vue de l'implémentation.

Un ensemble de signaux de synchronisation relieront les différents blocs du circuit, ces circuits de synchronisation permettront à chaque circuit de recevoir les données à traiter et de notifier au circuit suivant la fin de traitement. Cela a pour but que chaque bloc ait sa propre horloge et sa propre unité de contrôle. Un générateur d'horloges dérivera les horloges nécessaires à chaque bloc à partir de l'horloge principale du circuit. Ce dernier est formé d'un ensemble de DCMs (voir annexe V).

Dans le cadre de l'implémentation de l'algorithme GMDFα, nous n'avons implémentés que le bloc "Input Bloc". Cela est du aux raisons que nous allons citer en conclusion de ce rapport.

III. IMPLEMENTATION DU BLOC "INPUT BLOCK":

III.A. Spécification :

Ce bloc doit effectuer l'acquisition de R nouveau échantillons à partir d'un ADC, de compléter ce bloc par les 2N-R échantillons précédents puis d'effectuer une FFT sur le bloc ainsi obtenu. Les échantillons arrivant à la cadence d'acquisition du ADC.

III.B. Conception :

Au fur et à mesure que les échantillons arrivent, le bloc acq_ctrl les capturent et les estoquent dans la mémoire ram_8 qui est une ram à deux ports d'entrée sortie obtenue par l'intérmédiaire du Core Generator de Xilinx.

Une fois le bloc des R échantillons constitué, le bloc fft_ctrl lit le bloc de 2N échantillons à travers le deuxième port de la ram_8 et tout en activant le bloc xfft1024 (qui est configuré ici pour effectuer des FFTs sur 256 points comme sus cité). Il est à noter que les différents blocs cités ici ont des horloges différentes, ce qui permet au bloc acq_ctrl de continuer à enregistrer les nouveaux échantillons pendant que le bloc fft_ctrl effectue une FFT sur un bloc de taille 2N.

La figure ci-dessous montra les interconnexions entre les différents blocs formant le bloc "Input Block" :

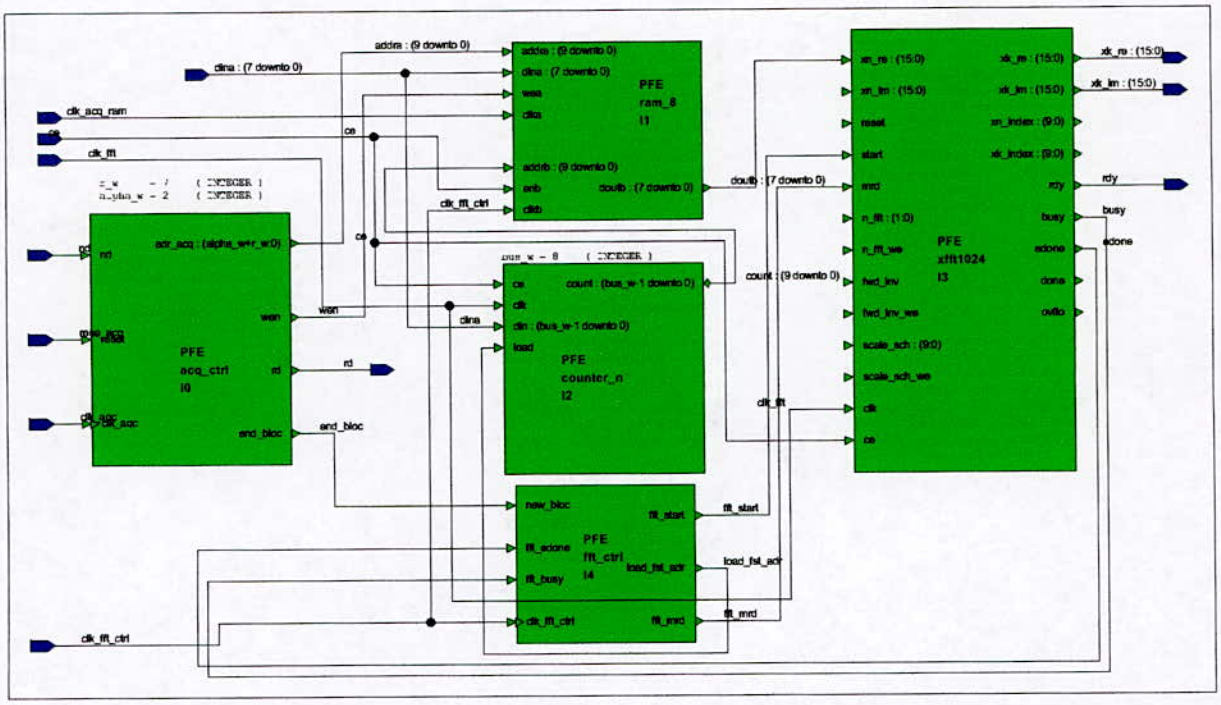


Figure 23-Bloc diagramme du bloc "Input Block"

III.B.1.Bloc acq_ctrl:

III.B.1.a)Implémentation :

Ce bloc est modélisé par la machine d'états finis suivante :

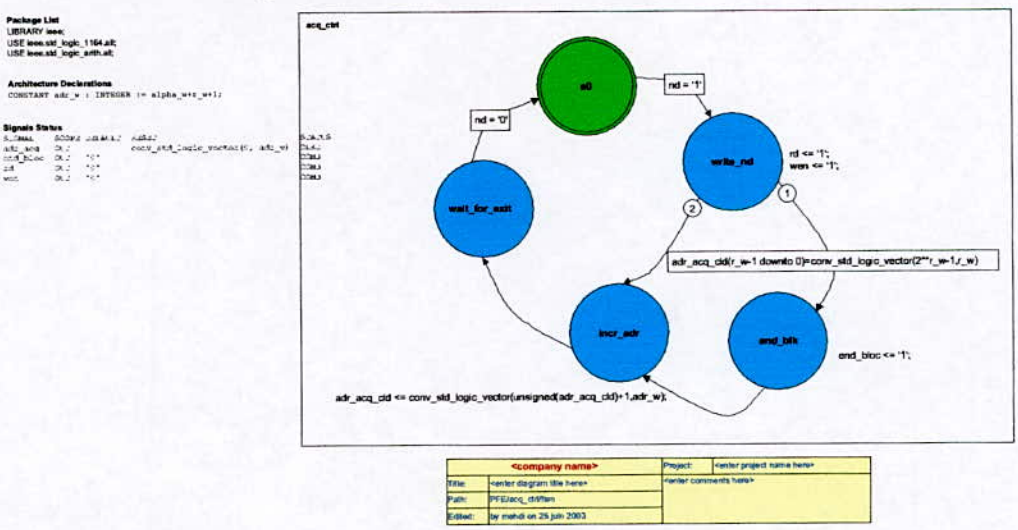


Figure 24-Machine d'état modélisant le bloc acq_ctrl

Le code VHDL généré par HDL designer series sera le suivant :

```
-- hds header_start
--
-- VHDL Entity PFE.acq_ctrl.symbol
--
-- Created:
--          by - mehdi.UNKNOWN (ETUDES)
--          at - 06:49:48 25/06/2003
--
-- Generated by Mentor Graphics' HDL Designer(TM) 2001.1 (Build 12)
--
-- hds header_end
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;


ENTITY acq_ctrl IS
  GENERIC(
    r_w    : INTEGER := 4;
    alpha_w : INTEGER := 2
  );
  PORT(
    clk_aqc  : IN    std_logic;
    nd       : IN    std_logic;
    reset    : IN    std_logic;
    adr_acq  : OUT   std_logic_vector (alpha_w+r_w DOWNTO 0);
    end_bloc : OUT   std_logic;
    rd       : OUT   std_logic;
    wen      : OUT   std_logic
  );

-- Declarations

END acq_ctrl ;

-- hds interface_end
--
-- VHDL Architecture PFE.acq_ctrl.fsm
--
-- Created:
--          by - mehdi.UNKNOWN (ETUDES)
--          at - 06:49:48 25/06/2003
--
-- Generated by Mentor Graphics' HDL Designer(TM) 2001.1 (Build 12)
--
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;


ARCHITECTURE fsm OF acq_ctrl IS

  -- Architecture Declarations
  CONSTANT adr_w : INTEGER := alpha_w+r_w+1;

  TYPE STATE_TYPE IS (
    s0,
    write_nd,
    end_blk,
    incr_adr,
```

```
  wait_for_exit
);

-- State vector declaration
ATTRIBUTE state_vector : string;
ATTRIBUTE state_vector OF fsm : ARCHITECTURE IS "current_state" ;


-- Declare current and next state signals
SIGNAL current_state : STATE_TYPE ;
SIGNAL next_state : STATE_TYPE ;
-- pragma synthesis_off
SIGNAL hds_next,hds_current,hds_clock: INTEGER;
-- pragma synthesis_on

-- Declare any pre-registered internal signals
SIGNAL adr_acq_cld : std_logic_vector (alpha_w+r_w DOWNTO 0) ;

BEGIN

-----------------------------------------------------------
clocked : PROCESS(
  clk_aqc
)
-----------------------------------------------------------
BEGIN
  IF (clk_aqc'EVENT AND clk_aqc = '1') THEN
    IF (reset = '1') THEN
      current_state <= s0;
      -- Reset Values
      adr_acq_cld <= conv_std_logic_vector(0, adr_w);
      -- pragma synthesis_off
      hds_current <= 0;
      -- pragma synthesis_on
    ELSE
      current_state <= next_state;
      -- pragma synthesis_off
      hds_current <= hds_next;
      hds_clock <= -1;
      hds_clock <= 0;
      -- pragma synthesis_on
      -- Default Assignment To Internals

      -- State Actions for internal signals only
      CASE current_state IS
      WHEN incr_adr =>
        adr_acq_cld <= conv_std_logic_vector(unsigned(adr_acq_cld)+1,adr_w);
      WHEN OTHERS =>
        NULL;
      END CASE;

    END IF;
  END IF;

END PROCESS clocked;

-----------------------------------------------------------
nextstate : PROCESS (
  adr_acq_cld,
  current_state,
```

```
  nd
)
-------------------------------------------------------------
BEGIN
  CASE current_state IS
  WHEN s0 =>
    IF (nd = '1') THEN
      next_state <= write_nd;
      -- pragma synthesis_off
      hds_next <= 1;
      -- pragma synthesis_on
    ELSE
      next_state <= s0;
      -- pragma synthesis_off
      hds_next <= 0;
      -- pragma synthesis_on
    END IF;
  WHEN write_nd =>
    IF (adr_acq_cld(r_w-1 downto 0)=conv_std_logic_vector(2**r_w-1,r_w)) THEN
      next_state <= end_blk;
      -- pragma synthesis_off
      hds_next <= 2;
      -- pragma synthesis_on
    ELSE
      next_state <= incr_adr;
      -- pragma synthesis_off
      hds_next <= 3;
      -- pragma synthesis_on
    END IF;
  WHEN end_blk =>
      next_state <= incr_adr;
      -- pragma synthesis_off
      hds_next <= 4;
      -- pragma synthesis_on
  WHEN incr_adr =>
      next_state <= wait_for_exit;
      -- pragma synthesis_off
      hds_next <= 5;
      -- pragma synthesis_on
  WHEN wait_for_exit =>
    IF (nd = '0') THEN
      next_state <= s0;
      -- pragma synthesis_off
      hds_next <= 6;
      -- pragma synthesis_on
    ELSE
      next_state <= wait_for_exit;
      -- pragma synthesis_off
      hds_next <= 0;
      -- pragma synthesis_on
    END IF;
  WHEN OTHERS =>
    next_state <= s0;
    -- pragma synthesis_off
    hds_next <= 0;
    -- pragma synthesis_on
  END CASE;

END PROCESS nextstate;
```

```
-------------------------------------------------------------
output : PROCESS (
  current_state
)
-------------------------------------------------------------
BEGIN
  -- Default Assignment
  end_bloc <= '0';
  rd <= '0';
  wen <= '0';
  -- Default Assignment To Internals

  -- State Actions
  CASE current_state IS
  WHEN write_nd =>
    rd <= '1';
    wen <= '1';
  WHEN end_blk =>
    end_bloc <= '1';
  WHEN OTHERS =>
    NULL;
  END CASE;

 END PROCESS output;

 -- Concurrent Statements
 -- Clocked output assignments
 adr_acq <= adr_acq_cld;


END fsm;
```

III.B.1.b) Test bench:

Le test bench utilisé pour la vérification est modélisé par le diagramme suivant :

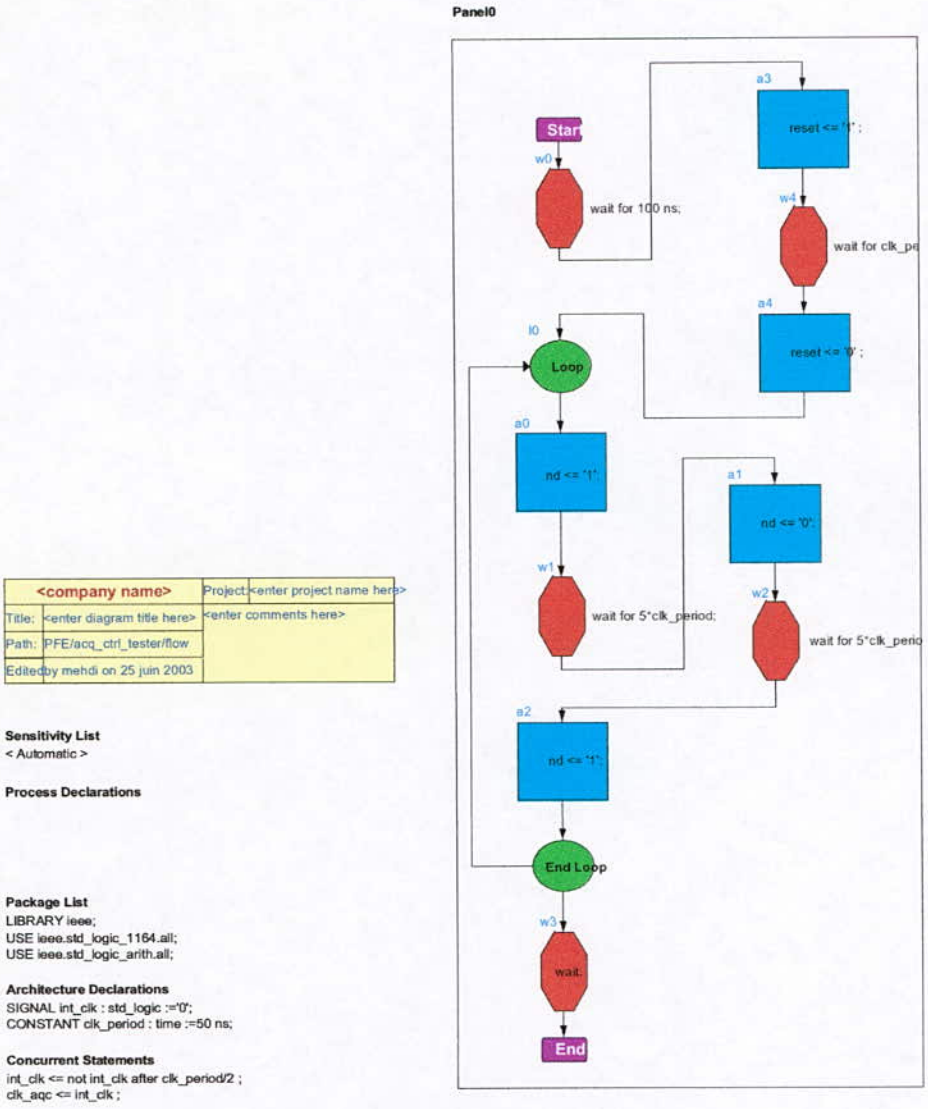


Figure 25 Test bench utilisé pour valider le bloc acq_ctrl

III.B.1.c)Simulation:

Les résultats de la simulation comportementale sont illustrés par les chronogrammes suivants :

Figure 26-Résultats de la simulation du bloc acq_ctrl

Nous pouvons vérifier à travers les chronogrammes que le bloc acq_ctrl fonctionne correctement car dès que les R échantillons sont enregistrés dans la ram_8, le signal end_bloc passe à 1.

III.B.2.Bloc counter_n:

III.B.2.a)spécification :

Le bloc counter_n est un compteur de n bits chargeable.

III.B.2.b)Implémentation :

Le bloc est décrit par le code VHDL suivant :

```
-- hds header_start
--
-- VHDL Architecture PFE.counter_n.symbol
--
-- Created:
--          by - mehdi.UNKNOWN (ETUDES)
--          at - 17:07:17 04/06/2003
--
-- Generated by Mentor Graphics' HDL Designer(TM) 2001.1 (Build 12)
--
-- hds header_end
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;


ENTITY counter_n IS
  GENERIC(
    bus_w : INTEGER := 8
  );
  PORT(
    clk   : IN    std_logic;
    ce    : IN    std_logic;
    load  : IN    std_logic;
    din   : IN    std_logic_vector (bus_w-1 DOWNTO 0);
    count : INOUT  std_logic_vector (bus_w-1 DOWNTO 0)
  );

-- Declarations

END counter_n ;

-- hds interface_end
ARCHITECTURE counter_vhd OF counter_n IS

BEGIN
process (clk)
BEGIN
```

```
if clk='1' and clk'event then
   if ce='1' then
            if load='1' then
                     count<= din;
            else
                     count<=count+1;
            end if;
   end if;
end if;
end process;

END counter_vhd;
```

III.B.3.Bloc fft_ctrl :

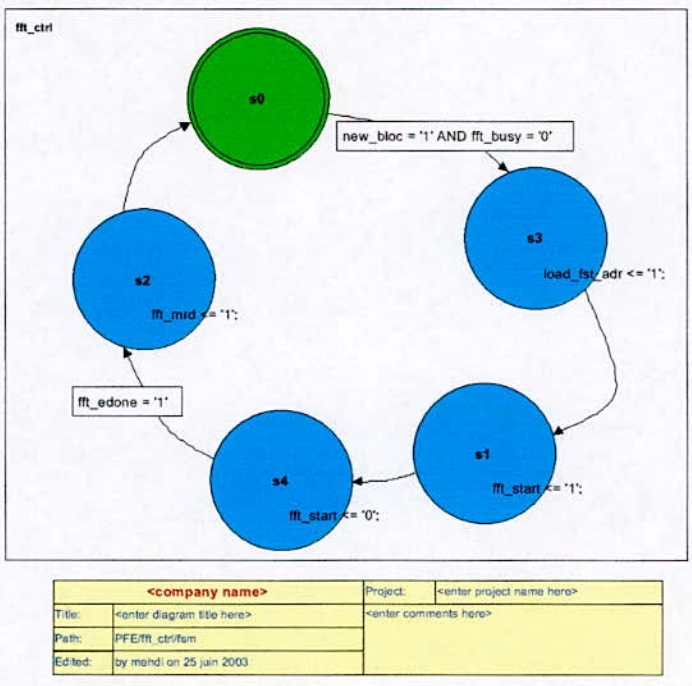
Ce bloc est utilisé en association avec le bloc counter_n pour controler le fonctionnement de la FFT du bloc "Input Block".

III.B.3.a)Spécification :

Dès que le bloc fft_ctrl reçoi reçoit un '1' sur le port new_bloc, il doit commencer à lire la mémoire ram_8 tout en démarrant le processeur FFT. Le compteur counter_n servant à incrémenter l'adresse de la mémoire ram_8.

III.B.3.b)Implémentation :

Le bloc fft_ctrl est modélisé par l'automate d'états finis suivant :



Package List
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

Signals Status

SIGNAL	SCOPE	DEFAULT	RESET	STATUS
fft_mrd	OUT	'0'		COMB
fft_start	OUT	'0'		COMB
load_fst_adr	OUT	'0'		COMB

Figure 27-Machine d'états finis décrivant le bloc fft_ctrl

Le code VHDL généré sera :

```
-- hds header_start
--
-- VHDL Entity PFE.fft_ctrl.symbol
--
-- Created:
--        by - mehdi.UNKNOWN (ETUDES)
--        at - 06:49:47 25/06/2003
--
-- Generated by Mentor Graphics' HDL Designer(TM) 2001.1 (Build 12)
--
-- hds header_end
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;


ENTITY fft_ctrl IS
  PORT(
    clk_fft_ctrl : IN     std_logic;
    fft_busy     : IN     std_logic;
    fft_edone    : IN     std_logic;
    new_bloc     : IN     std_logic;
    fft_mrd      : OUT    std_logic;
    fft_start    : OUT    std_logic;
    load_fst_adr : OUT    std_logic
  );

-- Declarations

END fft_ctrl ;

-- hds interface_end
--
-- VHDL Architecture PFE.fft_ctrl.fsm
--
-- Created:
--        by - mehdi.UNKNOWN (ETUDES)
--        at - 06:49:48 25/06/2003
--
-- Generated by Mentor Graphics' HDL Designer(TM) 2001.1 (Build 12)
--
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ARCHITECTURE fsm OF fft_ctrl IS
  -- Architecture Declarations
  TYPE STATE_TYPE IS (
    s0,
    s1,
    s2,
    s3,
    s4
  );
  -- State vector declaration
  ATTRIBUTE state_vector : string;
  ATTRIBUTE state_vector OF fsm : ARCHITECTURE IS "current_state" ;
```

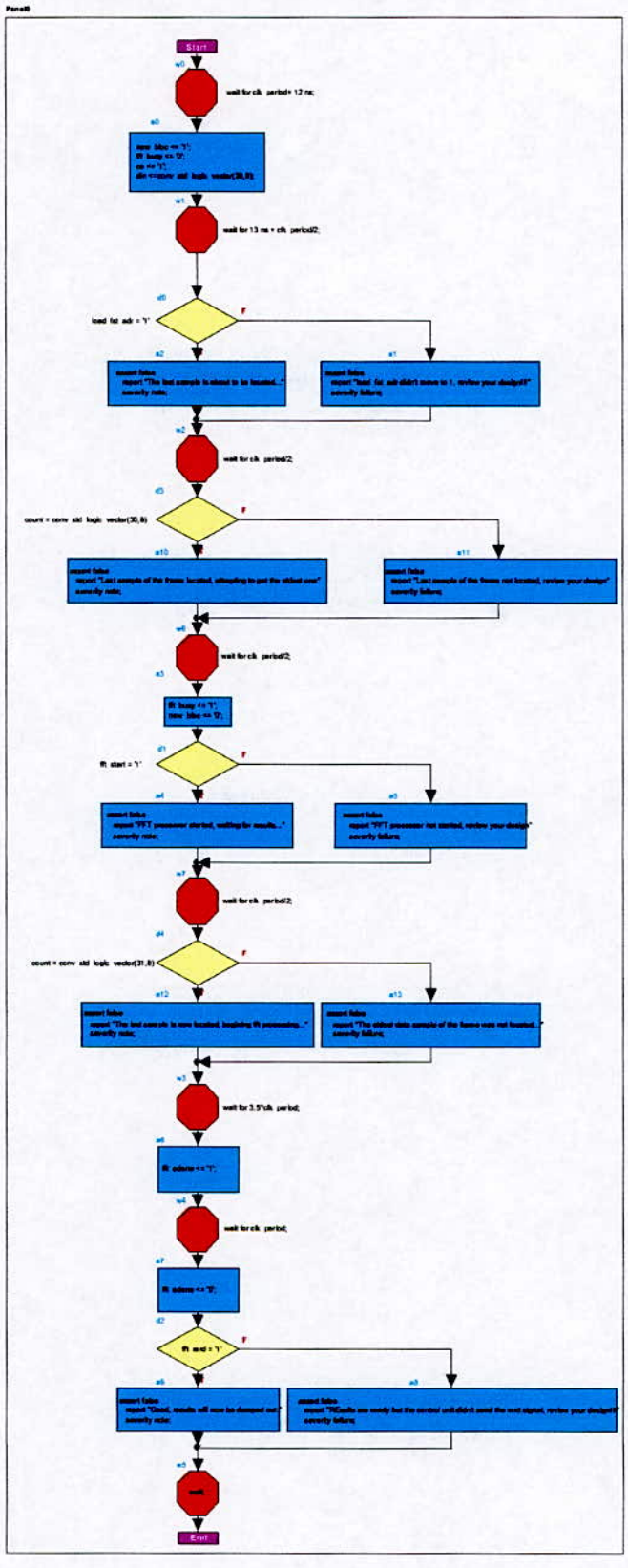
```
  -- Declare current and next state signals
  SIGNAL current_state : STATE_TYPE ;
  SIGNAL next_state : STATE_TYPE ;
  -- pragma synthesis_off
  SIGNAL hds_next,hds_current,hds_clock: INTEGER;
  -- pragma synthesis_on
BEGIN
  ----------------------------------------------------------------
  clocked : PROCESS(
    clk_fft_ctrl
  )
  ----------------------------------------------------------------
  BEGIN
    IF (clk_fft_ctrl'EVENT AND clk_fft_ctrl = '1') THEN
      current_state <= next_state;
      -- pragma synthesis_off
      hds_current <= hds_next;
      hds_clock <= -1;
      hds_clock <= 0;
      -- pragma synthesis_on
      -- Default Assignment To Internals
    END IF;
  END PROCESS clocked;
  ----------------------------------------------------------------
  nextstate : PROCESS (
    current_state,
    fft_busy,
    fft_edone,
    new_bloc
  )
  ----------------------------------------------------------------
  BEGIN
    CASE current_state IS
    WHEN s0 =>
      IF (new_bloc = '1' AND fft_busy = '0') THEN
        next_state <= s3;
        -- pragma synthesis_off
        hds_next <= 1;
        -- pragma synthesis_on
      ELSE
        next_state <= s0;
        -- pragma synthesis_off
        hds_next <= 0;
        -- pragma synthesis_on
      END IF;
    WHEN s1 =>
        next_state <= s4;
        -- pragma synthesis_off
        hds_next <= 2;
        -- pragma synthesis_on
    WHEN s2 =>
        next_state <= s0;
        -- pragma synthesis_off
        hds_next <= 3;
        -- pragma synthesis_on
    WHEN s3 =>
        next_state <= s1;
        -- pragma synthesis_off
        hds_next <= 4;
        -- pragma synthesis_on
```

```
    WHEN s4 =>
      IF (fft_edone = '1') THEN
        next_state <= s2;
        -- pragma synthesis_off
        hds_next <= 5;
        -- pragma synthesis_on
      ELSE
        next_state <= s4;
        -- pragma synthesis_off
        hds_next <= 0;
        -- pragma synthesis_on
      END IF;
    WHEN OTHERS =>
      next_state <= s0;
      -- pragma synthesis_off
      hds_next <= 0;
      -- pragma synthesis_on
    END CASE;
  END PROCESS nextstate;
  ----------------------------------------------------------------
  output : PROCESS (
    current_state
  )
  ----------------------------------------------------------------
  BEGIN
    -- Default Assignment
    fft_mrd <= '0';
    fft_start <= '0';
    load_fst_adr <= '0';
    -- Default Assignment To Internals
    -- State Actions
    CASE current_state IS
    WHEN s1 =>
      fft_start <= '1';
    WHEN s2 =>
      fft_mrd <= '1';
    WHEN s3 =>
      load_fst_adr <= '1';
    WHEN s4 =>
      fft_start <= '0';
    WHEN OTHERS =>
      NULL;
    END CASE;
  END PROCESS output;
  -- Concurrent Statements

END fsm;
```

III.B.3.c)Test bench:

Le bloc testeur du test bench du bloc fft_ctrl et counter_n est modélisé par le diagramme suivant :



<company name>		Project:	<enter project name here>
Title:	<enter diagram title here>	<enter comments here>	
Path:	FFT/R control_tester/flow		
Edited	by mehdi on 25 juin 2003		

Le test bench quand à lui est modélisé par le schéma suivant:

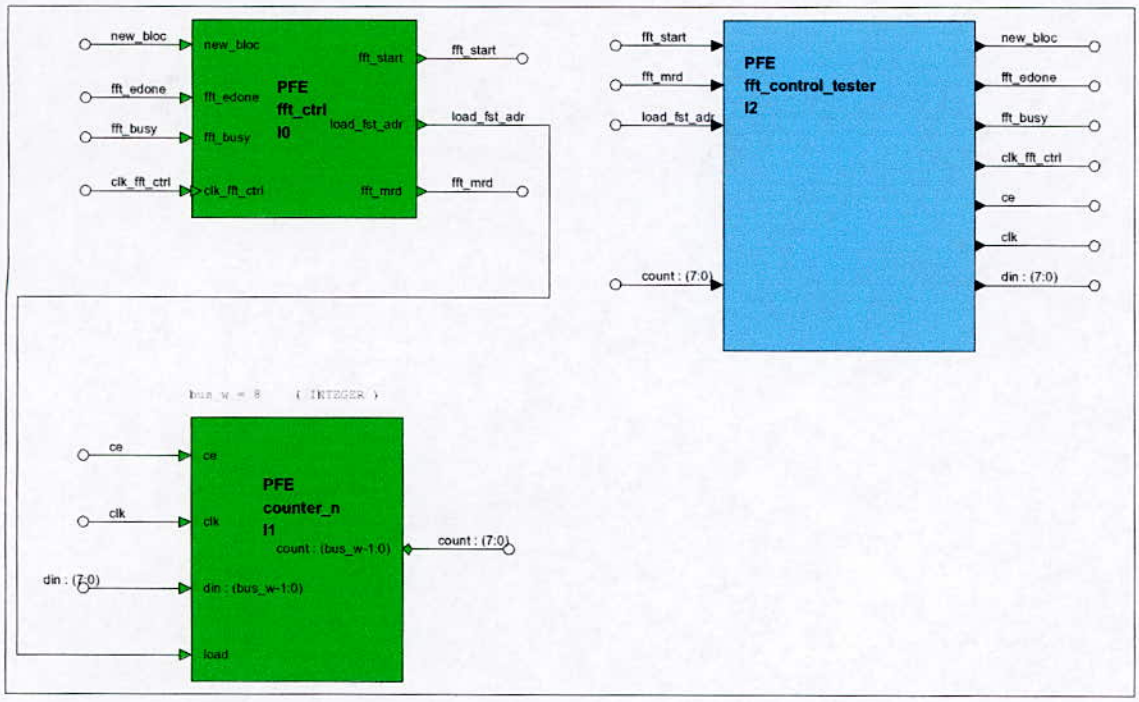


Figure 28-Test bench pour les blocs fft_ctrl et counter_n

La principale différence de ce dernier avec le test bench du bloc acq_ctrl est que la validation est automatique. En effet, en cas d'erreur, le simulateur affichera le message d'erreur spécifié par l'instruction "assert" et arrêtera l'exécution de la simulation.

III.B.3.d)Simulation :

En utilisant le test bench précédent, on n'a plus besoin d'analyser les chronogrammes, il suffit juste d'observer le transcript du simulateur (le cadre où sont affichés les messages et où sont tapées les commandes du simulateur).

Nous observons les messages suivants:

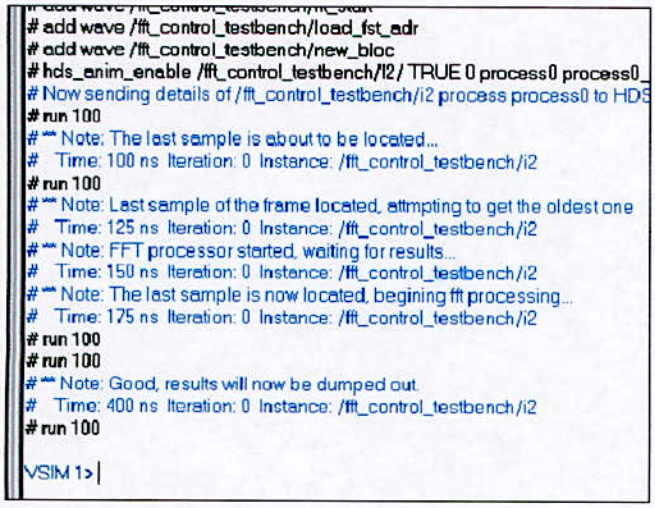


Figure 29-Transcript de ModelSim montrant les messages confirmant la validité de l'implémentation des blocs fft_ctrl et counter_n.

Les messages précédés du mot note nous informent que les blocs fft_ctrl et counter_n fonctionnent correctement.

On peut en plus effectuer une analyse de code coverage un une analyse avec le "performance analyzer". Mais celles-ci ne présentent aucun intérêt avec les designs de petite taille.

Conclusion

Bien que l'objectif initial de ce PFE était d'implémenter l'algorithme GMDFα sur circuit FPGA. Nous nous somme vite aperçus que pour atteindre cet objectif, il fallait tout d'abord acquérir :

- L'état de l'art des techniques d'implémentation VLSI.
- Une méthodologie de conception rigoureuse permettant de passer rapidement de l'idée vers la réalisation.
- Des outils de conception permettant de mettre en œuvre un projet VLSI quelque soit sa complexité ou sa taille.

Nous avons donc décidé de concentrer nos efforts sur ces trois points.

Pour ce qui concerne les techniques d'implémentation VLSI, nous avons remarqués que pour pouvoir les appliquer sur un algorithme de la taille du GMDFα, il faudrait disposer d'un logiciel spécifiquement conçu à cet effet. Nous avons donc pris la décision de développer un tool box sous Matlab pour arriver à cette fin. Ce toolbox ne contient pour l'instant que des fonctionnalités de base et est amené à être développé par la suite.

Pour ce qui concerne la méthodologie de conception Top-Down, nous considérons que c'est un de atouts majeurs que nous avons acquis à travers ce PFE. En effet, nous avons pris conscience que l'adoption d'une méthodologie claire nette et précise permet une réduction considérable du temps de développement et donc de pouvoir être plus compétitif.

Quand aux outils de développement, nous avons eu la chance de pouvoir utiliser les logiciels les plus perfectionnés dans le domaine. Ces logiciels apportent un gain non négligeable non seulement en convivialité en en simplicité d'emploi mais aussi en puissance et en pertinence des analyses qu'ils sont capables d'effectuer.

Nous sommes arrivés à la fin de ce PFE à la conclusion que même si nous n'avons pas aboutis à l'implémentation finale de l'algorithme GMDFα, nous nous sommes toutefois armés d'une batterie de techniques et d'outils nous permettant dorénavant d'aborder l'implémentation VLSI de manière plus efficace.

Annexes

I.RESOLUTION D'UN SYSTEME D'INEGALITES:

Soit un système de M inégalités à N inconnues où chaque inégalité est de la forme $r_i - r_j \leq k$ avec k entier.

L'algorithme suivant peut être utilisé pour résoudre ce système:

1. Dessiner un graphe de contraintes :
   a. Dessiner le nœud i pour chacune des N variables.
   b. Dessiner le nœud N+1.
   c. Pour chaque inégalité $r_i - r_j \leq k$, dessiner la flèche $j \rightarrow i$ ayant le poids k.
   d. Pour chaque nœud i=1,…,N, dessiner la flèche $N+1 \rightarrow i$ ayant le poids 0.
2. résoudre en utilisant un algorithme de résolution du problème du chemin le plus court :
   a. le système d'inégalités n'a de solutions que si le graphe de contraintes ne contient pas de cycles ayant un poids négatif.
   b. Si une solution existe, une des solutions est celle où r_i est le chemin le plus court du nœud N+1 vers le nœud i.

Nous utiliserons pour l'étape 2 l'algorithme de Bellman-Ford qui est décrit dans l'annexe II.A.

I.A.Programme Matlab :

```
function Sol=solve_ineq(system)
% Function for solving system of inequalities.
%Inputs the system in a 3 columns x M lines matrix.
%  The first column contains the i index of each inequality.
%  The 2nd column contains the j index of each inequality.
%  The 3rd column contains the k parameter of each inequality.
%If a solustion exists than it is copied out in the Sol variable
%and if not than the variable Sol will be equa to inf.
N1=max(system(1,:));
N2=max(system(2,:));
N=max([N1,N2]);
M=length(system(1,:));
w=inf*ones(N+1,N+1);
```

```
for i=1:M;
   w(system(2,i),system(1,i))=system(3,i);
end
for i=1:N
   w(N+1,i)=0;
end
[neg,Sol]=belford(w,N+1);
if neg==1
   Sol=inf;
End
```

II.ALGORITHMES POUR LA RESOLUTION DU PROBLEME DU CHEMIN LE PLUS COURT (SHORTEST PATH PROBLEM) :

II.A.Algorithme de Bellman-Ford :

Cet algorithme résout le problème de trouver le chemin le plus court du nœud U (que l'on nommera origine) vers tout autre nœud du graphe.

Cet algorithme a pour principe la construction de n-1 vecteurs $r^{(k)}$ n étant le nombre de nœuds et est décrit par:

1. $r^{(1)}(U) = 0$
2. *For* $k = 1$ *to* n
3. *If* $k \neq U$
4. $r^{(1)}(k) = w(U \xrightarrow{e} k)$
5. *For* $k = 1$ *to* $n-2$
6. *For* $V = 1$ *to* n
7. $r^{(k+1)}(V) = r^{(k)}(V)$
8. *For* $W = 1$ *to* n
9. *If* $r^{(k+1)}(V) > r^{(k)}(W) + w(W \xrightarrow{e} V)$
10. $r^{(k+1)}(V) = r^{(k)}(W) + w(W \xrightarrow{e} V)$
11. *For* $V = 1$ *to* n
12. *For* $W = 1$ *to* n
13. *If* $r^{(n-1)}(V) > r^{(n-1)}(W) + w(W \xrightarrow{e} V)$
14. *return FALSE and exit*
15. *return TRUE and exit*

Si une solution est trouvée TRUE est retourné et la solution sera donnée par le vecteur $r^{(n-1)}$.

II.A.1.Programme Matlab:

Le programme matlab est le suivant:

```
function [neg_path,short_pths]=belford(w,U)

%Implementation of the BELLMAN-FORD algorithm for computing the shortest path.
%Inputs: weight vector, start point.
```

```
%Outputs: Are there negative paths?, if no retur the shortest paths from the
%   point to every point of the graph.

n=size(w,1);
r=zeros(n-1,n);

r(1,U)=0;
for k=1:n
   if k~=U
      r(1,k)=w(U,k);
   end
end
for k=1:n-2
   for V=1:n
      r(k+1,V)=r(k,V);
      for W=1:n
         if r(k+1,V)>r(k,W)+w(W,V)
            r(k+1,V)=r(k,W)+w(W,V);
         end
      end
   end
end
for V=1:n
   for W=1:n
      if r(n-1,V)>r(n-1,W)+w(W,V)
         neg_path=1;
         return;
      end
   end
end
neg_path=0;
short_pths=r(n-1,:)';
```

II.B.L'algorithme de Foyd-Warshall :

Cet algorithme est similaire au précédent sauf qu'il détermine la solution pour tout nœud U vers tout nœud V.

L'algorithme en question est basé sur la construction de n+1 matrices $R^{(k)}$, si une solution est trouvée, la valeur TRUE est retournée et la solution sera la matrivce $R^{(n+1)}$.

L'algorithme de Foyd-Warshall est décrit par :

1. *For* $V = 1$ *to* n
2. *For* $U = 1$ *to* n
3. $r^{(1)}(U,V) = w(U \overset{e}{\rightarrow} V)$
4. *For* $k = 1$ *to* n
5. *For* $V = 1$ *to* n
6. *For* $U = 1$ *to* n
7. $r^{(k+1)}(U,V) = r^{(k)}(U,V)$
8. *If* $r^{(k+1)}(U,V) > r^{(k)}(U,k) + r^{(k)}(k,V)$
9. $r^{(k+1)}(U,V) = r^{(k)}(U,k) + r^{(k)}(k,V)$
10. *For* $k = 1$ *to* n
11. *For* $U = 1$ *to* n
12. *If* $r^{(k)}(U,U) < 0$
13. *return FALSE and exit*
14. *return TRUE and exit*

II.B.1.Pogramme Matlab :

```
function [neg_path,shortest_p]=flowar(w)

%Implementation of the Floyd-Warshall Algorithm.
%Inputs: the weigth matrix.
%Outputs: If there is no negative path, the shortest path matrix.

n=size(w,1);
r=zeros(n+1,n,n);
for V=1:n
  for U=1:n
    r(1,U,V)=w(U,V);
```

```
    end
  end
  for k=1:n
    for V=1:n
      for U=1:n
        r(k+1,U,V)=r(k,U,V);
        if r(k+1,U,V)>r(k,U,k)+r(k,k,V)
          r(k+1,U,V)=r(k,U,k)+r(k,k,V);
        end
      end
    end
  end
  for k=1:n
    for U=1:n
      if r(k,U,U)<0
        neg_path=1;
        return;
      end
    end
  end
  neg_path=0;
  shortest_p(:,:)=r(n+1,:,:);
```

III. PROGRAMME DE L'ALGORITHME LPM:

```
function [L,T_inf]=LPM(L1)
d=size(L1,1);
L=zeros(d,d,d);
L(1,:,:)=L1(:,:);
for m=2:d
   for i=1:d
      for j=1:d
         K=[-1];
         for k=1:d
            if (L(1,i,k)~=-1 & L(m-1,k,j)~=-1)
               K=[K L(1,i,k)+L(m-1,k,j)];
            end
         end
         L(m,i,j)=max(K);
      end
   end
end

for m=1:d
   for i=1:d
      T(m,i)=L(m,i,i)/m;
   end
end
T_inf=max(T);
T_inf=max(T_inf);
```

IV. IMPLEMENTATION MATLAB DE L'ALGORITHME DE CALCUL DE W ET DE D:

```
function [W,D]=calc_WD(T,w)
N=length(T)
M=max(T)*N;
w_p=zeros(N,N);
for i=1:N
   for j=1:N
      w_p(i,j)=M*w(i,j)-T(i);
   end
end
[neg,sol]=flowar(w_p);
W=zeros(N,N);
D=W;
for i=1:N
   for j=1:N
      if i~=j
         W(i,j)=ceil(sol(i,j)/M);
         D(i,j)=M*W(i,j)-sol(i,j)+T(j);
      else
         W(i,j)=0;
         D(i,j)=T(i);
      end
   end
end
```

Références bibliographiques

- [1] VLSI digital signal processing systems. Design and implementation. Keshab K.PARHI. ed. JOHN WILEY & SONS, INC. 1999.

- [2] Mise en oeuvre d'un annuleur d'écho acoustique. F.ABDA et S.MOUSSAOUI. Ecole Nationale Polytechnique. 2001.

- [3] Etude d'un annuleur d'écho acoustique. ABDULHAMID SAID et BELGACEM MONCEF. ENP. 1999.

- [4] FPGA Advantage 5.0 bookcase. Mentor Graphics. 2001.

- [5] Xilinx 5 software manuals.

- [6] VLSI architecture of the Generalized Multidelay Frequency-Domain algorithm for acoustic echo cancellation. Amer El Hawani et Patrice Le Scan. France Telecom – CNET. 1995.

- [7] A co-design methodology for telecommunication systems : a case study of an acoustic echo canceller. Adel Baganne, Jean Luc Philippe et Eric Martin. UBS University. 1997.

- [8] A multi-granularity data synchronized architecture for HW/SW embedded DSP systems. Michel AUGUIN, Cécile BELLEUDY, Guy GOGNIAT et Yvon JEGOU.