REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE YAHIA FARES DE MEDEA

Faculté de **T**echnologie

Département du **G**énie **E**lectrique

Mémoire de fin d'études de Master

Filière : electronique

Spécialité : électronique des système embarqués

Etude et implémentation sur FPGA d'un filtre a paramètre variable : Application sur les roulements a Billes

Présentés par :

AISSAOUI Fethi

ZERAIGUI Zohir

Proposés et Dirigés par :

LACHENANI Sid Ahmed

Promotion 2022

Remerciement

*Nous tenons à remercier « **Allah** » le tout puissant de nous avoir donné le courage, la force et la volonté pour réaliser ce mémoire.*

La présentation de ce travail nous avons offre d'exprimer toute mes reconnaissances aux personnes qui nous aidées et conseillés de façonsinestimables durant la préparation de ce mémoire.

Nous tenons à exprimer notre plus haute estime à Mr ***LACHENANI Sidahmed*** *notre encadrant pour s'assistance, leur encouragement et orientation qu'il a donné.*

Nous remercons encore toutes les personnes qui ont contribué de prèsou de loin la réalisation de ce mémoire.

Nous souhaitons que ce mémoire soit utile à toutes les futures.

ملخص :

درسنا في هذه الأطروحة نظامًا لتصفية الإشارة الخام المقاسة باستخدام مرشح RIF ، يجب تحديث معاملات المرشح وفقًا لكل حالة من أجل الكشف بشكل أفضل عن عيوب المحامل الكروية.

تم إنجاز العمل في برنامج Vivado HLS لتحويل مواصفات النظام في C إلى تنفيذ(مستوى نقل سجل RTL)الذي تقوم بتوليفه في FPGA ، وتحقيق تصميم في Vivado HLS باستخدام واجهات AXI من خلال مكامل IP .

الكلمات المفتاحية : المحامل ، Vivado HLS، مرشح RIF ، المعاملات ، FPGA، RTL ، تكامل IP .

Résumé :

Dans ce mémoire nous avons étudié un système de filtrage du signal brut mesure a l'aide d'un filtre rif , les coefficients du filtre doivent être mises à jour selon chaque cas pour une meilleurs détection des défauts de roulements à billes.

Le travail a été fait dans logiciel Vivado HLS pour transforme une spécification système en C en une implémentation (RTL Register Transfer Level) que tu synthétises dans un FPGA, et réaliser une conception sous Vivado HLS avec des interfaces AXI travers IP intégrateurs .

Mots clés : Roulements ,Vivado HLS, Filtre RIF , coefficients, FPGA, RTL , IP intégrateur

Abstract :

In this thesis we studied a system of filtering the raw signal measured with a filter FIR, the coefficients of the filter must be updated according to each case for a better detection of defects of ball bearings.

The work was done in Vivado HLS software to transform a system specification in C into an implementation (RTL Register Transfer Level) that you synthesize in an FPGA, and realize a design under Vivado HLS with interfaces AXI through IP integrator .

Keywords : Bearings ,Vivado HLS, FIR filter ,coefficients, FPGA , RTL , IP integrator.

Table des matières

Liste des figures……...I
Liste des tableau……..……....III
Liste des symboles…….…….. IV
Liste des abreviation……..V
Introduction générale…….…..…… ……..….1

Chapitre 01 Filtre et filtrage numerique et les roulements

1.1 Introduction 2
1.2 Définition d'un filtre..... 2
1.3 Différents types du filtre..... 2
1.4 Filtrage numérique..... 3
1.4.1 Les spécifications d'un filtre numérique..... 4
1.4.2 Invariance dans le temps..... 4
1.4.3 Causalité 4
1.4.4 Les caractéristiques des filtres numériques..... 5
1.4.5 Réalisation d'un filtre numérique 5
1.5 Convolution..... 5
1.6 Corrélation..... 6
1.7 Filtre à réponse impulsionnelle finie (RIF) 6
1.7.1 Structure de filtre RIF..... 7
1.7.2 caractéristiques des filtres numériques RIF..... 7
1.7.3 Les avantages et les inconvénients des filtres RIF 8
1.7.4 Conception de filtre 8
1.7.5 Méthode de conception de fenêtre..... 9
1.7.6 Méthode d'échantillonnage fréquentiel..... 10
1.7.7 Méthode de Parks et McClellan..... 10
1.8 Constitution d'un roulement..... 11
1.9 Les différents types de roulement 11
1.9.1 Les roulements à billes 11
1.9.2 Roulement à une rangée de billes à contact oblique :..... 12
1.9.3 Roulement à billes à contact radial..... 12
1.9.4 Les roulements à rouleaux..... 12
1.9.5 Roulements radiaux et axiaux :..... 14
1.9.6 Les roulements à aiguilles : 14
1.9.7 Butées 15
1.9.8 Roulements standard et spéciaux..... 15
1.10 Origine des défauts de roulement..... 15
1.11 Quelques défauts de roulement 16
1.11.1 Le grippage 16
1.11.2 La corrosion..... 17
1.11.3 Usure..... 17
1.11.4 La fissuration des bagues..... 17
1.11.5 L'écaillage 17
1.12 Fréquences caractéristiques pour les défauts d'un roulement..... 18
1.13 Conclusion..... 19

Chapitre 02 l'outils De Developpement Hls Vivado

2.1 Introduction 20
2.2 Historique et évolution de la HLS 20
2.3 La génération classique et la génération en HLS 21
2.3.1 Flot de conception RTL 21

2.3.2 Flot de conception HLS 22
2.3.3 Flot de conception SoC-FPGA 24
2.4 La chaine opérationnelle sous vivado HLS 27
2.5 Les éléments de base sous HLS 28
2.5.1 Les interfaces 28
2.5.2 Les interfaces de synthèse 28
2.5.3 Les principaux protocoles d'interface sous HLS :..... 29
2.5.4 Les types de données en langage C ou C++ sur Vivado : 31
2.5.5 Les interfaces de block level : 32
2.6 Mise en place d'un décodeur sur Vivado HLS : 33
2.7 Flot de conception avec Vivado HLS de Xilinx et gestion des optimisations et interfaces 33
2.7.1 Gestion des interfaces 36
2.7.2 Gestion des optimisations :..... 37
2.8 Avantages et problèmes du HLS :..... 38
2.8.1 Avantages du HLS 38
2.8.2 Les problèmes du HLS 39
2.9 Défis et limitations des outils HLS : 40
2.10 Conclusion 42

Chapitre 03 Simulation et implémentation

3.1 Introduction 43
3.2 Implémentation sur HLS 43
3.2.1 Création du projet.....44
3.2.1 Exécutez la simulation 49
3.2.2 Synthèse en code VHDL de notre fichier source en C 50
3.2.3 Simulation de la fonction synthétisée 51
3.2.4 Exportation du fichier pour la création d'un IPBLOCK.....51
3.3 Créer un projet Vivado 53
3.3.1 Importation de notre IPBLOCK sous Vivado.....54
3.3 Implémentation sur SDK.....55
3.4 Résultat de la simulation 56
3.4.1 Simulation avec MATLAB et implimentation avec Vivado 56
3.4.2 Comparaison entre les résultat obtenus sous MATLAB et Vivado.....57
3.5 Conclusion 57

Conclusion générale……..…….....58
Références…….…… 59

LISTE DES FIGURES

Figure 1.1 Schéma d'un filtre……..……2
Figure 1.2 Réponses fréquentielles idéales des 4 filtres de base ……3
Figure 1.3 Système linéaire..……..…4
Figure 1.4 Système invariant dans le temps..……..……...……4
Figure 1.5 Structure directe d'un filtre RIF..……..……7
Figure 1.6 Structure transposée d'un filtre RIF...……..……..……7
Figure 1.7: composantes d'un roulement ..……...……11
Figure 1.8 : Roulement à une rangée de billes à contact oblique...……...……12
Figure 1.9 : Roulement à rouleaux cylindriques...……...……13
Figure 1.10 : Roulement à rouleaux coniques...……..……..……..……13
Figure 1.11 : roulements à aiguilles.……..……14
Figure 1.12 : Butées.....……15
Figure 1.13 : Grippage d'une bague.……..……16
Figure 1.14 : Corrosion sur une piste de roulement...……...……17
Figure 1.15 : (a) écaillage de fatigue et (b) écaillage superficiel de la bague interne ……..…18
Figure 1.16 : Caractéristiques d'un roulement……..……..……18
Figure 2.1: Flot de conception RTL……..……..……22
Figure 2.2: Flot de conception HLS..……..……..……23
Figure 2.3: Architecture SoC-FPGA ..……..……25
Figure 2.4: Flot de conception SoC ..……..……..……..…26
Figure 2.5 : chaine opérationnelle sous vivado HLS...……..……...…27
Figure 2.6 : comportement de l'interface AP_HS……..……29
Figure 2.7 : comportement de l'interface AP_MEMORY……..……..……29
Figure 2.8 : comportement de l'interface AP_FIFO……..……...……30
Figure 2.9 : protocole d'interface de niveau bloc ……32
Figure 2.10 : Le traitement des flux de données...……...……..……33
Figure 2.11: Flot de conception Vivado HLS ..……34
Figure 2.12: Phase d'ordonnancement ….……..…….35
Figure 2.13 : Phase d'assignation ……..……...…35
Figure 2.14 : Flot de conception des applications sur des plateformes hétérogènes Reconfigurables ..…….....……42
Figure 3.1 Projet configuration..…….…….....……44
Figure 3.2 Fichiers sources du projet …….…….....……44

Figure 3.3 Fichiers de banc d'essai de projet..…….....……45
Figure 3.4 soulution configuration …….....45
Figure 3.5 program fir.c ……..……...……46
Figure 3.6 program fir.h ……..…….....……47
Figure 3.7 program fir_test.c …….…….....……47
Figure 3.8 une interface de synthèse pour le port y …….....…….……48
Figure 3.9 Sélection de l'adaptateur AXI4LiteS et de l'ensemble de dénomination ……..….48
Figure 3.10 Partie initiale de la sortie générée dans la vue Console..……..…….....…49
Figure 3.11 Signaux d'interface générés..……....……50
Figure 3.12 Le code Verilog synthétis..…….....……50
Figure 3.13 RTL/C CoSimulation ..……51
Figure 3.14 Boîte de dialogue Exporter RTL ..……...……51
Figure 3.15 Adaptateur IP-XACT généré……52
Figure 3.16 interface ghraphique de vivado(xlinx IDE) représentant les différents composants embarqués sur la Zybo Z7. …….....……53
Figure 3.17 Conception de bloc avec zed_audio_ctrl et connexions faites pour Zybo Z7…..53
Figure 3.18 Définition du chemin d'accès aux référentiels IP..……..…….……54
Figure 3.19 fir référentiels IP…….…….....……54
Figure 3.20 La conception matérielle complète……..…….....……55
Figure 3.21 Test de l'IP-core « add » sur la carte FPGA …….....…..……55
Figure 3.22 le signal avant et après filtrage …….…….....……56
Figure 3.23 comparison signal apres filtrage avec MATLAB et FPGA …….…….....……57

Liste des tableaux

Tableau 2.1 : les restrictions sur le passage des paramètres pour chacun des protocoles ……30
Tableau 2.2 : les entiers en langage C ou C++ sur vivado ……31
Tableau 2.3 : les flottants à virgule fixe ……31
Tableau 2.4 : Directives d'optimisation Vivado HLS ……37
Tableau 2.5 : Configurations Vivado HLS ……38

Liste de Symboles

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

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

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

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

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

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

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

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

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

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

FR: La fréquence de rotation.

dm : le diamètre du cercle primitif.

n : le nombre de billes.

db : le diamètre de la bille.

α : l'angle de contact.

BPFI : Fréquence de défaut interne, HZ.

BPFO : les fréquences de Défaut externe, HZ.

B : les fréquences de Défaut élément roulant, HZ.

T : période.

Liste des abréviation

FPGA: Field Programmable Gte Array.

RIF : Réponse Impulsionnelle Finie.

VHDL: Very High Speed Integrated Circuit.

FFT :(Fast Fourier Tranform) Transforme de Fourier rapide.

BPF : Fréquence corps roulants.

BPFI : (Ball Pass Frequency Inner) Fréquence bague intérieure.

BPFO : (Ball Pass Frequency Outter) Fréquence bague extérieure.

FPGA : Field Programmable Gate Array.

MAC : Multiply and accumulate.

HLS : high level synthesis.

Introduction générale

Dans les domaines industriel les roulements sont largement utilisés, dans différente types des machines (électrique, mécaniques, véhicule électrique …), Pour un diagnostic des défauts des roulements et une maintenance préventive plus efficace nous trouvons plusieurs travaux de recherches utilisent les signaux vibratoires qu'ils contiennent des informations précieuses sur les défauts [1].

Notre travail est basé sur le filtrage du signal brut ou l'amélioration du rapport signal sur bruit ; qu'est une étape de prétraitement clé avant l'analyse du spectre d'enveloppe pour que la détection des défauts des roulements soit plus efficace. Cette technique nécessite généralement des théories de traitement de signal. Pour l'étude et l'implémentation sur FPGA d'un filtre a paramètre variable. L'FPGA est considéré comme le dispositif le plus efficace pour les applications en raison de sa performance, sa faible consommation d'énergie, et sa re-configurabilité.

Le but de notre travail est l'implémentation sur FPGA d'un filtre a paramètre variable pour l'amélioration du rapport signal sur bruit pour une meilleur détection de défaut des roulements à billes , pour cela on a organisé notre mémoire comme suivant:

- Le premier chapitre est consacré à la présentation des généralités sur les différents types des filtres et en détaille le filtre RIF, en même temps données une définition sur les roulements, et leurs différents types de défauts.

- Le deuxième chapitre contient une description et généralité sur l'outils HLS VIVADO utilise dans ce travail.

- Le troisième chapitre présente l'implémentation de notre solution, le block désigne de notre solution ; avec les résultats de l'implémentation.

Ce mémoire a pour objectif, d'appliquer le traitement du signal dans le domaine de maintenance industrielle, en effectuant un filtrage du signal brute avec des coefficients modifiables pour un meilleur diagnostic pour la détection et la caractérisation des défauts des roulements à billes sur circuit FPGA.

Chapitre 01

Filtre et filtrage numerique

et les roulements

1.1 Introduction

Ce chapitre présenter les filtres numériques et leurs différents types, caractéristiques et structurelles. Pour cela, nous étudierons les filtres à réponse impulsionnelle finie et les structures de leurs réalisations.

En mécanique, un roulement est un dispositif destine à guider assemblage en rotation, c'est-à-dire : permettre à une pièce à tourner par a rapport à une autre selon un axe de rotation défini ; Les roulements sont utilisés dans les machines rotatives et considérés comme un composant important, Toutes les machines en fonctionnement produisent des vibrations, et des efforts dynamiques engendrés par les pièces en mouvement. Ainsi, une machine neuve en excellent état de fonctionnement produit très peu de vibrations [2].

La majorité des défauts sont des creux ou des fissures apparaissant sur les composants du roulement et sont causés par l'usure ou une mal utilisation. Si ces défauts ne sont pas détectés au début, à travers le temps cela peut conduire à un échec du roulement et de la machine. Cela peut causer un grave problème de sécurité et pertes économiques. Dans les industries, la surveillance des vibrations est généralement faite pour vérifier la santé des roulements en termes de détection des défauts avant d'atteindre à l'étape critique, dans la plus part des cas, une opération de filtrage est nécessaire pour une meilleures détection des défauts . [2]

1.2 Définition d'un filtre

Un filtre est un circuit électronique qui réalise une opération de traitement du signal. Autrement dit, il atténue certaines composantes d'un signal et en laisse passer d'autres. Il constitue la base du traitement du signal dans les disciplines de l'ingénierie et s'applique à tous les types de signaux (sons, images, vidéo, vibrations sismiques, ...), la figure 1.1 représente un exemple d'un Schéma de filtre

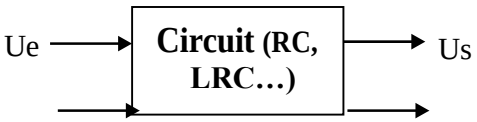


Figure 1.1 Schéma d'un filtre.

1.3 Différents types du filtre

Dans le texte suivant en représentes les différentes types de filtre.

- **Filtre passe bas**

Il ne laisse passer que les fréquences au-dessous de sa fréquence de coupure. C'est un atténuateur d'aiguës pour un signal audio. On pourrait l'appeler coupe-haut (voir figure 1.2).

- **Filtre passe haut**

Il ne laisse passer que les fréquences au-dessus d'une fréquence déterminée, appelée "fréquence de coupure". Autrement dit, il « laisse passer ce qui est haut ». C'est un atténuateur de graves pour un signal audio. On pourrait aussi l'appeler coupe-bas (voir figure 1.2).

- **Filtre passe bande**

Il ne laisse passer qu'une certaine bande de fréquences (et atténue tout ce qui est au-dessus ou en-dessous). Il est très utilisé dans les récepteurs radio, tv pour isoler le signal que l'on désire capter (voir figure 1.2), utilise dans notre cas.

- **Filtre coupe-bande**

Aussi appelé filtre trappe, cloche ou réjecteur de bande, est le complémentaire du passe-bande. Il atténue une plage de fréquences (voir figure 1.2).

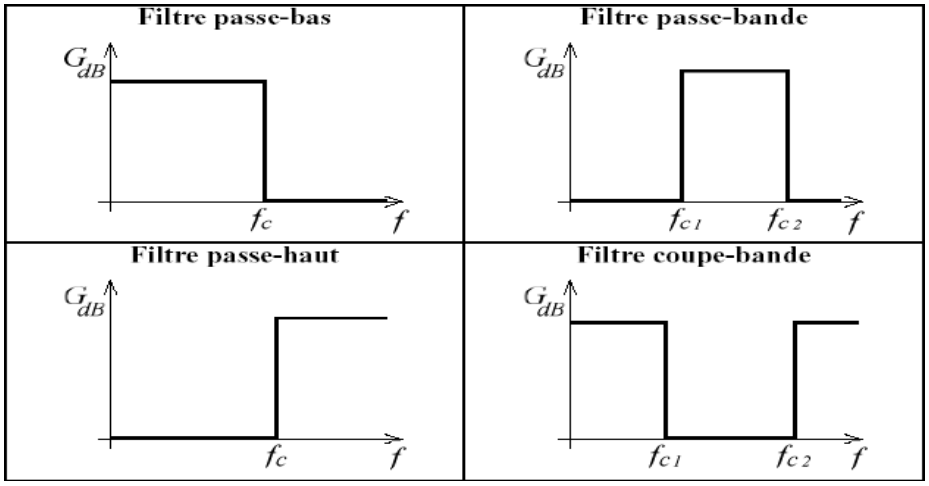


Figure 1. 2 Réponses fréquentielles idéales des 4 filtres de base.

1.4 Filtrage numérique

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

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

1.4.1 Les spécifications d'un filtre numérique

Les spécifications d'un filtre numérique sont donne comme suivants :

a Linéarité

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

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

$$y_1(n) = h(x_1(n)) \;\; et \;\; y_2(n) = h(x_2(n)) \qquad (1.1)$$

Le système h est linéaire si

$x_3(n) = ax_1(n) + bx_2(n)$ donc $y_3(n) = h(x_3(n))$ - $ah(x_1(n)) + bh(x_2(n)) = ay_1(n) + by_2(n)$ (1.2)

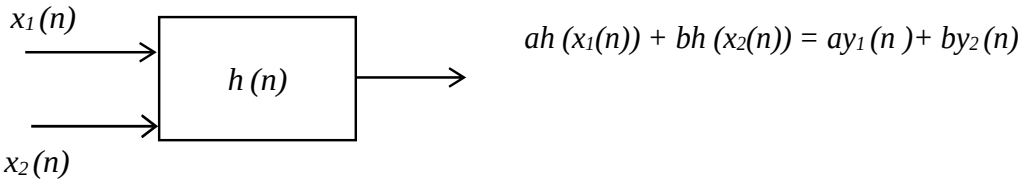


Figure 1. 3 Système linéaire.

1.4.2 Invariance dans le temps

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

$y_2(n) = y_1(n - n_0)$ (voir figure 1.4).



Figure 1. 4 Système invariant dans le temps.

1.4.3 Causalité

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

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

1.4.4 Les caractéristiques des filtres numériques

- **La réponse impulsionnelle**

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

- **Principe de stabilité**

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

- **L'équation aux différences**

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

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

- **La fonction de transfert**

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

$$h(z) = \frac{y(z)}{x(z)} = \frac{\sum_{K=0}^{N} b_K Z^{-k}}{1+\sum_{K=1}^{M} a_K Z^{-K}} \qquad (1.4)$$

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

1.4.5 Réalisation d'un filtre numérique

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

1.5 Convolution

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

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

1.6 Corrélation

La corrélation est un ensemble d'applications fondamentales de détection, extraction, détermination de caractéristiques dynamiques d'un système linéaire. Les fonctions de corrélation traduisent la similitude d'un signal ou de deux signaux au niveau de la formeet de la position en fonction du paramètre de translation t [3].

- Autocorrélation

Autocorrélation est une étude de la ressemblance du processus avec lui-même au cours du temps, et par conséquent, si le signal est périodique, la fonction d'autocorrélation permettra de détecter cette périodicité [3]. La fonction d'autocorrélation est définie par :

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

- **Intercorrélation**

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

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

1.7 Filtre à réponse impulsionnelle finie (RIF)

Le filtre à réponse impulsionnelle finie ou filtre RIF est un filtre numérique qui est caractérisé par une réponse seulement basée sur les valeurs du signal d'entrée. Donc, quel que soit le filtre, sa réponse impulsionnelle sera stable et de durée finie dépendante du nombre de cœfficients du filtre. ils sont cependanttrès largement utilisés car ils possèdent des propriétés uniques (phase, linéarité, stabilité,flexibilité) , Il peut aussi être appelé filtre non récursif ou encore filtre à moyenne mobile.

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

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

N est la longueur de la réponse impulsionnelle

{{{

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

On remarque qu'en exploitant l'équation (I.8) et que les coefficients b_i de filtre sont également les valeurs de la réponse impulsionnelle $h(n)$, qui se trouve donc être limitéedans le temps [4].

$$h(n) = \sum_{i=0}^{N-1} b_i \delta(n-i) \quad (1.10)$$

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

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

1.7.1 Structure de filtre RIF

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

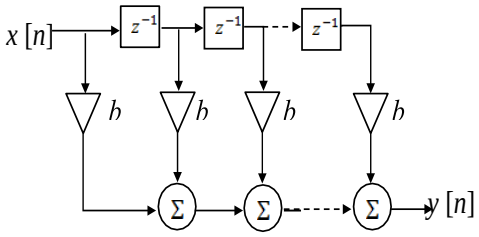


Figure 1. 5 Structure directe d'un filtre RIF.

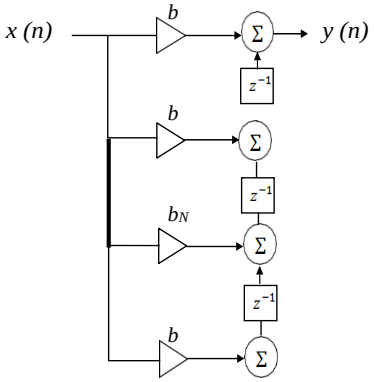


Figure 1. 6 Structure transposée d'un filtre RIF.

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

1.7.2 caractéristiques des filtres numériques RIF

Les termes suivants sont utilisés pour caractériser les performances et le comportement des filtres RIF :

- **Coefficients du filtre:** c'est un ensemble de constantes, utilisé pour multiplier ou pondérer les valeurs retardés de l'échantillon. Pour un filtre RIF.[5]
- **Réponse impulsionnelle:** Temps de réponse de la séquence du filtre lorsque l'entrée est une impulsion. Une impulsion est un échantillon unique de valeur égale à l'unité précédée et suivie par des valeurs nulles (zéro échantillons évalués). Pour un filtre RIF sa réponse impulsionnelle est l'ensemble des coefficients filtre. [5]
- **Tap :** Le nombre d'étage N du filtre RIF, généralement N, nous informe sur le type du RIF et surtout, sur quantité de mémoire nécessaire, et le nombre d'étapes de calculs. [5]
- **Mac :** Dans un contexte de filtrage utilisant les RIFs, un "MAC" est l'opération de

multiplication d'un coefficient par l'échantillon de données retardé correspondant et en accumulant le résultat. Il ya généralement un MAC par étage. [5]

1.7.3 Les avantages et les inconvénients des filtres RIF

Les avantages des filtres RIF

- Il existe des méthodes de conception de filtre RIF simple à mettre en oeuvre.
- Muni d'entrée-sortie bornée(BIBO)
- Stabilité , puisque la sortie est une somme d'un nombre fini de multiples finis des valeurs d'entrée, elle ne peut donc pas être supérieure à $\sum$ fois la plus grande valeur apparaissant dans l'entrée.
- Les Filtres RIF Peut facilement être conçu pour être une phase linéaire en rendant la séquence de coefficients symétrique. Cette propriété est parfois souhaitée pour les applications sensibles à la phase, par exemple les communications de données, la sismologie ,le traitement vidéo et musique ,les filtres croisés et la maîtrise .
- Coefficient de quantification. C'est une propriété importante pour avoir la mise en oeuvre d'un filtre sur un processeur DSP ou sur un circuit intégré.
- Les Filtres RIF ont également une faible sensibilité pour filtrer les erreurs.

Les inconvénients des filtres RIF

- Cher en réalisation.
- Le retard entre l'entré et le sorti (phase) peut être relativement long.

1.7.4 Conception de filtre

Un filtre RIF est conçu en trouvant les coefficients et l'ordre des filtres qui répondent à certaines spécifications, qui peuvent être dans le domaine temporel (par exemple un filtre adapté) et / ou le domaine fréquentiel (le plus courant). Les filtres adaptés effectuent une corrélation croisée entre le signal d'entrée et une forme d'impulsion connue. La convolution RIF est une corrélation croisée entre le signal d'entrée et une copie inversée dans le temps de la réponse impulsionnelle. Par conséquent, la réponse impulsionnelle du filtre adapté est "conçue" en échantillonnant la forme d'impulsion connue et en utilisant ces échantillons dans l'ordre inverse comme coefficients du filtre.[6]

Lorsqu'une réponse en fréquence particulière est souhaitée, plusieurs méthodes de conception différentes sont courantes:

- Méthode de conception de fenêtre.
- Méthode d'échantillonnage de fréquence.
- Méthode de la moindre MSE (erreur quadratique moyenne).

- Méthode Parks-McClellan (également connue sous le nom de méthode Equiripple, Optimal ou Minimax).
- L' algorithme d'échange Remez est couramment utilisé pour trouver un ensemble optimal de coefficients d'équiripple. Ici , l'utilisateur spécifie une réponse de fréquence désirée, une fonction de pondération pour les erreurs de cette réponse, et un ordre de filtre *N* . L'algorithme trouve alors l'ensemble des N+1 coefficients qui minimisent l'écart maximal par rapport à l'idéal. Intuitivement, cela trouve le filtre qui est aussi proche que possible de la

réponse souhaitée étant donné que seul N+1 des coefficients peuvent être utilisés. Cette méthode est particulièrement aisée en pratique puisqu'au moins un texte [7] comprend un programme qui prend le filtre et *N* souhaités, et renvoie les coefficients optimaux.

Les filtres RIF Equiripple peuvent également être conçus à l'aide des algorithmes FFT.[8] L'algorithme est de nature itérative. La DFT d'une conception de filtre initiale est calculée à l'aide de l'algorithme FFT (si une estimation initiale n'est pas disponible, h [n] = delta [n] peut être utilisé). Dans le domaine de Fourier ou domaine FFT, la réponse en fréquence est corrigée selon les spécifications souhaitées, et la FFT inverse est ensuite calculée. Dans le domaine temporel, seuls les N premiers coefficients sont conservés (les autres coefficients sont mis à zéro). Le processus est ensuite répété de manière itérative: la FFT est à nouveau calculée, correction appliquée dans le domaine fréquentiel et ainsi de suite.

Les progiciels tels que MATLAB , GNU Octave , Scilab et SciPy fournissent des moyens pratiques d'appliquer ces différentes méthodes.

1.7.5 Méthode de conception de fenêtre

Dans la méthode de conception de fenêtre, on conçoit d'abord un filtre IIR idéal, puis on tronque la réponse impulsionnelle infinie en la multipliant par une fonction de fenêtre de longueur finie . Le résultat est un filtre à réponse impulsionnelle finie dont la réponse en fréquence est modifiée par rapport à celle du filtre IIR. La multiplication de l'impulsion infinie par la fonction de fenêtre dans le domaine temporel entraîne la convolution de la réponse en fréquence de l'IIF avec la transformée de Fourier (ou DTFT) de la fonction de fenêtre. Si le lobe principal de la fenêtre est étroit, la réponse en fréquence composite reste proche de celle du filtre IIR idéal.

La réponse idéale est généralement rectangulaire et l'IIF correspondant est une fonction sinc Le résultat de la convolution du domaine fréquentiel est que les bords du rectangle sont effilés et que des ondulations apparaissent dans la bande passante et la bande d'arrêt. En travaillant en arrière, on peut spécifier la pente (ou largeur) de la région effilée (*bande de transition*) et la hauteur des ondulations, et ainsi dériver les paramètres du domaine fréquentiel

d'une fonction de fenêtre appropriée. La poursuite en arrière vers une réponse impulsionnelle peut être effectuée en itérant un programme de conception de filtre pour trouver l'ordre de filtre minimum. Une autre méthode consiste à restreindre l'ensemble de solutions à la famille paramétrique des fenêtres de Kaiser , qui fournit des relations de forme fermée entre les paramètres du domaine temporel et du domaine fréquentiel. En général, cette méthode n'atteindra pas l'ordre de filtrage minimum possible, mais elle est particulièrement pratique pour les applications automatisées qui nécessitent une conception de filtre dynamique à la volée.

Le procédé de conception de fenêtre est également avantageux pour créer des filtres demi-bande efficaces, car la fonction sinc correspondante est égale à zéro à tous les autres points d'échantillonnage (à l'exception du point central). Le produit avec la fonction de fenêtre ne modifie pas les zéros, donc près de la moitié des coefficients de la réponse impulsionnelle finale sont nuls. Une implémentation appropriée des calculs RIF peut exploiter cette propriété pour doubler l'efficacité du filtre.[9]

1.7.6 Méthode d'échantillonnage fréquentiel

Cette méthode [10] consiste à prélever un nombre fini d'échantillons de la réponse fréquentielle H(ω) à partir d'un certain gabarit.

La réponse fréquentielle est présentée par ses échantillons H(n), si H(ω) possède des discontinuités, elle ne sera pas présentée qu'approximativement par un nombre fini d'échantillons. Pour N coefficients de H(n) données, on peut calculer N valeur de la réponse impulsionnelle h(k) donnée par:

$$\text{H (K)} = TFD^{-1}\big(H(n)\big) = \frac{1}{N}\sum_{K=0}^{N-1} H(n)\, e^{+\frac{2\pi nK}{N}j} \tag{1.12}$$

on obtient la fonction de transfert par :

$$\text{H(z)} = \sum_{K=0}^{N-1} h\,(k) Z^{-K} = \frac{1-Z^{-N}}{N}\sum_{n=0}^{N=1} H(n)\frac{1-e^{-jN\omega}}{1-e^{-j\omega}e^{+j\frac{\pi n}{N}}} \tag{1.13}$$

La réponse fréquentielle du filtre dans ce cas est :

$$H_f(\omega) = H(z)\,|_{z=e^{j\omega}} = \frac{1}{N}\sum_{n=0}^{N-1} H(n)\ \frac{1-e^{-jN\omega}}{11-e^{-j\omega}e^{+j\frac{2\pi n}{N}}} \tag{1.14}$$

1.7.7 Méthode de Parks et McClellan

C'est la méthode optimale selon le critère de Chebyshev pour déterminer les coefficients d'un filtre RIF [11]. Cet algorithme, n'est qu'une de l'algorithme de Remez appliqué à la synthèse des RIF. Cette méthode, basée sur une distribution uniforme de l'ondulation sur l'ensemble de

la bande passante et sur distribution uniforme de l'affaiblissement sur l'ensemble de la bande de coupure, recherche itérativement les coefficients h(k) afin d'avoir, avec un ordre minimal, un filtre qui respecte aux mieux le gabarit fixé. Les filtres RIF résultant, appelés

aussi filtres minimax, possèdent une réponse en phase linéaire. Ils sont nettement plus performants que les filtres RIF fenêtrés et présentent par rapport à ces derniers les avantages suivants :

- Pour un gabarit identique, l'ordre d'un filtre RIF de Parks-McClellan est nettement inférieur.

- L'ondulation dans la bande passante et l'affaiblissement minimal dans la bande de coupure sont configurables séparément. Malgré ces avantages, cette méthode reste assez complexe à mettre en oeuvre en raison de son exigence en termes de coût de calcul.

L'ordre de McClellan et Parks donné par:

$$N - 1 = \frac{D_\infty(\delta_1,\delta_2) - F(\delta_1,\delta_2)(\Delta f)^2}{\Delta f} \quad (1.15)$$

1.8 Constitution d'un roulement

La majorité des roulements sont composés d'une bague intérieure et d'une bague extérieure, éléments roulants (billes ou rouleaux), et d'une cage. Les éléments roulants situés entre les deux bagues du roulement sont excentrique les uns des autres par la cage qui les guide et facilite leur rotation [12] (voir Figure 1.7).

C'est après le descriptif de chaque composant du roulement :

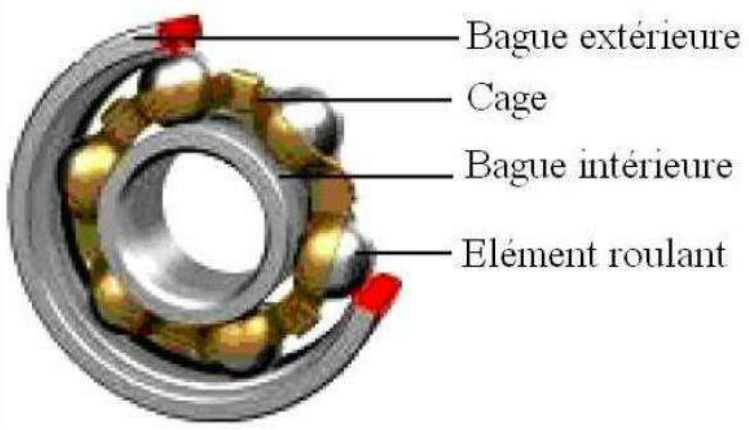


Figure 1. 7: composantes d'un roulement [13].

1.9 Les différents types de roulement

Il existe plusieurs type de roulement et sont classées selon le type de l'élément roulant

1.9.1 Les roulements à billes

Les roulements à billes sont classés en différentes catégories suivant la forme de leur bague (roulements à billes à gorge profonde, à contact oblique). Les roulements à billes sont les plus utilisés dans le monde industriel parce qu'ils présentent le meilleur rapport performance/prix [14].

Ces roulements peuvent être à une ou deux rangées de billes, à contact radial ou oblique. Les roulements à contact oblique permettent de supporter, en plus des efforts radiaux, des charges axiales plus importantes. Exemple de roulement à contact oblique [14].

1.9.2 Roulement à une rangée de billes à contact oblique :

Les roulements à contact oblique peuvent supporter une charge axiale, mais ne peuvent être montés seuls à cause de l'angle de contact. Ils doivent toujours être montés par paires ou combinés. Les roulements à billes à contact oblique à deux rangées (pistes doubles) ont un angle de contact est de 25°. Les roulements à billes à quatre points de contact peuvent supporter des charges axiales dans les deux directions. ls doivent être montés par paire et en opposition (voir Figure 1.8).

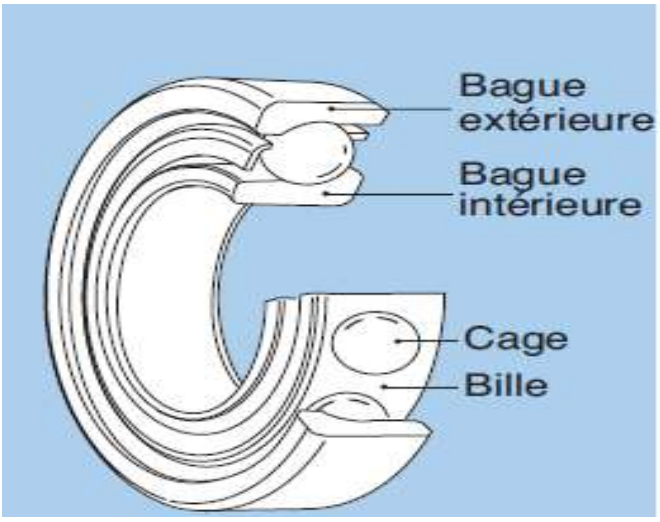


Figure 1. 8 **:** Roulement à une rangée de billes à contact oblique [14].

1.9.3 Roulement à billes à contact radial

Très économiques ce sont les plus utilisés en petites et moyennes .sous charge ils se comportent comme un roulement à contact oblique dont l'angle d'inclinaison sera variable

1.9.4 Les roulements à rouleaux

Les roulements à rouleaux sont classés suivant la forme des rouleaux (Roulements à rouleaux cylindriques, coniques)

Roulement à rouleaux cylindriques :

Les éléments roulants sont des rouleaux, ce qui donne une capacité de charge élevée. Les rouleaux sont guidés par les épaulements de la bague intérieure et/ou extérieure. Celles-ci peuvent être séparables pour faciliter le montage, et elles peuvent être montées en ajustement serre sur l'arbre et dans le logement. S'il n'y a pas d'épaulements, la bague concernée peut se déplacer librement en direction axiale. Les roulements à rouleaux cylindriques sont idéaux pour être utilisés comme des « paliers libres » qui encaissent la dilatation de l'arbre. Lorsqu'il

y a des épaulements, le roulement peut supporter une légère charge axiale entre les faces des rouleaux et les épaulements. La forme des faces de rouleaux et des épaulements a été adaptée pour augmenter la capacité de charge axiale. La conception du roulement de type de roulement rouleaux coniques augmente la capacité de charge radiale [15] (voir figure 1.9).

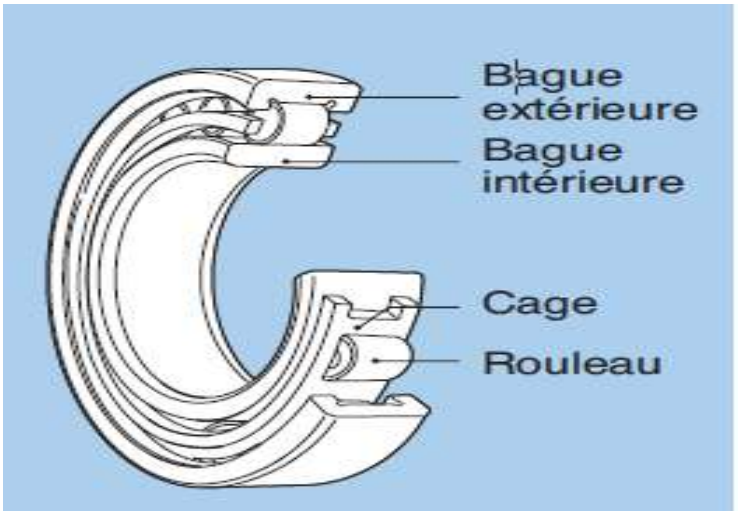


Figure 1. 9 : Roulement à rouleaux cylindriques [14].

Roulement à rouleaux coniques :

Les roulements à rouleaux coniques sont conçus de manière à ce que le chemin de roulement des bagues intérieures et extérieures et les génératrices des rouleaux coniques se coupent en un point qui se trouve sur l'axe du roulement. Les enroulements reçoivent des charges combinées provenant de l'intérieure et roulent en se servant de cet appui comme guidage. Une force induite se produit axialement lorsqu'une charge radiale est appliquée. Les roulements doivent donc être montés par paire. La bague intérieure à rouleaux et la bague extérieure est séparables, ce qui facilite le montage avec un jeu ou pré-charge. Le réglage du jeu interne (ou pré-charge) après montage est délicat et demande une attention particulière. Les roulements à rouleaux coniques sont capables de supporter de fortes charges axiales et radiales [13] (voir figure 1.10).

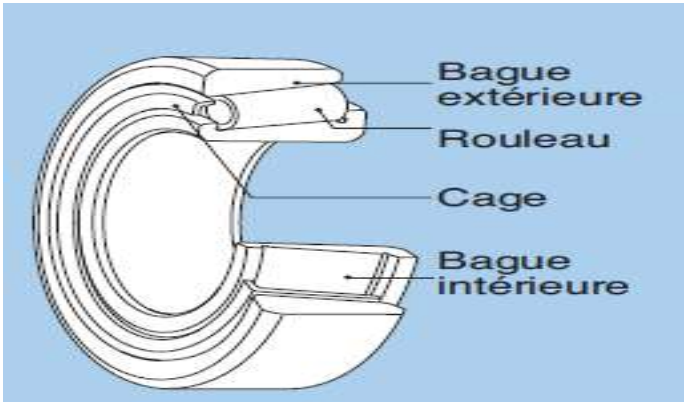


Figure 1. 10 **:** Roulement à rouleaux coniques [14].

1.9.5 Roulements radiaux et axiaux :

La plupart des roulements peuvent supporter à la fois des charges radiales et des charges axiales. Les roulements ayant un angle de contact inférieur à 45° supportent mieux les charges radiales que les charges axiales. Ils sont appelés roulements radiaux. Les roulements ayant un angle de contact supérieur à 45° supportent mieux les charges axiales que les charges radiales.

Ils sont appelés roulements axiaux ou butées. Il est possible de faire des combinaisons de roulements radiaux et axiaux qui possèdent les caractéristiques propres des roulements radiaux et des roulements axiaux [14].

1.9.6 Les roulements à aiguilles :

Les éléments roulants des roulements à aiguilles sont des aiguilles dont le diamètre maximal est de 5mm, et qui sont 3 à 10 fois plus longues que leur diamètre. La section des roulements est mince, mais la capacité de charge élevée malgré leur taille. Etant donné le grand nombre d'aiguilles, la rigidité de ces roulements est importante, et ils sont idéaux pour des mouvements d'oscillations et de pivotement .Il existe de nombreux types de roulements (voir figure 1.11**)**.

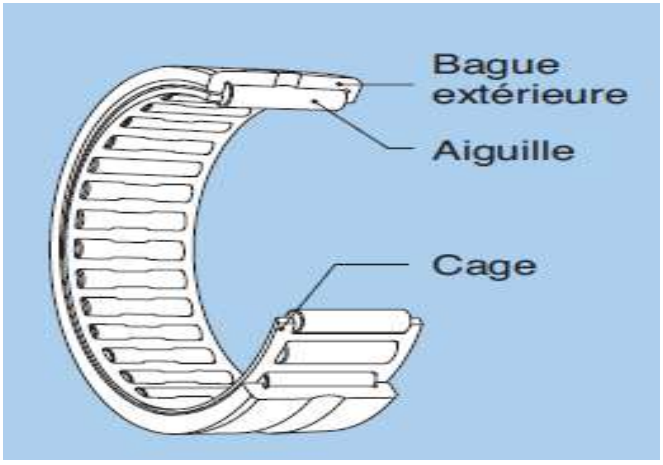


Figure 1. 11 **:** roulements à aiguilles [14].

1.9.7 Butées

Il existe de nombreux types de butées qui diffèrent en fonction de la forme des éléments roulants et des applications. La vitesse de rotation admise est généralement faible et il faut être attentif à la lubrification. En plus des modèles donnés ci-dessous, il y a de nombreux types de butées pour des applications spéciales. Pour plus de détail, veuillez consulter le catalogue concerné (voir figure 1.12).

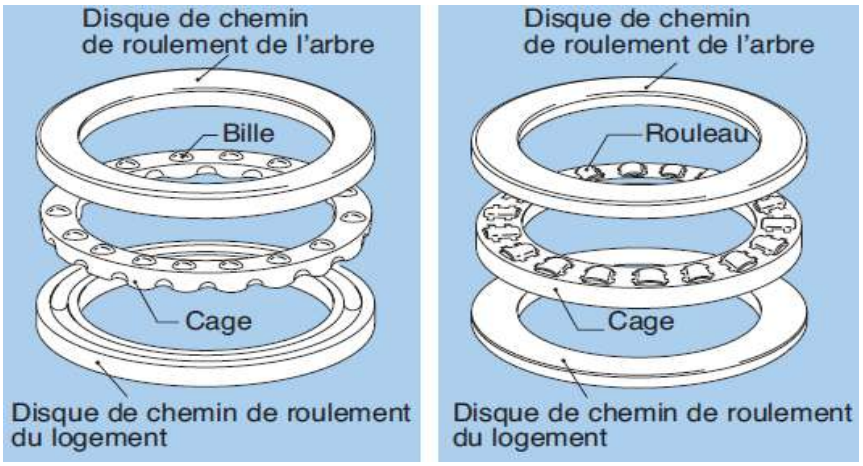


Figure 1. 12 **:** Butées [14].

Butée à billes :

Peu utilisées, elles supportent des charges axiales uniquement dans un seul sens. Les vitesses de rotation sont limitées à couse de la force centrifuge sur les billes

1.9.8 Roulements standard et spéciaux

Grâce à leur normalisation et leur interchangeabilité internationale, les roulements peuvent être employés de façon très rationnelle. C'est pourquoi il est conseillé d'utiliser des roulements standards avec un équipement mécanique. Par ailleurs, il est possible ou parfois nécessaire d'utiliser des roulements non normalisés et de fabrication spéciale permettant de garantir des fonctions spécifiques au sein d'une machine [14].

Dans notre étude en travaille sur Les roulements à billes, qui est le plus utilise dans le domaine industriel.

1.10 Origine des défauts de roulement

Il existe plusieurs défauts de roulement, les plus connus sont indiqué ces dessous :

- **Mauvaise lubrification :** Le mauvais choix d'un lubrifiant où des quantités inadaptées de lubrifiant causent environ **36%** des avaries de roulements. Une mauvaise lubrification réduite d'une manière significative la durée de vie d'un roulement et comme,

bien souvent, un roulement est un composant mécanique difficile d'accès, cela engendre des arrêts de maintenance longs et donc coûteux.

- **Pollution :** Un roulement est un composant mécanique de précision. Son bon fonctionnement risque d'être gravement compromis par la contamination des chemins de roulement et/ou du lubrifiant par des corps étrangers. On estime ainsi que 14% des défaillances de roulements sont dues à ce phénomène.
- **Mauvais montage :** Environ 16% des avaries de roulements sont dues à un mauvais montage ou à l'utilisation d'outils inadaptés [13].
- **Fatigue:** Les machines ne fonctionnent pas toujours dans des conditions optimales, loin s'en faut. Surcharges, entretiens insuffisants ou inadaptés sont responsables d'environ **34%** des défaillances de roulements. Un roulement qui fonctionne dans des conditions de surcharge. [13]

1.11 Quelques défauts de roulement

Les roulements sont parmi les composants les plus critiques des machines et représentent une source de panne fréquente. On peut noter deux types des défauts caractéristiques de la détérioration des roulements : les défauts dits naturels dus à la fatigue des roulements et des défauts dues à un mauvais montage ou une mauvaise utilisation du roulement [13].

En associant les différentes techniques de contrôle, l'expérience du personnel de la maintenance ainsi que l'historique des machines.

1.11.1Le grippage

Le grippage est un mode d'endommagement instantané. Les matériaux en contact se soudent sous pression et température généralement dans des conditions de vitesses moyennes et élevées. Il résulte d'un manque de lubrifiant ; les contacts métal-métal entraînent des échauffements qui facilitent les microsoudures et le transfert de métal [13].

Donc, on peut dire que le grippage est un phénomène caractérisé par des micros arrachements des particules métalliques (voir figure 1.13).

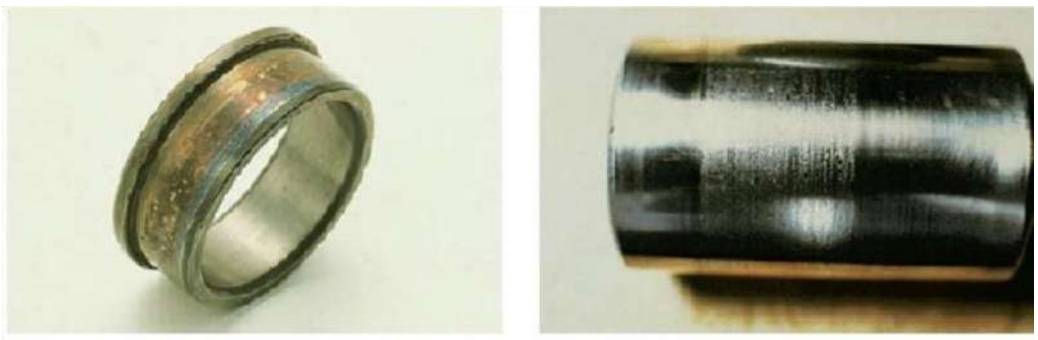


Figure 1. 13 : Grippage d'une bague [12].

1.11.2La corrosion

- La corrosion chimique se produit par la présence de l'humidité aux points de contact des éléments roulants avec les chemins. Surtout quand les roulements viennent d'être nettoyés et sont contaminés par la transpiration des mains.
- La corrosion de contact (fretting) apporte une pâte brune formée par la rouille et le lubrifiant. Une corrosion profonde entraîne la rupture des bagues. due au mauvais choix d'ajustements entre les bagues et les arbres.



Figure 1. 14 : Corrosion sur une piste de roulement [13].

1.11.3Usure

- Usure par déformation : C'est une déformation plastique des chemins des roulements, causée par des chocs importants ou une vibration à haute fréquence. C'est une usure sans perte de matière (présence des empreintes)
- Usure par abrasion : L'usure par abrasion donne aux roulements une couche brillante, due au glissement des éléments roulants sur les pistes, et ce par le manque ou une mauvaise lubrification.

1.11.4La fissuration des bagues

C'est l'apparu des fissures et des petites cassures sur les éléments du roulement.

Les coups lors du montage, le non-respect des portées de roulement (ajustement trop serré). Sont responsables sur ce type de détérioration, elle provoquées par une déformation plastique.

1.11.5L'écaillage

Le cas le plus rencontré est un défaut d'écaillage dû à la fatigue des roulements. Ce défaut survient sous l'effet de la fatigue due aux contraintes de cisaillement alternées qui sévissent en sous-couche. Le fonctionnement élastohydrodynamique (type de lubrification dépendant

l'élasticité du matériau) crée, par dépassement de la limite d'élasticité, des microdéformations permanentes et des fissures qui se propagent vers la surface et conduisent

à l'écaillage. L'écaillage est un processus continu qui s'accélère plus ou moins après apparition des premières fissures. On peut distinguer deux types d'écaillage :

- Ecaillage de fatigue : C'est un écaillage unilatéral, il est caractérisé par des petites fissures et arrachement de fragments de matière. Il se produit lors de l'application d'une surcharge, ou lors d'un défaut désalignement.
- Ecaillage superficiel : Causé par une concentration de contraintes, due à une contrainte accentuée (résultent probablement d'un défaut d'alignement). Ou bien par la modification de la zone de pré charge. Aussi par la présence de particules étrangère à l'intérieur du roulement

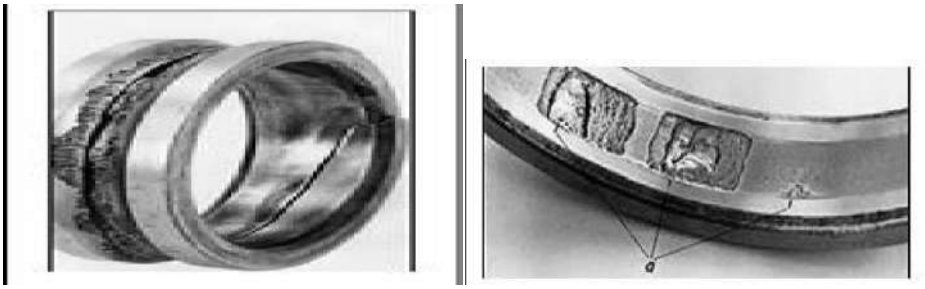


Figure 1. 15 : (a) écaillage de fatigue et (b) écaillage superficiel de la bague interne [13].

1.12 Fréquences caractéristiques pour les défauts d'un roulement

Les défauts : écaillage, grippage, corrosion (qui entraine l'écaillage) ; les avaries les plus fréquentes sur les roulements sont les défauts d'écaillage. Lors de la mise en rotation, un train d'impulsion est généré par ces défauts, à une fréquence bien déniée que l'on appelle "fréquence caractéristique" de défaut du roulement. Ce signal périodique est l'origine de nombreuses méthodes de détection de défaut de roulement. Les fréquences caractéristiques sont déterminées à partir de la géométrie du roulement et de la cinématique de la machine étudiée

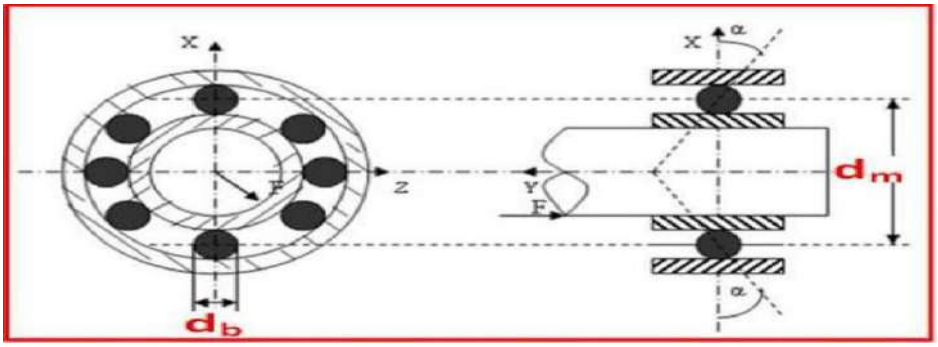


Figure 1. 16 : Caractéristiques d'un roulement.

- n : le nombre d'éléments roulants (billes, rouleaux ou aiguilles).

- dm : le diamètre primitif.

- db: le diamètre des éléments roulants.

- α : angle de contact.

1. Un défaut sur la bague extérieure générera un signal contenant des chocs espacés de la période de passage des billes sur la bague extérieure, la fréquence de ces chocs est appelée BPFO (Ball Passage Frequency Outer race).

$$\boldsymbol{BPFO} = f_{outer}(Hz) = \frac{n}{2} F_r(1 - \frac{d_b}{d_m}\cos\alpha) \quad (1.16)$$

Fr : est la vitesse de rotation.

2. La fréquence de passage d'un élément roulant sur un défaut de bague intérieure, cette fréquence est BPFI (Ball Passage Frequency Inner race).

$$\boldsymbol{BPFI} = f_{inner}(Hz) = \frac{n}{2} f_r(1 + \frac{d_b}{d_m}\cos\alpha) \quad (1.17)$$

3. Pour un défaut sur une bille, la vibration sera générée à deux fois la fréquence de rotation de la bille BSF (Ball Spin Frequency), puisque la bille sur un tour cognera sur la piste externe et interne :

$$\boldsymbol{BSF} = f_{ball}(Hz) = f_r \frac{n}{2} [1 - (\frac{d_b}{d_m}\cos\alpha)^2] \quad (1.18)$$

Le calcul de ces fréquences caractéristiques nécessite une connaissance précise des caractéristiques dimensionnelles du roulement.

Notre travail consiste a détecté ces defaults en analysons les signaux bruts et faire correspondre le signal finale à la fréquence caractéristique du roulement**.**

1.13 Conclusion

Dans ce chapitre en premier temps on a vu d'une façon générale les filtres numériques, ses avantages et ses inconvénients, en deuxième temp on a vu un aperçu sur les filtres RIF, et on a présenté trois méthodes pour la synthèse des filtre RIF (méthode de fenêtrage temporel, méthode d'échantillonnage fréquentiel et la méthode de Park et McClellan.

Et nous avons présenté une étude théorique des organes mécaniques des machines tournantes et leurs défauts. Ce dernier permet de donner une idée sur les impacts des défauts présentés par les signaux à analyser.[16]

Dans notre cas, on a basé sur le filtrage du signal brute des défauts du roulement sur la bague intérieure et extérieure et leurs déférentes fréquences vibratoires.

CHAPITRE 02

L'OUTILS DE DEVELOPPEMENT

HLS VIVADO

2.1 Introduction

Ces derniers temps, les utilisations et les applications de HLS sont évolués, pour augmenter l'efficacité et la qualité des conceptions basées sur FPGA, pour répondre aux aspirations et aux attentes des concepteurs de divers dispositifs. Pour en savoir plus sur HLS, nous abordons dans ce chapitre en détail ce qu'est HLS .

2.2 Historique et évolution de la HLS :

La synthèse de haut niveau a commencé vers les années 1980 par les travaux sur les méthodes heuristiques (mathématiques) visant la conception de compilateurs [17] et d'outils de synthèse destinés aux circuits numériques et aux circuits intégrés à très grande échelle « VLSI » [18] [19] [20]. La naissance du premier circuit FPGA à base de mémoire SRAM (*static random-access memory*), mise en maché par Xilinx en 1985 [21], ainsi que l'apparition des langages de description de matériel (HDL) au niveau RTL (*Register Transfert Logic*) tels Verilog [22] et VHDL [23] ont mené à l'élaboration d'une suite d'outils automatique pour la synthèse, le placement et le routage des circuits numériques [24].

Les HDL sont encore largement utilisés comme entrées aux outils de simulation et de synthèse logique, conduisant alors à la définition de leurs circuits synthétisables. La synthèse à partir d'une description de matériel à un plus haut niveau d'abstraction que le RTL vise à remédier au problème de complexité des circuits et des systèmes.

L'approche HLS utilise comme entrée une description de l'application au niveau système.Cette entrée permet une description sous la forme logicielle classique (exemple C++) en omettant les détails d'implémentation matérielle tels que les contraintes de temps (*design constraints*) et la nature des interfaces utilisées par une telle application. Dans cette thèse la synthèse de haut niveau est définie comme étant la génération d'une spécification synthétisable décrite avec un langage matériel (VHDL ou Verilog) à partir d'une spécification décrite au niveau système avec un langage de haut niveau (C/C++, OpenCL) [25].

Les travaux de recherche sur la synthèse de haut de niveau se sont intensifiées par l'imbrication de plusieurs sous-domaines, notamment la conception conjointe matérielle/logicielle, l'estimation de performances, l'exploration de l'espace de conception, le partitionnement sur plusieurs éléments de calcul, l'interfaçage entre les éléments de calcul, la covérification et cosimulation matérielle/logicielle. D'ailleurs, l'intérêt des chercheurs aux domaines qui sont fortement liés à la synthèse de haut niveau a pris de l'ampleur, de manière que le nombre d'articles scientifiques qui traitent les problèmes liés à la synthèse de haut niveau

ne cessent d'augmenter significativement. Ceci montre l'envergure de la recherche sur la synthèse de haut niveau en tant qu'approche de conception pour les circuits FPGA et les plateformes hétérogènes configurables. Parmi les résultats de ces recherches, on trouve la naissance de plusieurs outils HLS fournis par le milieu académique tel l'outil LegUp [26] ou les outils commerciaux Vivado HLS [27] ou Intel HLS compiler [28]. Par ailleurs, Intel offre l'outil *Intel FPGA SDK for OpenCL* [29] qui permet l'implémentation des applications élaborées en OpenCL sur des FPGA. Cette thèse se limitera à l'exploration d'OpenCL, un des langages les plus répandus dans ce domaine, en vue de son adoption comme un langage de programmation des plateformes hétérogènes reconfigurables de type FPGA+CPU.

2.3 La génération classique et la génération en HLS :

La conception d'un circuit passe par plusieurs étapes pour obtenir un FPGA mettant en œuvre l'application souhaitée. Deux méthodes existent pour la génération de circuits FPGA : l'approche classique se basant sur les transferts entre registres (Register Transfer Level ou RTL) et l'approche de haut niveau se basant sur la synthèse de haut niveau (High Level Synthesis ou HLS).

2.3.1 Flot de conception RTL :

Le flot de conception classique par RTL est représenté dans la Figure 2.1 [30]. Le point d'entrée pour la conception classique d'un circuit FPGA est la description RTL, où à partir d'un algorithme le développeur écrit manuellement une première version en langage de description matérielle, tel que VHDL, Verilog, ou SystemVerilog . A partir de cette description, plusieurs étapes sont nécessaires pour arriver à la génération du circuit souhaité, où l'ensemble des étapes est appelé synthèse. Tout d'abord, le code RTL passe par une phase de synthèse logique, où le RTL est traduit à un ensemble d'éléments logiques interconnectés appelé netlist. Cette netlist peut être composé d'additionneurs, de blocs mémoires, etc. Différents logiciels de synthèse existent comme XST de l'environnement ISE de Xilinx, Quartus II d'Altera, etc. Cette étape donne aussi l'utilisation en surface et la fréquence.

Si les caractéristiques matérielles (surface et fréquence) ne correspondent pas aux contraintes et aux besoins, une modification de la description RTL est nécessaire pour chercher une meilleure solution. Après la phase de la synthèse logique, il y a la phase d'implémentation physique pour une cible FPGA donnée.

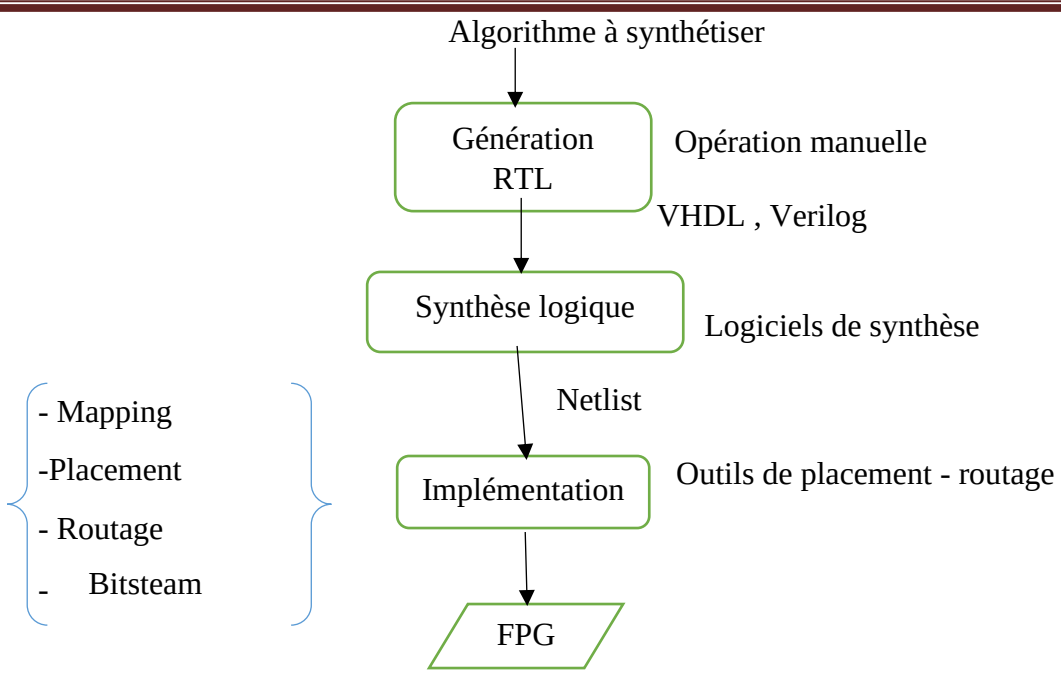


Figure 2.1: Flot de conception RTL.

Cette phase se compose du mapping, du placement et du routage. Le mapping se charge de traduire la netlist en des composants dans le FPGA,généralement des LUTs et des FFs. Le placement et le routage consistent à placer les ressources dans le FPGA et à les relier entre elles par des interconnexions. Le résultat final obtenu est un bitstream, qui est un flux de données binaires à charger dans le FPGA et permettant sa programmation.

2.3.2 Flot de conception HLS :

La conception à base de synthèse de haut niveau consiste à automatiser la conception du RTL [31]. Le point d'entrée d'un flot de HLS (voir Figure 2.2) est une description dans un langage de haut niveau (C, C++). L'utilisateur peut donner des indications spécifiques (types des opérateurs, le partage des ressources, etc.) et en sortie, un ou plusieurs fichiers sont obtenus décrivant l'application dans un langage de description matérielle (VHDL, Verilog, etc). A la génération, les caractéristiques matérielles (surface, fréquence et latence) sont données. Ainsi l'utilisateur peut garder la solution ou essayer de l'améliorer, selon le besoin, en utilisant des commandes spécifiques du logiciel de HLS. Ces outils HLS permettent par conséquence une exploration rapide de l'espace des solutions.

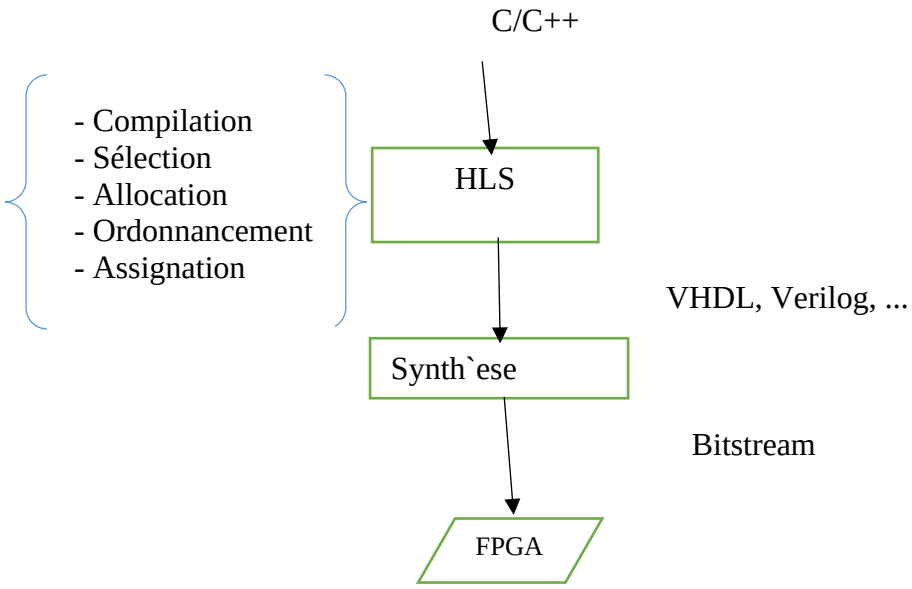


Figure 2.2: Flot de conception HLS.

Les étapes de la synthèse de haut niveau sont (voir Figure 2.2) :

a) **La compilation** permet de vérifier la syntaxe et la sémantique et traduire la description en une représentation intermédiaire. Ainsi elle se charge des optimisations telles que la propagation de constantes et la transformation de boucles [32].

b) **La sélection permet de définir les types d'opérateurs à partir de la bibliothèque de** composants matériels. Le choix des opérateurs dépend des indications en surface, vitesse, consommation, etc. [33].

c) **L'allocation** permet de fixer le nombre d'instances nécessaires dans l'architecture

finale [34]. Cela peut se faire par l'utilisateur dans le cas de contraintes de ressources ou automatiquement dans le cas de contraintes de temps.

d) **L'ordonnancement** permet d'affecter l'instant de début d'exécution de chaque opérateur respectant les contraintes exigées et les dépendances de données fournies par la compilation [35].

e) **L'assignation** permet d'associer une instance d'opérateur à chaque opération et un registre à chaque variable [36].

Plus de détails sur le flot de conception de haut niveau et ses spécificités sont donnés dans [37].

De nombreux outils existent pour la synthèse haut niveau, nous pouvons citer quelques-uns :

— **Handle-C** [36], créé en 1996, permet de définir des sections parallèles et de construire des canaux de communication (FIFOs) entre les unités.

Il supporte aussi les types de variables et signaux avec des précisions au niveau bit.

— **Stream-C** [39], basé sur le modèle de programmation

Communicating Sequential Process (CSP) de Hoare [40] et cible des applications opérant sur des flux de données.

— **SPARK** [41] transforme du C en VHDL. Cette HLS a été conçu pour étudier les impacts des transformations et des heuristiques appliqués à la génération du code VHDL à partir de code C.

— Catapult [42] est un des outils industriels les plus répandus. Il prend en entrée du C++ ou du SystemC et génère du code RTL pour FPGA ou ASIC. Ces dernières avancées sont l'intégration de méthodes de vérification du code généré.

— **Vivado Xilinx HLS** [43] fait partie de la suite d'outils Xilinx. Il se base sur un environnement de développement (EDI) Eclipse, intégrant la compilation et la simulation RTL. Il génère du SystemC, VHDL et Verilog. Il est possible d'exporter directement une application sous forme de bloc IP depuis le HLS, facilement int´egrables dans un projet.

Dans nos travaux, nous nous appuyons sur l'outil Vivado HLS de Xilinx, pour l'implémentation d'un accélérateur FPGA sur SoC du calcul de l'inférence Bayésienne. Dans la section suivante, nous donnons le flot de conception pour FPGA à base de Vivado HLS ainsi que les différentes spécifications du logiciel.

2.3.3 Flot de conception SoC-FPGA :

Les processeurs et les FPGA sont les noyaux les plus performants de la plupart des systèmes embarqués. Parmi les améliorations les plus récentes du monde FPGA, on retrouve les périphériques SoC- FPGA (System on Chip- FPGA).

Un SoC FPGA intègre un noyau de processeur dur et une logique programmable sur le même circuit. L'intégration de la fonctionnalité de gestion de haut niveau des processeurs et des opérations en temps réel, le traitement des données ou les fonctions d'interface d'un FPGA dans un seul dispositif forment un système embarqué encore plus puissant (voir l'architecture générale dans la Figure 2.3). Les trois plus grands fournisseurs FPGA, Xilinx, Altera et Microsemi (Actel) commencent tous à fabriquer de tels circuits.

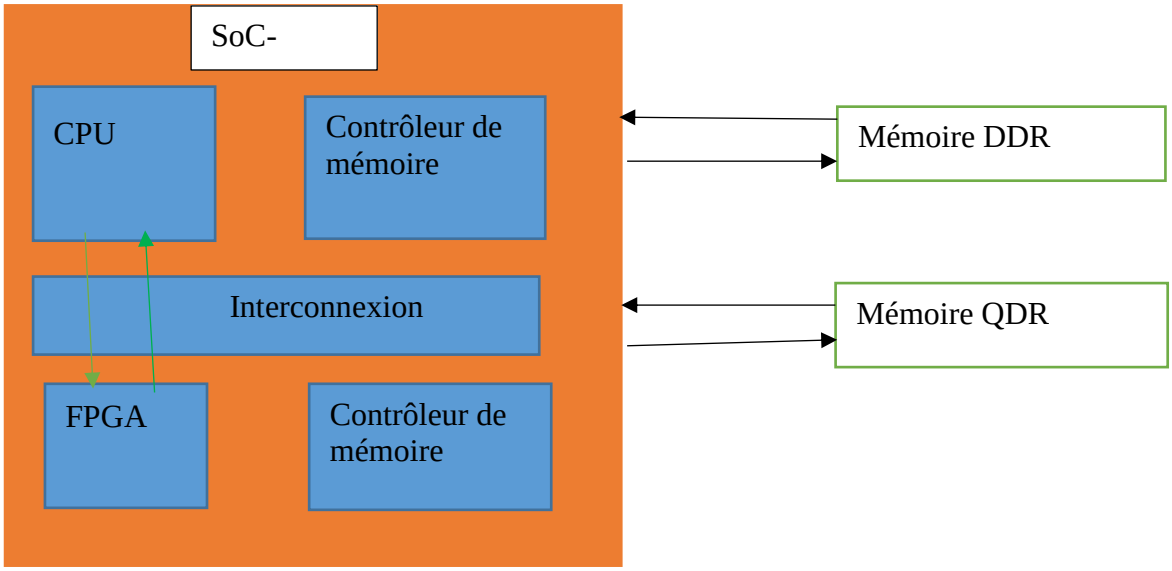


Figure 2.3: Architecture SoC-FPGA

Le flot de conception pour un SoC se base sur ce qu'on appelle le Co-Design logiciel/matériel Le Co-Design permet d'exploiter le partage matériel et logiciel dans le but de respecter les contraintes de conception telles que le coût, les performances et la consommation énergétique [44].

La Figure (2.4) représente le flot de conception pour un SoC. Dans l'étape de partitionnement, le choix entre les parties à réaliser en matériel et les parties en logiciel est défini. Par la suite, la synthèse logicielle consiste en la génération du code exécutable sur le processeur et la synthèse matérielle à générer le bitstream pour l'exécution sur le FPGA. La synthèse d'interface est une étape essentielle, où les transferts de données entre la partie matérielle et la partie logicielle sont gérés, ainsi que les protocoles et les modes de communication.

Les outils de synthèse de haut niveau permettent de simplifier ce flot, en utilisant des directives, que nous détaillons par la suite avec les outils Xilinx.

Xilinx SDSoC est un nouvel environnement qui permet aussi une estimation des ressources et des performances avec des propositions d'architecture, ainsi que pour le partitionnement.

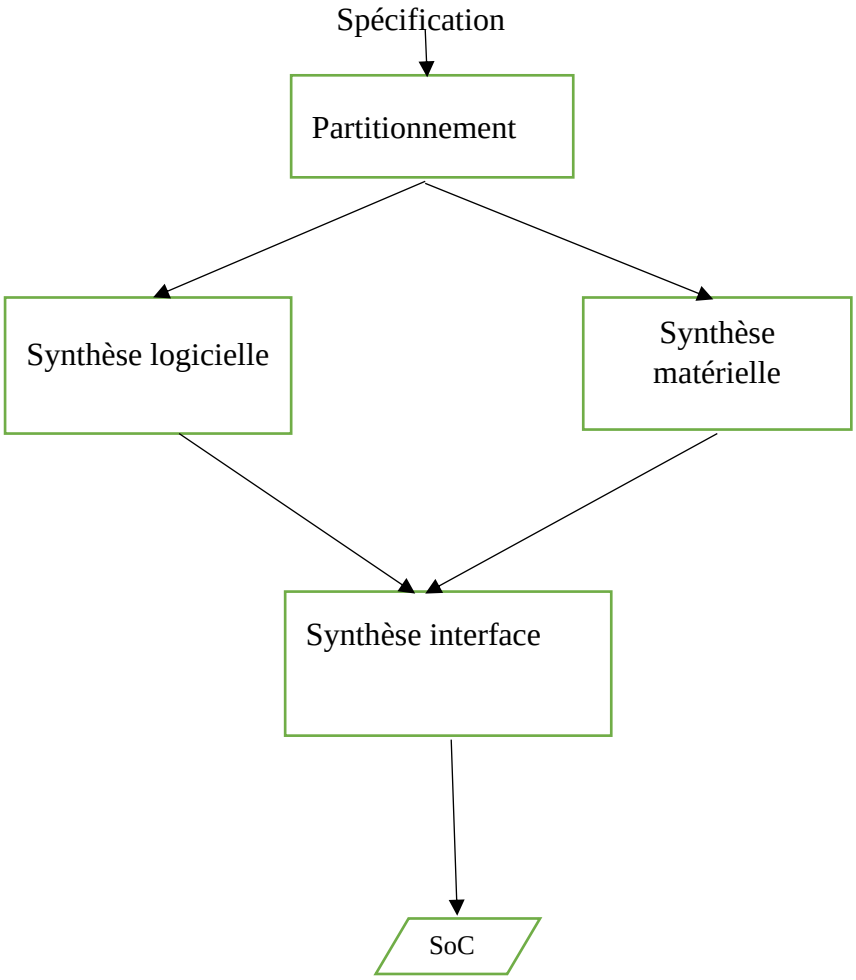


Figure 2.4: Flot de conception SoC.

2.4 La chaine opérationnelle sous vivado HLS :

Les différentes opérations définir par l'organigramme suivant (figure 2.5) :

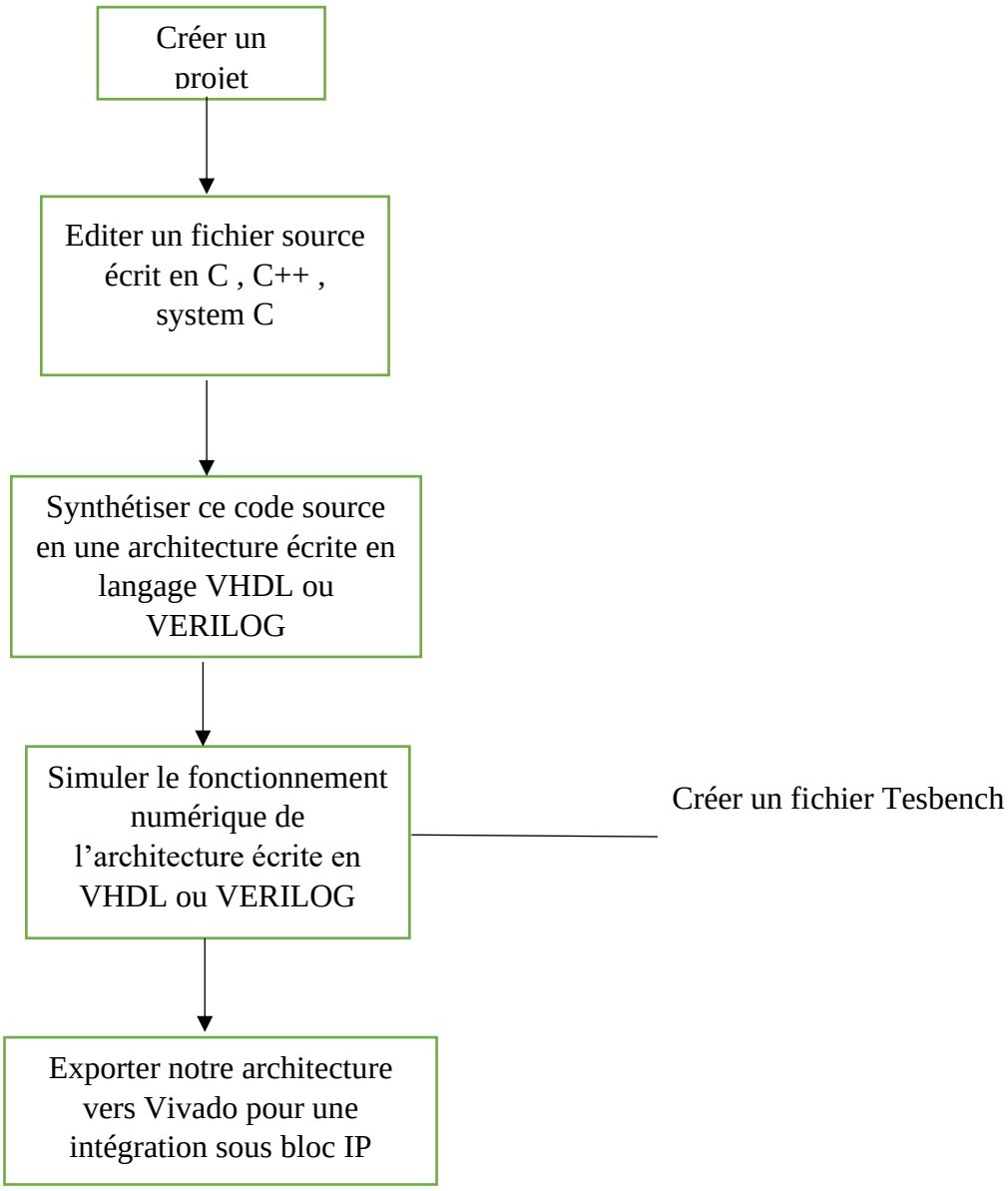


Figure 2.5 : chaine opérationnelle sous vivado HLS

L'écriture d'un programme en C, C++ ou system C doit répondre aux exigences de la synthèse. Durant cette première partie nous allons décrire de manière théorique les éléments qui nous permettrons par la suite de coder un programme C ou C++ en toute connaissance de cause .

2.5 Les éléments de base sous HLS :

2.5.1 Les interfaces :

C'est l'élément essentiel qui va nous permettre par la suite de créer des interfaces d'entrées, sorties, entrées sorties sur notre bloc IP.

On peut être amené à créer deux types d'interfaces :

a) Les interfaces de synthèse qui sont réalisées en utilisant les arguments d'une fonction en langage C ou C++. Les interfaces de synthèse permettent de communiquer notre IPBLOC à d'autres IPBLOCS ou vers le processeur

b) Les interfaces de block Level qui sont liées aux noms de la fonction de haut niveau en langage C. Ce type d'interface permet de contrôler l'exécution d'un sous-système de contrôler et de coordonner les échanges d'information entre chaque sous –système

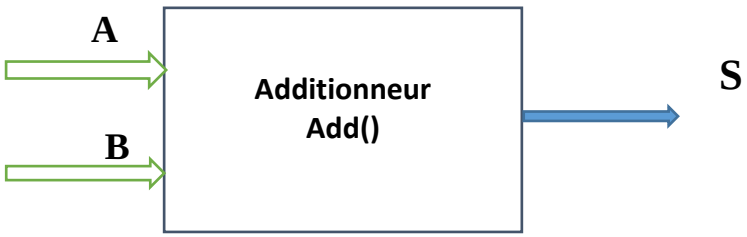
2.5.2 Les interfaces de synthèse :

Elles sont donc créées grâce aux arguments des fonctions. Suivant les opérations effectuées par chaque argument ou paramètres d'entrées de la fonction, automatiquement le synthétiseur détermine la direction de l'interface ou du port.

Exemple :

```
Void Add (  int A , int B, Int *S)
{
*S= A+B ;
}
```

Après synthèse les arguments A, B, S deviennent des interfaces de synthèse.



Chaque port ou interface obéit à un protocole de communication que l'on doit choisir sur HLS

* Un passage de paramètre par valeur ne peut être qu'une entrée

* Un passage par référence peut devenir une entrée ou une sortie

* Une valeur de retour devient une sortie

2.5.3 Les principaux protocoles d'interface sous HLS :

Ap_none : C'est l'un des protocoles les plus simples avec aucun protocole d'interface et aucuns signaux de contrôle additionnels

Ap-ack : Ce protocole se comporte différemment suivant si l'interface est en entrées ou en sorties. Pour les entrées un port d'acquittement de sortie est additionné et maintenu à l'état haut sur le même cycle d'horloge que l'entrée est lue. Pour un port en sortie un port d'acquittement en entrée est additionné.

Ap_hs : le suffixe HS veut dire handshaking ou poignée de main. Ce protocole permet un processus de poignée de main entre un producteur et un consommateur de données. Ils nécessitent deux ports de contrôle

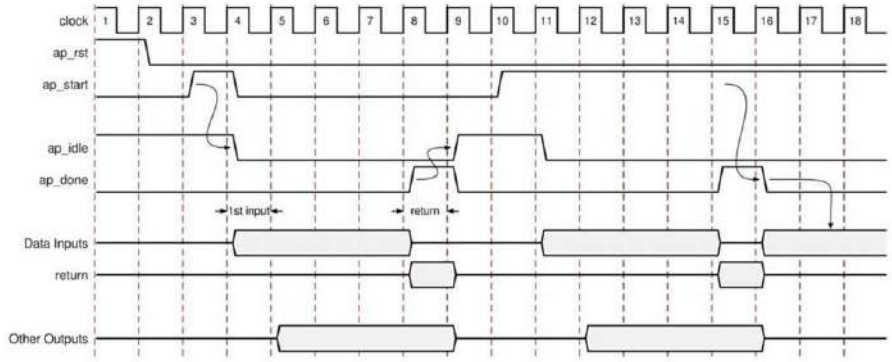


Figure 2.6 : comportement de l'interface AP_HS

Ap_memory : Ce protocole basé sur une mémoire permet un accès aléatoire à une mémoire. Le seul argument compatible avec ce protocole est le type tableau. Ils nécessitent des signaux de validation d'horloge et d'adresses.

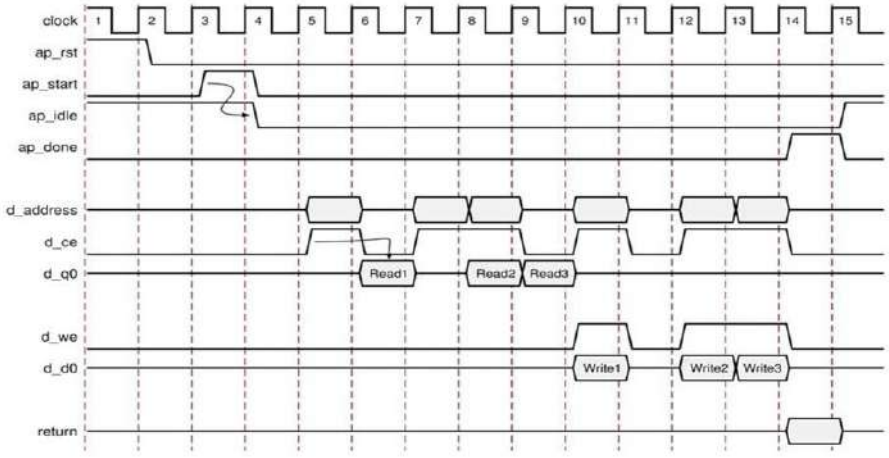


Figure 2.7 : comportement de l'interface AP_MEMORY

AP_FIFO : Ce protocole basé sur une mémoire FIFO permet un accès séquentiel à une mémoire, ils ne nécessitent donc pas d'adresse

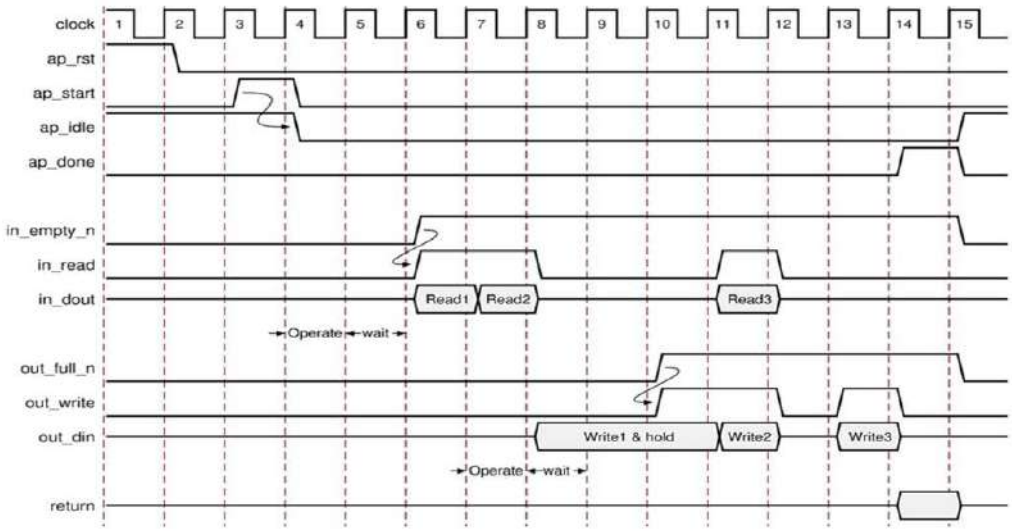


Figure 2.8 : comportement de l'interface AP_FIFO

AP_BUS : celui-ci est une simple interface de bus générique

Le bus **AXI** qui est du type AXI STREAM

Le bus **AXILITE**

Le bus **M_axi** qui spécifie une interface du type AXI4 maitre

Le tableau (2 .1) ci-dessous spécifie **les restrictions** sur le passage des paramètres pour chacun des protocoles :

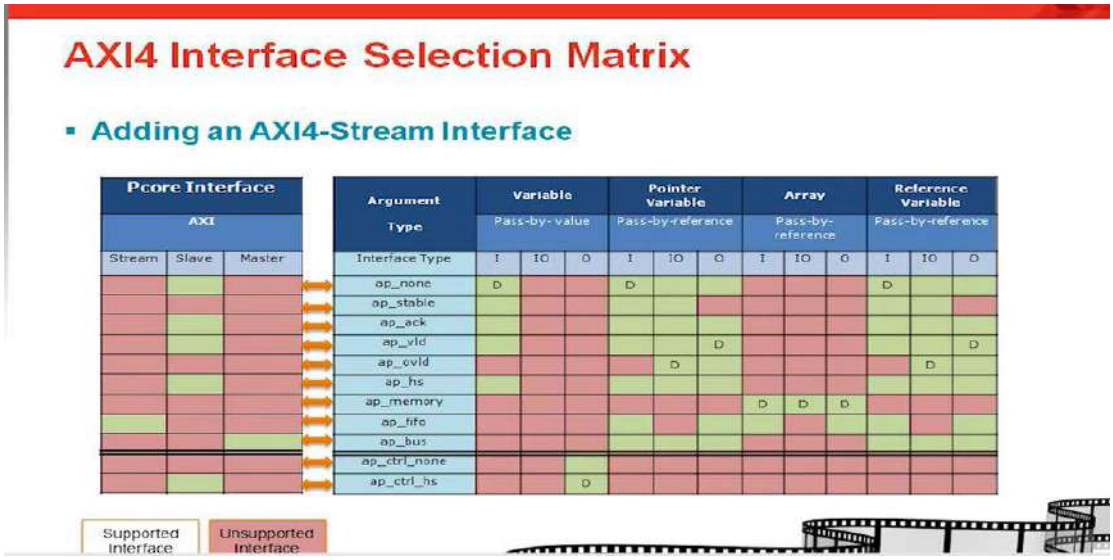


Tableau 2.1 : les restrictions sur le passage des paramètres pour chacun des protocoles

.

2.5.4 Les types de données en langage C ou C++ sur Vivado :

On peut utiliser les mêmes types qu'en langage C ou C++. Si l'on apporte une certaine précision sur le format des données on peut utiliser la syntaxe suivante :

Pour les entiers

Language	Integer Data Type	Description	Required Header
C	int N (e.g .int7)	signed integer of N bits precision	#include "ap_cint.h"
	uint N (e.g .uint7)	unsigned integer of N bits precision	
C++	Ap_int<N> (e.g .ap_int<7>)	signed integer of N bits precision	#include "ap_int.h"
	Ap_uint<N> (e.g .ap_uint<7>)	unsigned integer of N bits precision	

Tableau 2.2 : les entiers en langage C ou C++ sur vivado

```c
// C code example
#include "ap_cint.h"

void top_level_function (..)
{
     // declarations
       int6 small_signed;
       uint10 big_unsigned;
       int22 vbig_signed;
       ...
}
```

```c
// C++ code example
#include "ap_int.h"

void top_level_function (..)
{
     // declarations
       ap_int<6> small_signed;
       ap_uint<10> big_unsigned;
       ap_int<22> vbig_signed;
       ...
}
```

Les flottants à virgule fixe

Language	**Fixed Point Data Type**	**Description**	**Required Header**
C++	**ap_fixed<W,I,Q,O,N>**	**Signed fixed point number of I integer bits and W-I fractional bits.**	**#include "ap_fixed .h"**
	ap_ufixed<W,I,Q,O,N>	**unsigned fixed point number of I integer bits and W-I fractional bits.**	

Tableau 2.3 les flottants à virgule fixe

Le I correspond au nombre de bits de la partie entière W au nombre de bits total (partie entière + fractionnaire).

Les autres lettres ne seront pas décrites

2.5.5 Les interfaces de block level :

Elles sont créées en utilisant le nom de la fonction de plus haut niveau en langage C ou C++. La synthèse d'interface au niveau bloc applique un protocole d'entrées sorties à un IP bloc en additionnant des signaux de contrôle afin de contrôler les opérations de l'IPbloc. Une synthèse au niveau bloc est contrôlé par les modes d'interface ap_ctrl_hs et ap_ctrl_none qui sont appliqués à la fonction ou port de retour de la fonction.

Si un protocole de niveau bloc est utilisé il opère indépendamment des protocoles des interfaces de synthèse qui ont été adopté sur chaque port individuel. Il y a aussi deux signaux d'entrées qui sont rajoutés *ap_clk* et *ap_rst*. Ces deux signaux sont nécessaires à cause des opérations internes des blocs qui sont synchrones.

Il est important de noter qu'une interface de bus AXI4_lite Peut-être appliquée aussi pour un protocole d'interface de niveau bloc.

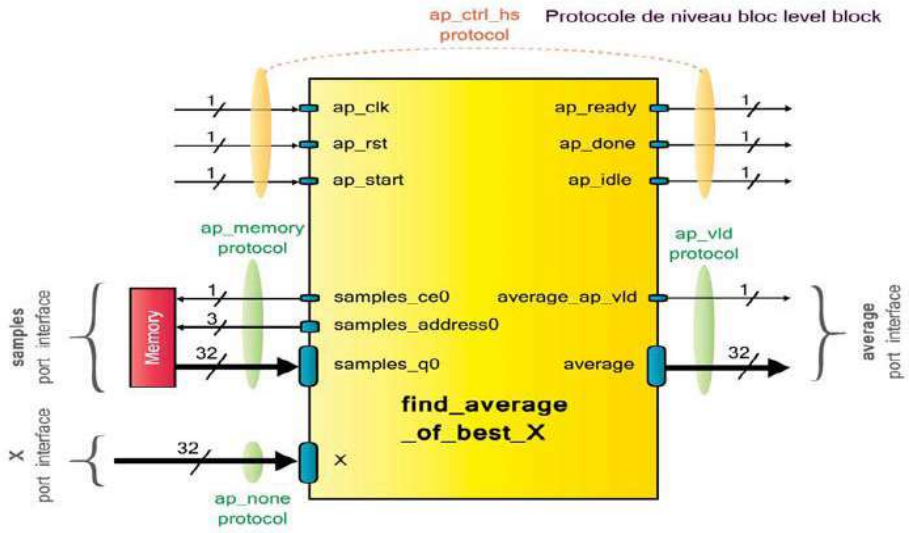


Figure 2.9 : protocole d'interface de niveau bloc .

ap_axi_s is a signed interpretation of the AXI stream

• ap_axi_u is an unsigned interpretation of the AXI stream

```
template<int D,int U,int TI,int TD>
struct ap_axis{
ap_int<D> data;
ap_uint<D/8> keep;
ap_uint<D/8> strb;
ap_uint<U> user;
ap_uint<1> last;
ap_uint<TI> id;
ap_uint<TD> dest;
};

template<int D,int U,int TI,int TD>
```

```
struct ap_axiu{
ap_uint<D> data;
ap_uint<D/8> keep;
ap_uint<D/8> strb;
ap_uint<U> user;
ap_uint<1> last;
ap_uint<TI> id;
ap_uint<TD> dest;
};
```

2.6 Mise en place d'un décodeur sur Vivado HLS :

Le traitement des flux de données est illustré ci-dessous

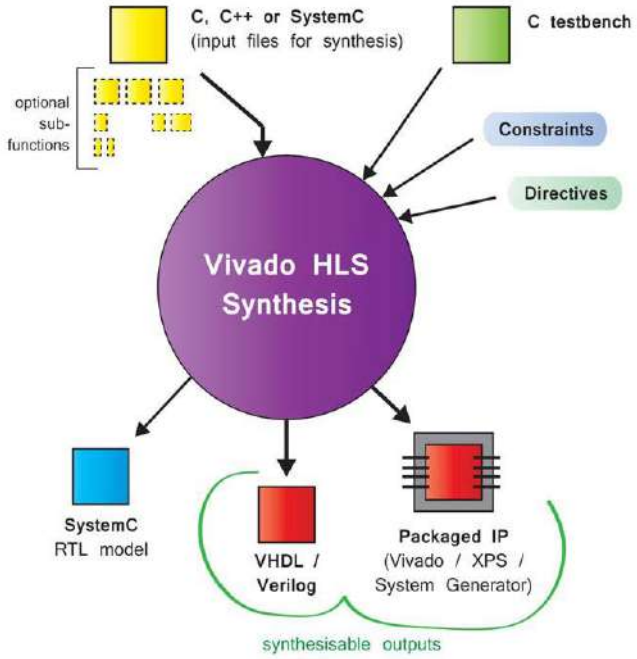


Figure 2.10 : Le traitement des flux de données

Le fichier de directive permettra d'établir sous forme de texte, les spécifications pour chacune des interfaces de synthèse. Le fichier C testbench fournira les signaux de test pour la validation de la synthèse

2.7 Flot de conception avec Vivado HLS de Xilinx et gestion des optimisations et interfaces :

Vivado HLS est un logiciel de Xilinx pour la synthèse de haut niveau [45]. Il fait la synthèse d'une fonction C (C, C++ ou SystemC) qui peut être intégrée dans un système matériel. Il offre un ensemble complet de fonctionnalités pour la mise en œuvre la plus optimale du programme C et il est facilement intégrable dans les outils de conception Xilinx.

La Figure (2.11) donne le flot de conception Vivado HLS avec les fichiers d'entrée et de sortie. Vivado HLS permet de :

— compiler, simuler et déboguer l'algorithme C,

— synthétiser l'algorithme C et donner la description RTL, avec ou sans directives d'optimisation des utilisateurs. Ces directives d'optimisation permettent de donner des indications spécifiques par rapport au partage des ressources, l'accès aux mémoires, etc., Nous détaillons cette partie par la suite.

— avoir des rapports complets et les capacités d'analyse,

— vérification automatisée de l'implémentation du RTL,

— préparer l'implémentation RTL dans un des formats disponibles.

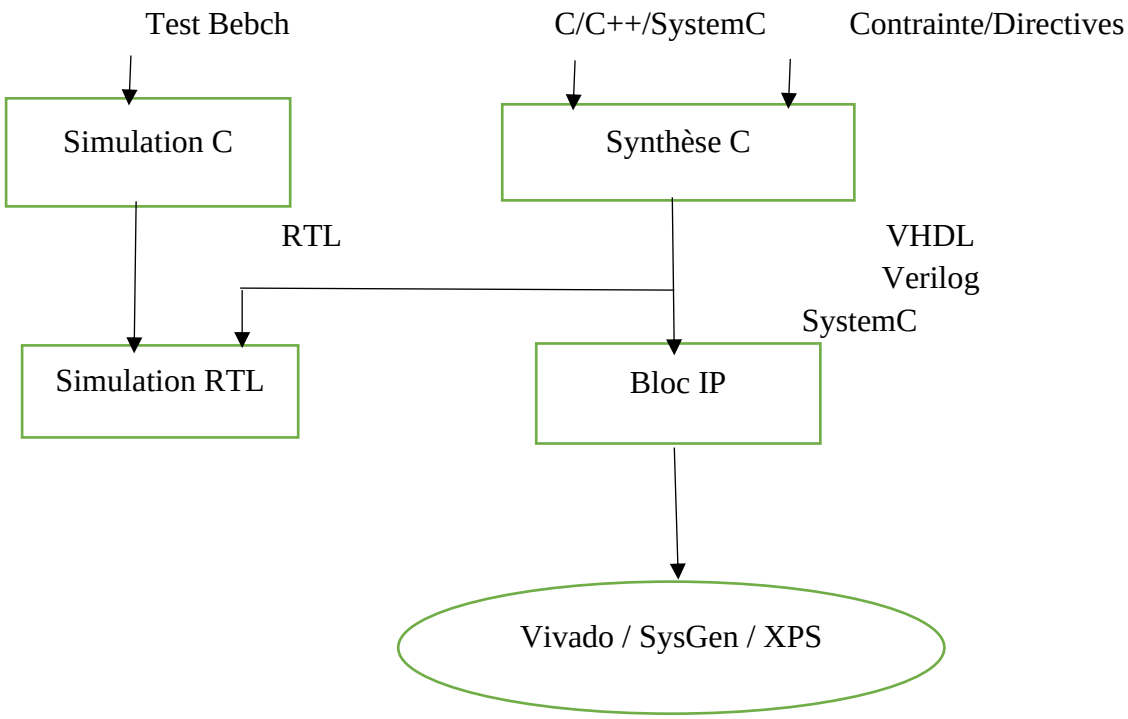


Figure 2.11: Flot de conception Vivado HLS.

L'allocation et l'assignation sont les processus au coeur de la synthèse de Vivado HLS. Nous expliquons cela sur l'exemple suivant :

```
int foo(char x, char a, char b, char c) {
char y;
y = x*a+b+c;
return y
}
```

Pendant le processus d'ordonnancement, le HLS détermine dans quelle opération du cycle d'horloge il faut se produire. Les décisions prises lors de la planification tiennent compte de la fréquence d'horloge, des informations de synchronisation de la bibliothèque des

périphériques cibles, et de toute directive d'optimisation spécifiée par l'utilisateur. Les directives d'optimisation sont détaillées dans la partie suivante.

Pour l'exemple, la phase d'ordonnancement est représentée dans la Figure (2.12).

La multiplication et la première addition doivent être exécutées dans le premier cycle d'horloge, puis la deuxième addition. Ainsi, la sortie est disponible à la fin du deuxième cycle d'horloge.

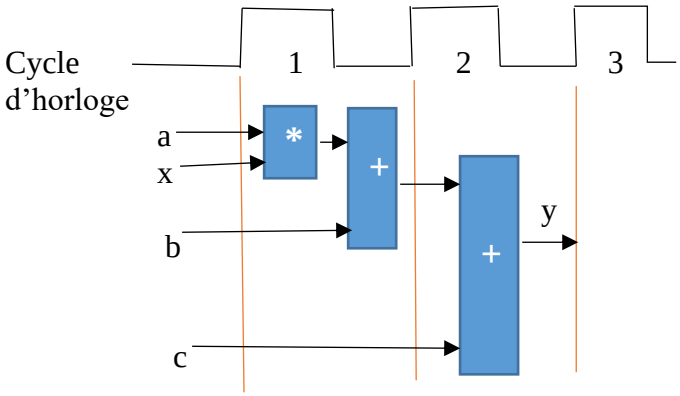


Figure 2.12: Phase d'ordonnancement.

La Figure 2.13 représente la phase d'assignation, où les ressources matérielles sont déterminées pour les opérations définies à l'ordonnancement. Il est possible d'utiliser un DSP48 pour implémenter à la fois une multiplication et une addition.

Une ressource DSP48 est un bloc de calcul disponible dans l'architecture FPGA qui offre l'équilibre idéal de haute performance.

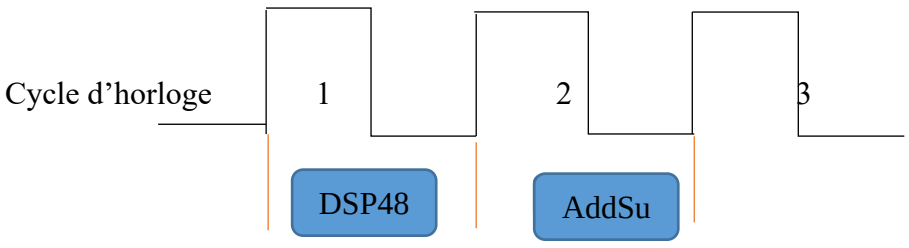


Figure 2.13 : Phase d'assignation.

Pour les entrées/sorties (I/O) de cet exemple, ce sont des ports de données simples. Etant donné que chaque variable d'entrée est un type char, les ports de données d'entrée sont tous sur 8 bits. Le résultat de la fonction est un type int de 32 bits et le port de données de sortie est de 32 bits.

Nous pouvons voir les avantages de l'implémentation du code C dans le matériel.

Toutes les opérations se terminent en seulement deux cycles d'horloge. Dans un CPU, même cet exemple de simple calcul prend beaucoup plus de cycles d'horloge.

Gestion des optimisations et des interfaces dans Vivado HLS :

A partir d'un code C, plusieurs implémentations matérielles sont possibles, avec différentes performances et ressources utilisées. Pour la gestion de l'implémentation selon le besoin, des directives sont proposées pour la gestion des interfaces, les ressources utilisées, les optimisations, etc.

2.7.1 Gestion des interfaces :

Les communications entre le processeur et le FPGA se font via des bus AXI (Advanced eXtensible Interface) [46].

Il existe trois types d'interface AXI :

— AXI4 : pour les besoins de cartes mémoire à haute performance,

— AXI4-Lite : pour une communication simple, à faible débit, mappée en mémoire,

— AXI4-Stream : pour les données de diffusion à grande vitesse.

La communication se fait à travers 3 types de port :

— Port GP : interface esclave 2 x 32 bits ou interface maître 2 x 32 bits. Les ports GP sont conçus pour une flexibilité maximale, permettant l'accès au registre de la partie logicielle vers la partie matérielle et inversement,

— Port HP : interface esclave 4 x 64 bits. Les ports HP sont conçus pour un accès maximal de bande passante à la mémoire externe,

— Port ACP : interface esclave A x 64 bits. Les ports ACP permettent à un accélérateur matériel d'accéder à faible latence au cache des processeurs.

Vivado HLS prend en charge l'ajout d'interfaces AXI suivants :

— AXI4-Stream (axis)

— AXI4-Lite esclave (s axilite)

— AXI4 master (m axi)

L'exemple suivant montre les directives à rajouter pour l'utilisation d'un AXI4- Stream de deux tableaux A et B sur Vivado HLS.

```
void ExempleAXI(int A[50], int B[50]) {
#pragma HLS INTERFACE axis port=A
#pragma HLS INTERFACE axis port=B
int i;
for(i = 0; i < 50; i++){
B[i] = A[i] + 5;}
}
```

2.7.2 Gestion des optimisations :

Une liste des directives principales d'optimisation fournies par Vivado HLS est donnée dans le Tableau 2.4 [47].

Directive	Description
ALLOCATION	limiter le nombre d'opérations, de noyaux ou de fonctions. Cela force le partage des ressources mais ¸ca augmente la latence.
ARRAY MAP	Regrouper des petits tableaux dans un seul grand tableau. Cela aide à réduire l'utilisation des BRAM.
ARRAY PARTITION	Diviser un grand tableau en plusieurs petits tableaux. Cela améliore l'accès aux données.
ARRAY RESHAPE	Reformer un tableau de plus grande largeur. Cela améliore l'accès aux BRAM sans utiliser plusieurs.
DATA PACK	Former un seul scalaire avec un mot de plus grande largeur.
DATAFLOW	Permet le pipeline des tâches, pour s'exécuter simultanément. Cela permet de minimiser l'intervalle.
DEPENDENCE	Surmonter les dépendances et permettre aux boucles d'être pipelinées.
EXPRESSION BALANCE	Désactiver l'équilibrage automatique des expressions.
FUNCTION INSTANTIATE	Optimiser localement différentes instances de la même fonction.
INLINE	Supprimant toute la hiérarchie des fonctions. Cela permet l'optimisation logique et améliore la latence.
LATENCY	Spécifier une contrainte de latence minimale et maximale.
LOOP FLATTEN	Affaisser les boucles imbriquées en une seule boucle.
LOOP MERGE	Fusionner les boucles consécutives. Cela pour réduire la latence globale et améliorer l'optimisation logique.
PIPELINE	Réduire l'intervalle d'initiation en autorisant l'exécution simultanée des opérations dans une boucle ou une fonction.
STREAM	Spécifier qu'un tableau doit être implémenté comme FIFO ou RAM lors de l'optimisation dataflow.
UNROLL	Créer pour les boucles plusieurs opérations indépendantes plutôt qu'une seule collection d'opérations.

Tableau 2.4: Directives d'optimisation Vivado HLS

En plus des directives d'optimisation, Vivado HLS fournit des paramètres de configuration pour modifier le comportement par défaut de la synthèse (voir Le Tableau 2.5 [47]).

Directive	Description
Config Array Partition	Limiter le nombre d'opérations, de noyaux ou de fonctions. Déterminer comment les tableaux sont partitionnés.
Config Bind	Déterminer le niveau d'effort à utiliser pendant la phase d'assignation.
Config Compile	Diviser un grand tableau en plusieurs petits tableaux
Config Dataflow	Spécifier le canal par défaut et la profondeur FIFO dans l'optimisation dataflow.
Config Interface	Contrôler les ports d'I/O non associés aux arguments de la fonction.
Config RTL	Fournir un contrôle sur la sortie RTL
Config Schedule	Déterminer le niveau d'effort à utiliser pendant la phase d'ordonnancement.

Tableau 2.5: Configurations Vivado HLS

2.8 Avantages et problèmes du HLS :

2.8.1 Avantages du HLS :

Avec une synthèse de haut niveau, l'algorithme peut être facilement transféré du logiciel à une implémentation matérielle. Actuellement, beaucoup de compilateurs de génération peuvent travailler avec le code C standard, compiler de C à RTL, et s'appuyer sur les outils spécifiques au fournisseur de FPGA pour faire la synthèse sur un FPGA. Cependant, il suffit rarement de compiler du code logiciel pour FPGA. L'algorithme du logiciel a souvent été optimisé pour l'implémentation sur un processeur à base de CPU. Il est généralement nécessaire de restructurer l'algorithme pour qu'il corresponde mieux au matériel. Cependant, au lieu d'effectuer cette étape manuellement dans la traduction de C à RTL, cette restructuration peut être effectuée directement avec HLS sur la source de haut niveau, ce qui est plus facile et moins sujette aux erreurs. L'un des principaux avantages de travailler directement dans un langage de haut niveau est que le chemin de contrôle est souvent implicite dans la représentation du langage. Les outils HLS analysent la structure de l'algorithme (boucles, branches, etc.) pour extraire et construire automatiquement le chemin de contrôle. Par contre, l'implémentation manuelle dans RTL nécessite de coder explicitement le chemin de contrôle ainsi que le chemin de données. Pour les algorithmes complexes, la conception du chemin de contrôle peut nécessiter autant d'efforts que la conception du chemin de données.

Pour chacune des conceptions générées, la simulation ou la vérification est nettement plus rapide au sein des outils de haut niveau. Ceci est dû à la vérification à lieu à un niveau supérieur [48] .

Dans les environnements de coprocesseur, où le calcul est partagé entre un FPGA et un CPU, HLS permet de compiler le même code source en tant que matériel ou logiciel (par exemple [49], [50]). Cela permet d'échanger rapidement différents modules entre l'implémentation matérielle et logicielle, ce qui facilite la recherche de l'optimal. Pour un code

structuré de manière appropriée, les outils HLS modernes peuvent générer des conceptions aussi efficaces que la RTL codée à la main en termes de ressources et de vitesse de traitement ([51],[52], [53]).

Les outils de synthèse de haut niveau (HLS) peuvent fournir des avantages significatifs pour l'implémentation des algorithmes de tri sur les FPGAs. La représentation de niveau supérieur (généralement basée sur C) permet d'exprimer plus facilement les algorithmes, réduisant ainsi considérablement les temps de développement. Le niveau supérieur facilite également l'exploration de l'espace de conception, ce qui facilite l'optimisation du compromis entre les ressources et la vitesse de traitement.

2.8.2 Les problèmes du HLS :

Avec la plupart des outils HLS, l'algorithme doit être implémenté avec une façon particulière pour permettre aux outils de synthèse d'identifier et d'exploiter le parallélisme au niveau architectural. Cela nécessite une restructuration du code. Même avec une telle restructuration du code, le matériel résultant ne peut pas être bien optimiser et souffre toujours des mauvais performance. La programmation matérielle bas niveau et haut niveau peut souvent mener à une utilisation inefficace du matériel. De plus, les algorithmes basés fortement sur les pointeurs et l'arithmétique du pointeur ne se synthétisent pas bien avec le matériel [53] . Dans le logiciel, un pointeur est une adresse d'une variable dans une mémoire. Dans ce cas, il y a deux problèmes majeurs dans le contexte d'une implémentation FPGA. Premièrement, la mémoire sur puce est distribuée, avec de nombreux blocs indépendants et relativement petits qui possèdent chacun son propre espace d'adressage.

Deuxièmement, dans le matériel plusieurs variables sont stockées dans des registres. Un autre problème pour HLS est la récursivité. Elle est une autre technique logicielle qui ne se génère pas bien au matériel. Les fonctions ou procédures du logiciel stockent les variables locales sur la pile, de sorte que chaque appel dispose d'un espace séparé pour le stockage de la variable. Sur un FPGA, une fonction est implémentée comme un bloc de matériel, avec un appel de procédure multiplexant ce matériel pour chaque invocation.

Par contre, il n'y a pas la notion de pile dans HLS. Les variables locales sont généralement stockées dans des registres partagés avec chaque appel de la procédure. La réutilisation de registres par des appels successifs limite l'utilisation des fonctions récursives à la récursion de queue. En général, les algorithmes récursifs doivent être restructurés pour utiliser une itération équivalente.

En effet, la conception du système est difficile en raison de la nécessité de concevoir la synchronisation. La conception de logiciel mono-thread n'est pas un problème, car tout est

implicitement séquentiel. Par contre, avec plusieurs threads, il est nécessaire d'assurer la synchronisation aux points clés. Donc, il y a beaucoup plus de tâches à synchroniser.

Alors que HLS fait raisonnablement bien la programmation et le pipelining des séquences d'opérations, et la synchronisation de la distribution du calcul sur plusieurs processeurs parallèles, il se débat avec une synchronisation plus complexe. Le code RTL généré par les outils HLS n'est pas particulièrement lisible par un humain, ce qui le rend très difficile à modifier au niveau hardware. Globalement, le but de l'utilisation de HLS est d'éviter la programmation RTL. Cependant, il est toujours nécessaire de vérifier la sortie RTL ([48],

[54]) . Si la vérification RTL indique une erreur, il est donc très difficile de préciser ou se trouve le problème dans HLS. En logiciel, toutes les structures de données sont stockées dans une seule mémoire monolithique.

2.9 Défis et limitations des outils HLS :

Une excellente étude des outils HLS est présentée par Windh et al (2015) [55] , les auteurs ont présenté une évaluation de cinq outils HLS : Xilinx *Vivado HLS* [*56*] , *Intel* (Altera) *FPGA SDK for OpenCL* (Intel, 2013) [57] , *Bluespec System Verilog* (BSV) (Arvind, 2009) [58] , *ROCCC* (Najjar, Villarreal, & Halstead, 2016) [59] et LegUp (Canis et al., 2013) [26] en fournissant une vue d'ensemble de leurs flots de conception, les optimisations qu'ils offrent et une analyse qualitative de leurs implémentations matérielles. L'évaluation de l'outil de Intel « *Intel FPGA SDK for OpenCL »* montre que les implémentations générées par cet outil sous forme de fichiers Verilog sont non lisibles par les développeurs. De plus, il n'existe pas de méthodes qui permettent de faire un lien entre les simulations au niveau RTL et la simulation logicielle à haut niveau. Par exemple, un signal ou un module à bas de niveau (RTL) n'a aucune correspondance à haut niveau. Une fonction à haut niveau peut être implémentée par différents modules au niveau RTL non connus et non identifiés pour l'utilisateur. De plus, l'utilisation des outils d'instrumentation ou de débogage tel ChipScope (Xilinx, 2012b) [60] n'offre aucune compréhension à la fonctionnalité d'application lorsque cette dernière est décrite avec un langage de haut niveau. En effet, l'observation des signaux au niveau RTL est inutile puisque le développeur n'a aucune idée des modules RTL générés par les outils de synthèse de haut niveau. Cette obscurité à bas niveau complexifie l'analyse des implémentations générées. Ces limitations empêchent les développeurs d'investiguer les goulots d'étranglement dans les applications déployées.

Dans l'article "*A Survey and Evaluation of FPGA High-Level Synthesis Tools*" (Nane et al., 2016) [61] , les développeurs de trois outils HLS académiques: « Delft workbench automated

reconfigurable VHDL generator » (DWARV) (Nane et al., 2012) [62] , « BAMBU » (Pilato & Ferrand, 2013) [63] et « LegUp » (Canis et al., 2013) [26] ont présenté une méthodologie permettant une évaluation approfondie de performance de ces différents outils en utilisant un ensemble de bancs d'essai (benchmarks). Une comparaison avec un outil commercial est aussi présentée.

Cette évaluation est basée sur trois métriques de performance : la fréquence maximale (Fmax en MHz), la latence (le nombre de cycles d'horloge nécessaires pour une exécution de l'application) et le temps de *wall-clock* (période d'horloge minimale × latence). Les résultats de cette étude montrent que chaque outil HLS évalué peut améliorer considérablement les performances grâce à des optimisations spécifiques à chaque application ou domaine d'applications. Cependant, les développeurs logiciels doivent tenir compte que ces optimisations, ciblant les architectures matérielles reconfigurables, sont considérablement différentes à celles dédiées au logiciel (par exple, la réorganisation des données du cache).

De plus, les résultats ont montré que les outils de HLS académiques et commerciaux ne sont pas radicalement différents en termes de qualité de résultats. Cependant, les outils commerciaux offrent plus de fonctionnalités que les outils académiques.

Une revue approfondie de la littérature qui a couvert 46 articles et 118 applications associées est faite par Lahti et al. (2019) [64] pour investiguer la différence de la qualité des résultats (quality of results; QoR) et de la productivité entre les deux flots de conception HLS et RTL. Les résultats montrent qu'en moyenne, le QoR du flot RTL est meilleur que celui des outils de HLS. Cependant, le temps de développement moyen avec les outils RTL est trois fois plus grand que celui du HLS, et un développeur obtient une productivité entre quatre et six fois plus élevée avec l'approche de conception HLS. En outre, pour aider à combler l'écart de qualité de service, cette thèse explore dans son troisième chapitre les optimisations des applications basées sur l'approche HLS et plus particulièrement le flot de conception OpenCL pour FPGA.

Tous les auteurs des travaux examinés et présentés dans cette sous-section affirment que les outils HLS ont fait des progrès significatifs permettant la configuration des FPGA avec comme entrée une spécification de l'application décrite avec des langages de programmation de haut niveau. Cependant, des modifications manuelles visant l'adaptation vers les architectures matérielles utilisées sont nécessaires pour l'optimisation ou la compilation sans erreurs. De plus, les performances sont souvent décevantes et inférieures à celles des CPU à cause des mauvaises optimisations adoptées. Bien que les outils HLS ont amené le développement avec les FPGA à un niveau d'abstraction comparable à celui des CPU pour un concepteur logiciel, leurs limitations en termes d'optimisation automatique, d'instrumentation et de débogage diminuent la productivité et présentent toujours un défi majeur face à

l'unification des méthodes de conception présentes dans les plateformes hétérogènes reconfigurables Figure (2.14).

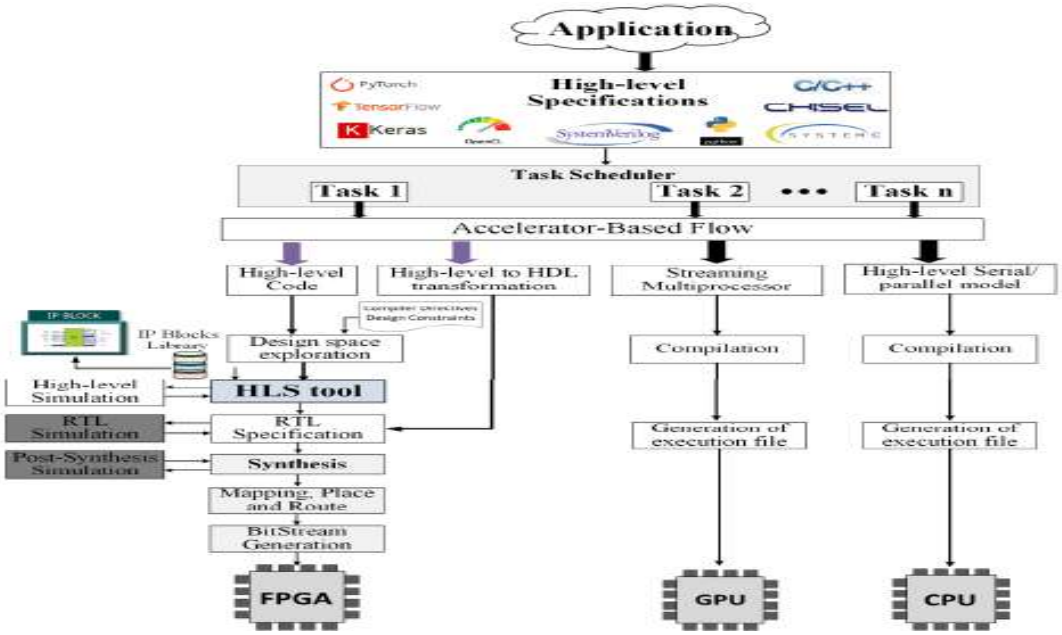


Figure 2.14 : Flot de conception des applications sur des plateformes hétérogènes Reconfigurables .

2.10 Conclusion :

Dans ce chapitre nous avons présenté une étude descriptive après une Historique d'évolution de la HLS les différents éléments de base sous HLS, leurs avantages et les problèmes qui trouves , en fin nous montrons les Défis et limitations des outils HLS .

CHAPITRE 03
SIMULATION ET IMPLEMENTATION

3.1 Introduction

Ce chapitre présente notre implémentation pour la réalisation d'un filtre a coefficients rechargeables, pour cela, les étapes de conception pour générer un adaptateur IP-XACT à partir d'une conception utilisant Vivado HLS et l'adaptateur IP-XACT est données ,pour à la fin générée un système de processeur utilisant IP Integrator dans Vivado, tous cela pour l'élimination du bruit sur le signal bruit.

Note que pour la validation de notre travail on a utilisé une base des données qui contient les signaux de défaut et en appliquant notre filtre à ces données, à la fin en compare les résultats par rapport ou résultats donne par simulation Matlab

La base des données utilisées

Pour étudier, analyser et diagnostiquer les défauts de roulement, nous avons choisi de travailler sur les donneés des experiences de « Machinery Failure Prevention Technology (MFPT) Challenge Data »

Les données du MFPT [65] contiennent 23 ensembles de données collectées à partir de machines dans diverses conditions de panne. Les 20 premiers ensembles de données sont collectés à partir d'un banc d'essai de roulements, avec 3 dans de bonnes conditions, 3 avec des défauts de bague extérieure sous charge constante, 7 avec des défauts de bague extérieure sous diverses charges et 7 avec des défauts de bague intérieure. Dans notre travail, seules les données collectées à partir du banc d'essai avec des conditions connues sont utilisées.

3.2 Implémentation sur HLS

Pour l'implémentation sur HLS, le but de notre travail est de concevoir un filtre FIR simple avec des coefficients rechargeables (filtre à coefficients non statiques) en utilisant Vivado HLS. Comme point de départ, on a conçus un filtre FIR à coefficient statique basé sur l'exemple de conception HLS de
https://islab.soe.uoguelph.ca/sareibi/TEACHING_dr/ENG6530_RCS_html_dr/outline_W2017/docs/ASSIGN_dr/ASSIGN7_dr/L4-FIR

La conception passe par les étapes suivantes :

- C Simulation,
- C Synthesis et
- C/RTL Co-Simulation.

Ensuite, on a une deuxième version de filtre FIR avec des coefficients rechargeables. Cette deuxième version de fonctionnement recharge les valeurs des coefficients avec les données d'entrée. Les coefficients sont stockés dans une variable statique implémentée comme un noyau RAM_1P.

Pour l'implémentation de la solution sur HLS, on a suivi les étapes suivantes

3.2.1 Création du projet

Lancez le logiciel Vivado HLS et Créez un nouveau projet comme indique dans la figures suivantes, Fournir un nom et un chemin pour ce nouveau projet

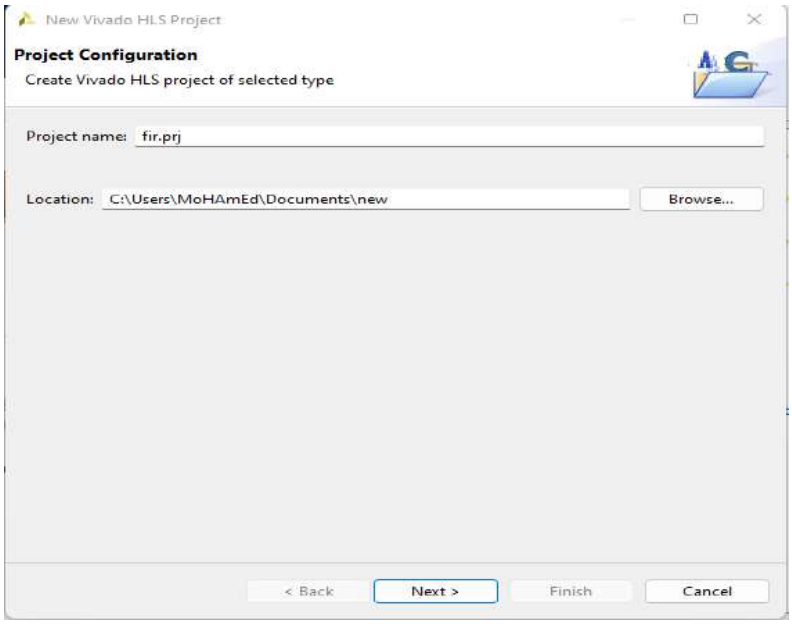


Figure 3. 1 Projet configuration.

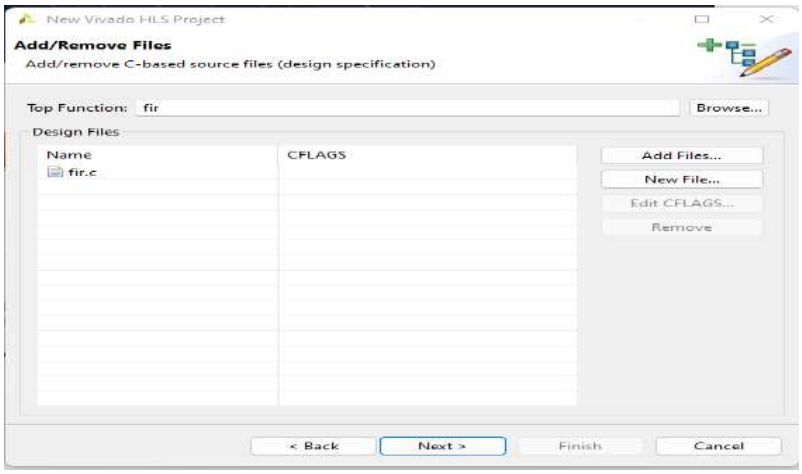


Figure 3. 2 Fichiers sources du projet

La fenêtre suivante de l'assistant de projet vous permet d'ajouter au projet les fichiers associés au banc de test au projet

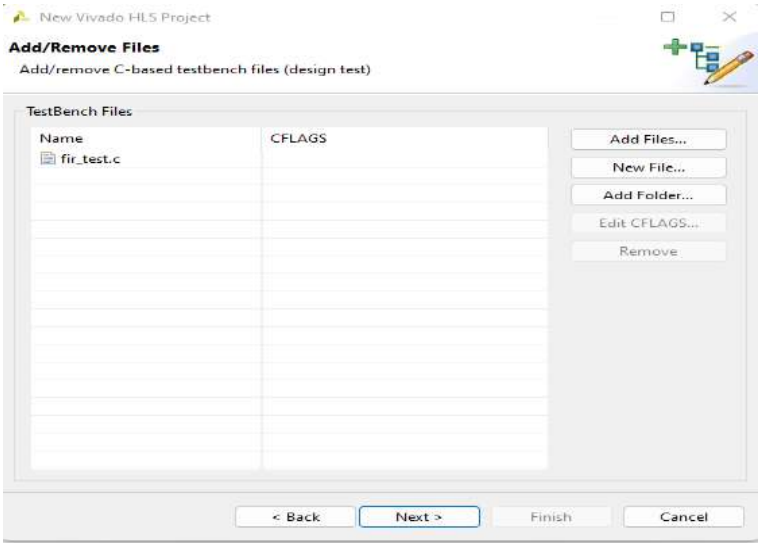


Figure 3. 3 Fichiers de banc d'essai de projet.

Dans la plupart des exemples de conception fournis avec Vivado HLS, le banc de test se trouve dans un fichier distinct de la conception. Le fait d'avoir le banc de test et la fonction à synthétiser dans des fichiers séparés permet de maintenir une séparation nette entre le processus de simulation et de synthèse.

Fournir le type de FPGA pour la plateforme Zybo Z7

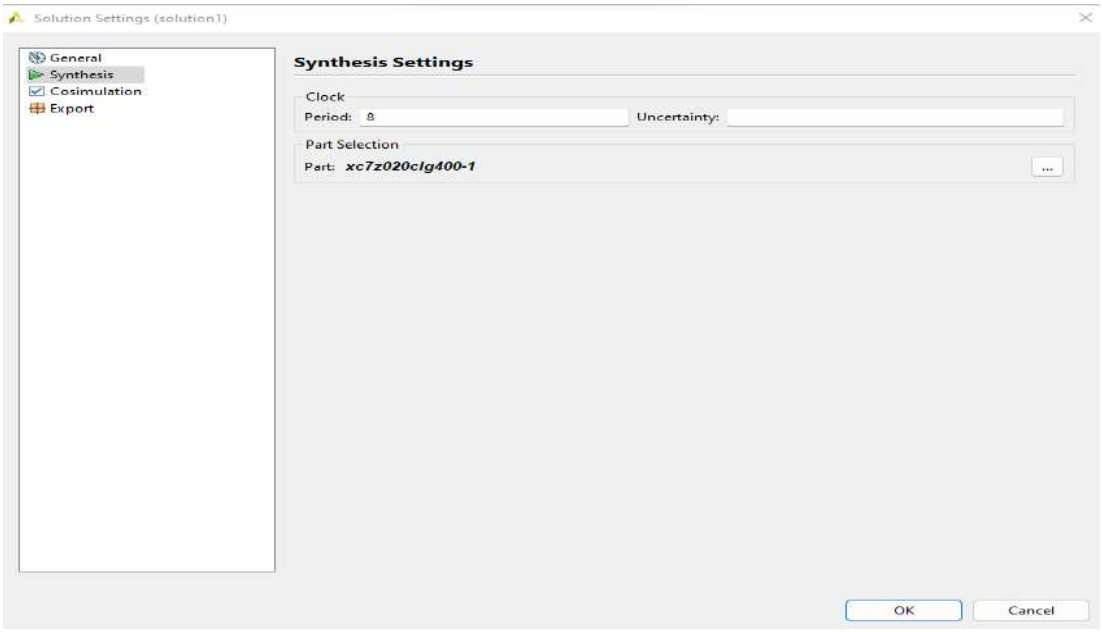


Figure 3. 4 soulution configuration.

La configuration du fichier fir.c est données comme suivants

```
#include "fir.h"

void fir (
  data_t *y,
  data_t x,
  coef_t coe[N+1]
 ) {


  static data_t shift_reg[N];
  acc_t acc;
  int i;

  acc =(acc_t)shift_reg[N-1]*(acc_t)coe[N];
  loop: for (i=N-1;i!=0;i--) {
    acc+=(acc_t)shift_reg[i-1]*(acc_t)coe[i];
    shift_reg[i]=shift_reg[i-1];
  }
  acc+=(acc_t)x*(acc_t)coe[0];
  shift_reg[0]=x;
  *y = acc>>15;
}

```

Figure 3. 5 program fir.c.

Le filtre FIR utilise le input x comme entrée d'échantillon et un pointeur vers la sortie d'échantillon calculée. Ces deux éléments sont définis par le type de données data_t. Les coefficients sont programme aussi comme entrée nommée coef_t . L'algorithme séquentiel est appliqué et la valeur accumulée (sortie d'échantillon) est calculée dans la variable acc de type acc_t.

Le fichier fir.h est le fichier d'en-tête ; comprend ap_cint.h afin de pouvoir utiliser une largeur de données définie par l'utilisateur (d'une précision arbitraire). Il définit également le nombre de coefficients (N), le nombre d'échantillons à générer (dans le testbench), et les types de données coef_t, data_t, et acc_t. Le coef_t et le data_t sont courts (16 bits). Puisque l'algorithme itère (multiplie et accumule) sur 201 taps, il y a une possibilité de croissance de 6 bits et donc acc_t est défini comme int38. Puisque acc_t est plus grand que l'échantillon et largeur du coefficient

fir.h

```
#ifndef _FIR_H_
#define _FIR_H_
#include "ap_cint.h"
#define N   201
#define SAMPLES N+10 // just few more samples then number of taps
typedef short   coef_t;
typedef short   data_t;
typedef int38   acc_t;
#endif
```

Figure 3. 6 programme fir.h.

Maintenant en s'intéresse au fichier fir_test.c sous testbench comme ci-dessous

```
#include <stdio.h>
#include <math.h>
#include "fir.h"
void fir (
  data_t *y,
  data_t x,
  coef_t coe[N+1]
);

int main () {
    FILE *fp;
    float val1;
  coef_t taps0;
  data_t signal, output;
//for(int i=0;i<N;i++){
    //taps0[i]=taps[i]
//}
  fp=fopen("fir_impulse.dat","w");
  int i;
  for (i=0;i<SAMPLES;i++) {
      if(i==0)
          signal = 0x8000;
      else
          signal = 0;
      fir(&output,signal,&taps0);
      printf("%i %d %d\n",i,(int)signal,(int)output);
//        fprintf(fp,"%i %d %d\n",i,signal,output);
  }
  fclose(fp);
```

Figure 3. 7 program fir_test.c

interface de synthèse

Choisir un protocole pour notre interface de synthèse comme dans la figure suivante :

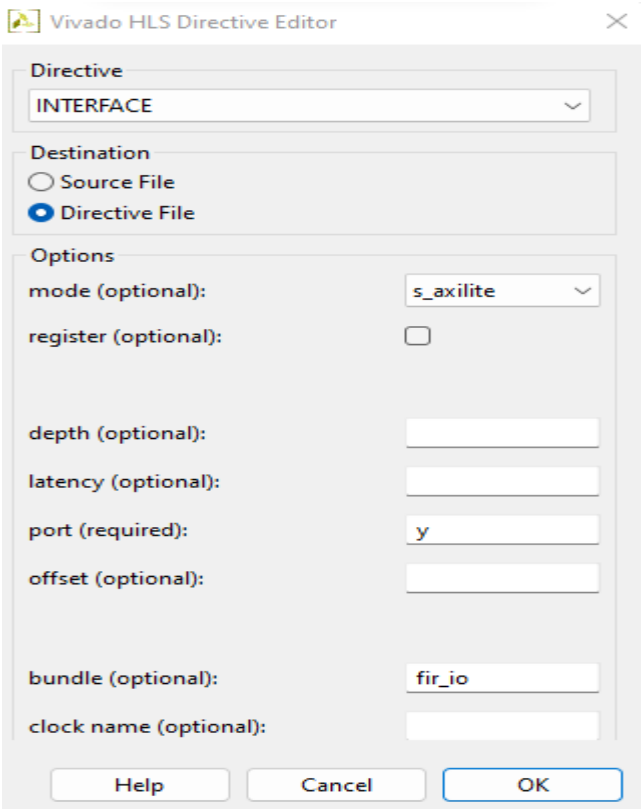


Figure 3. 8 une interface de synthèse pour le port y.

De la même façon créez une interface de synthèse pour les ports x et c

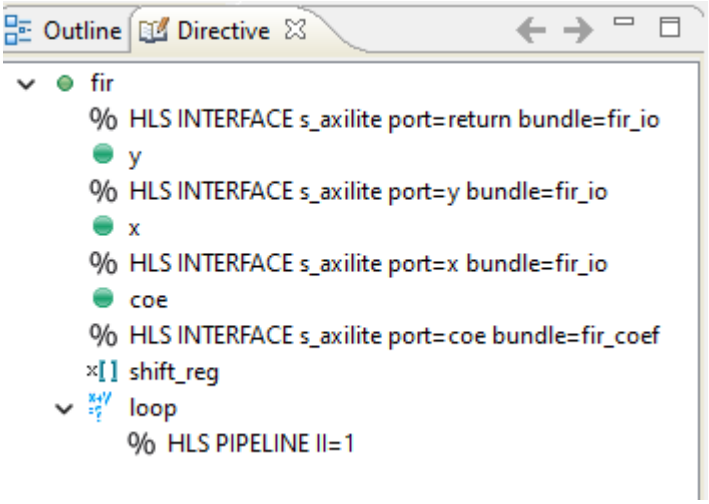


Figure 3. 9 Sélection de l'adaptateur AXI4LiteS et de l'ensemble de dénomination.

3.2.1 Exécutez la simulation

Pour l'exécution de la simulation on suit les étapes suivantes :

- La vérification dans le flux HLS de Vivado peut être séparée en deux processus distincts.
- La validation pré-synthèse qui valide que le programme C implémente correctement la fonctionnalité requise.
- La vérification post-synthèse qui vérifie que le RTL est correct.
- Les deux processus sont appelés simulation : la simulation C et la co-simulation C/RTL.
- Avant la synthèse, la fonction à synthétiser doit être validée avec un banc d'essai utilisant la simulation C. Un banc d'essai C comprend une fonction de haut niveau main() et la fonction à synthétiser. Il peut inclure d'autres fonctions. Un banc d'essai idéal possède les attributs suivants :
- Le banc de test s'autocontrôle et vérifie que les résultats de la fonction à synthétiser sont corrects.

Si les résultats sont corrects, le banc d'essai renvoie une valeur de 0 à main(). Dans le cas contraire, le banc d'essai doit renvoyer toute valeur non nulle.

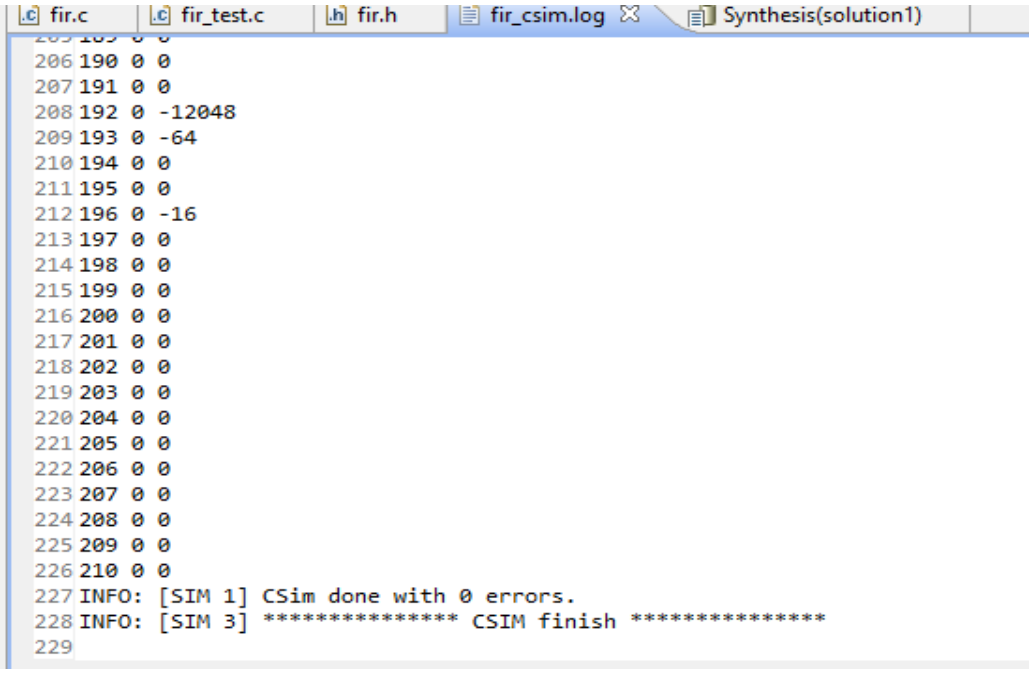


Figure 3. 10 Partie initiale de la sortie générée dans la vue Console.

3.2.2 Synthèse en code VHDL de notre fichier source en C

Le rapport de la synthèse nous est donné dans la fenêtre ci-dessous :On voit apparaitre les différentes interfaces de synthèse

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
s_axi_fir_io_AWVALID	in	1	s_axi	fir_io	array
s_axi_fir_io_AWREADY	out	1	s_axi	fir_io	array
s_axi_fir_io_AWADDR	in	10	s_axi	fir_io	array
s_axi_fir_io_WVALID	in	1	s_axi	fir_io	array
s_axi_fir_io_WREADY	out	1	s_axi	fir_io	array
s_axi_fir_io_WDATA	in	32	s_axi	fir_io	array
s_axi_fir_io_WSTRB	in	4	s_axi	fir_io	array
s_axi_fir_io_ARVALID	in	1	s_axi	fir_io	array
s_axi_fir_io_ARREADY	out	1	s_axi	fir_io	array
s_axi_fir_io_ARADDR	in	10	s_axi	fir_io	array
s_axi_fir_io_RVALID	out	1	s_axi	fir_io	array
s_axi_fir_io_RREADY	in	1	s_axi	fir_io	array
s_axi_fir_io_RDATA	out	32	s_axi	fir_io	array
s_axi_fir_io_RRESP	out	2	s_axi	fir_io	array
s_axi_fir_io_BVALID	out	1	s_axi	fir_io	array
s_axi_fir_io_BREADY	in	1	s_axi	fir_io	array
s_axi_fir_io_BRESP	out	2	s_axi	fir_io	array
ap_clk	in	1	ap_ctrl_hs	fir	return value
ap_rst_n	in	1	ap_ctrl_hs	fir	return value
interrupt	out	1	ap_ctrl_hs	fir	return value

Export the report(.html) using the Export Wizard

Open Analysis Perspective Analysis Perspective

Figure 3. 11 Signaux d'interface générés.

Le code Verilog synthétisé se trouve dans le fichier fir.v

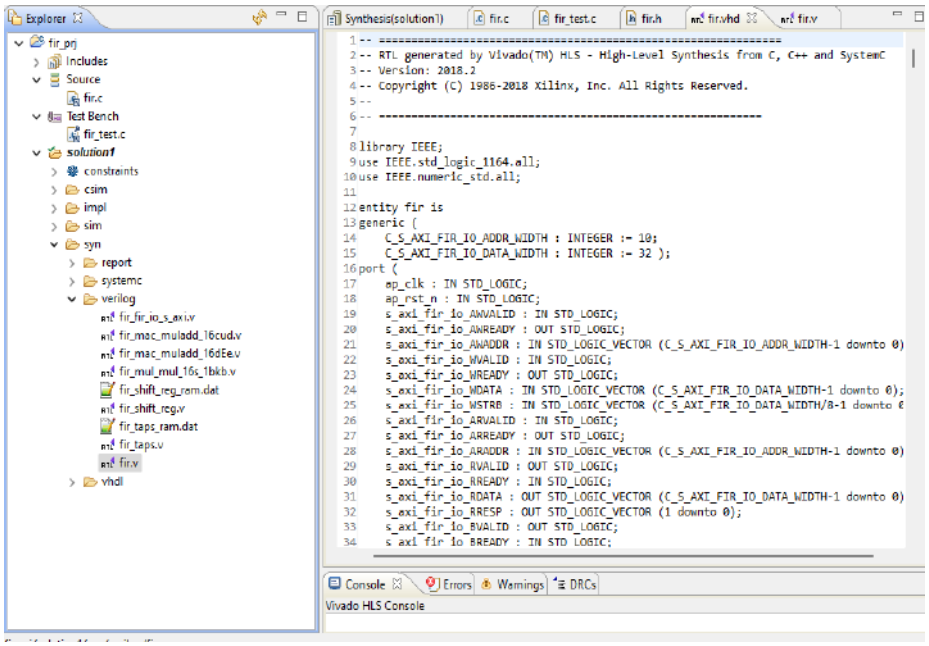


Figure 3. 12 Le code Verilog synthétis.

3.2.3 Simulation de la fonction synthétisée

Un répertoire sim est créé dans le dossier solution lorsque la vérification RTL est terminée. La figure suivante montre les sous-dossiers créés.



Figure 3. 13 RTL/C CoSimulation.

- Les dossiers de rapport contiennent le rapport et le fichier journal pour chaque type de RTL simulé.

- Un dossier de vérification est créé pour chaque type de RTL qui est vérifié. Le dossier de vérification est nommé verilog ou vhdl. Si un format RTL n'est pas vérifié, aucun dossier n'est créé.

- Les fichiers RTL utilisés pour la simulation sont stockés dans le dossier de vérification. La simulation RTL est exécutée dans le dossier de vérification.

3.2.4 Exportation du fichier pour la création d'un IPBLOCK

Pour l'exportation du fichier de la création d'un IPBLOCK on Cliquez sur exportRTL

Les sélections disponibles dans le menu déroulant Format Selection dépendent du dispositif FPGA ciblé pour la synthèse.

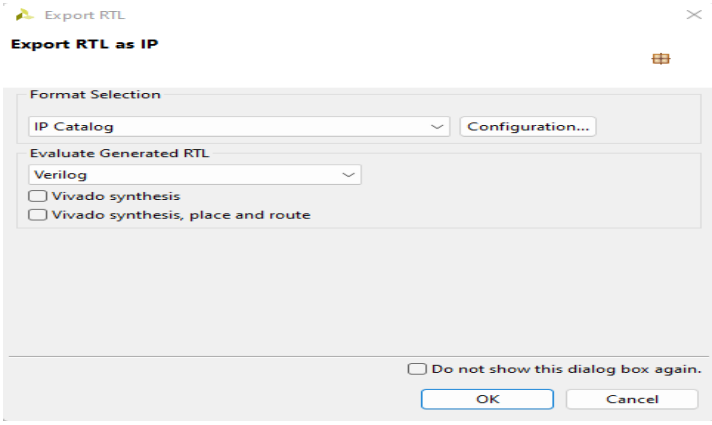


Figure 3. 14 Boîte de dialogue Exporter RTL .

Développez le répertoire ip et observez plusieurs fichiers et sous-répertoires. L'un des sous-répertoires d'intérêt est le répertoire drivers qui se compose de fichiers header, c, tcl, mdd et makefile. Un autre fichier intéressant est le fichier zip, qui est le fichier de référentiel ip qui peut être importé dans une conception IP Integrator.

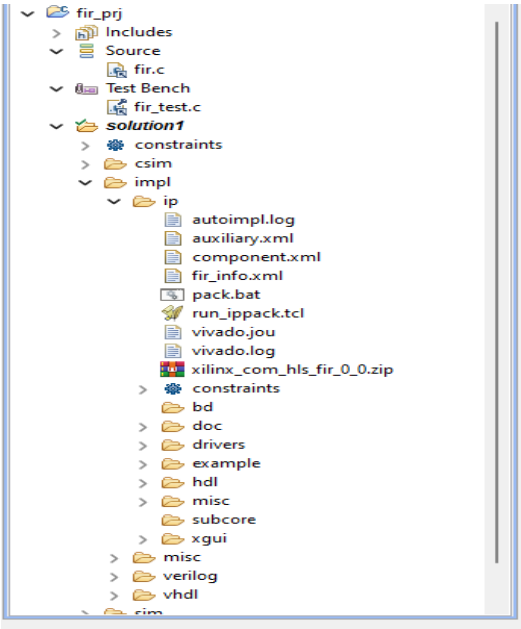


Figure 3. 15 Adaptateur IP-XACT généré.

Le dossier syn contient quatre sous-dossiers. Un dossier de rapport et un dossier pour chacun des formats de sortie RTL.

Le dossier report contient un fichier de rapport pour la fonction de niveau supérieur et un pour chaque sous-fonction de la conception, à condition que la fonction n'ait pas été inlined à l'aide de la directive INLINE ou inlined automatiquement par Vivado HLS. Le rapport pour la fonction de niveau supérieur fournit des détails sur l'ensemble de la conception.

Les dossiers verilog, vhdl et systemc contiennent les fichiers RTL de sortie. La figure précédente montre le dossier verilog développé. Le fichier de premier niveau porte le même nom que la fonction de premier niveau pour la synthèse. Dans la conception C, il y a un fichier RTL pour chaque fonction (non inlined). Il peut y avoir des fichiers RTL supplémentaires pour implémenter les sous-blocs (bloc RAM, multiplicateurs pipelinés, etc.).

La conception passe par C Simulation et C Synthesis et C/RTL Co-Simulation. Aucune erreur, avertissement ou blocage de simulation n'est signalé.

3.3 Créer un projet Vivado

La figure 3.14 represente les modules de connexion entre le filtre FIR (notre IP core crée en HLS)qui communique avec la zedboard a travers une interface AXI.

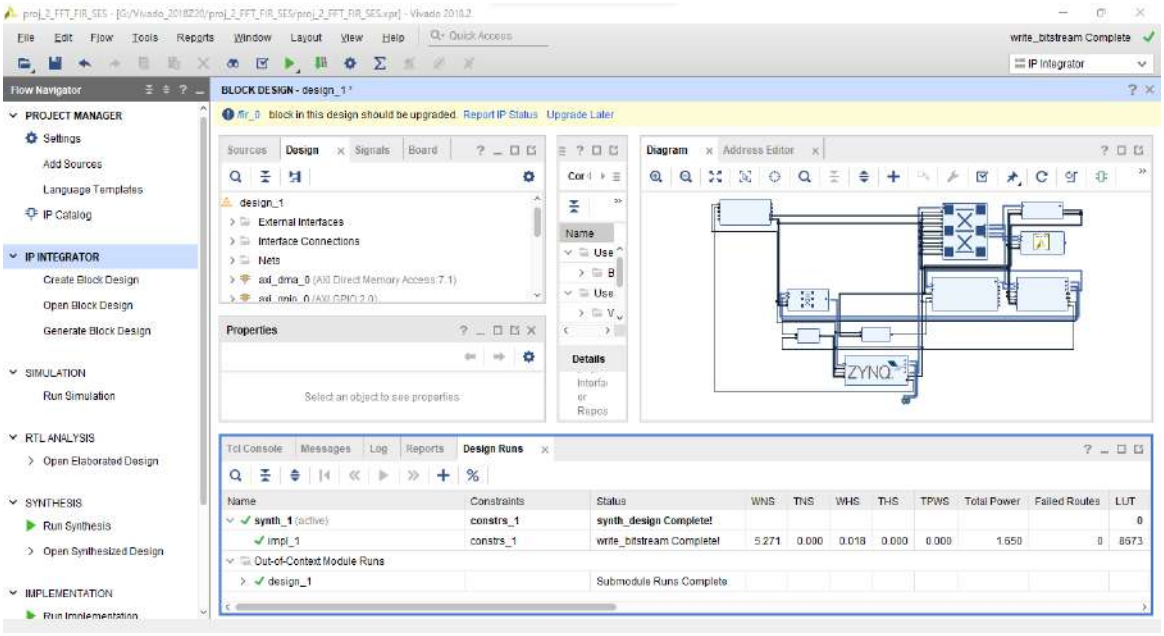


Figure 3. 16 interface ghraphique de vivado(xlinx IDE) représentant les différents composants embarqués sur la **zybo z7**

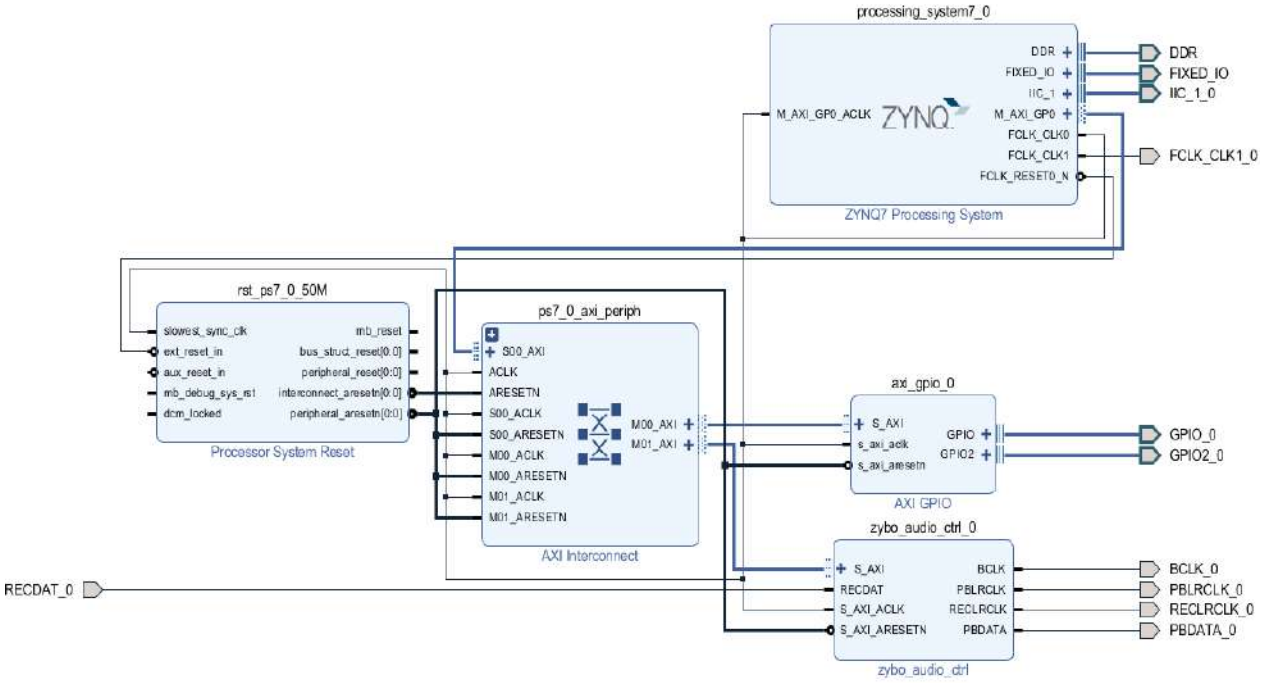


Figure 3.17 Conception de bloc avec z**ybo**_audio_ctrl et connexions faites pour Z**ybo Z7**.

3.3.1 Importation de notre IPBLOCK sous Vivado

Importez notre IPBLOCK cliquez sur projet settings et additionnez l'IP

Le répertoire sera scanné et ajouté dans la fenêtre Dépôts IP, et la première entrée IP sera affichée dans la fenêtre IP in Selected Repository.

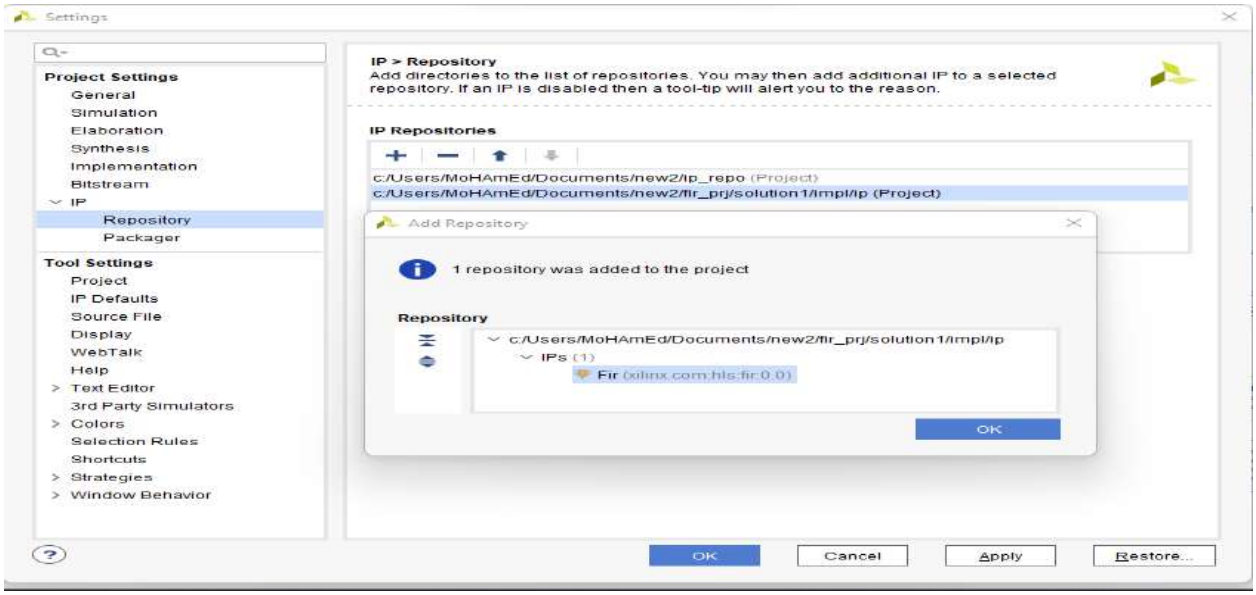


Figure 3.18 Définition du chemin d'accès aux référentiels IP .

Ouvrir le bloc design system et additionnez un IP bloc



Figure 3.19 fir référentiels IP

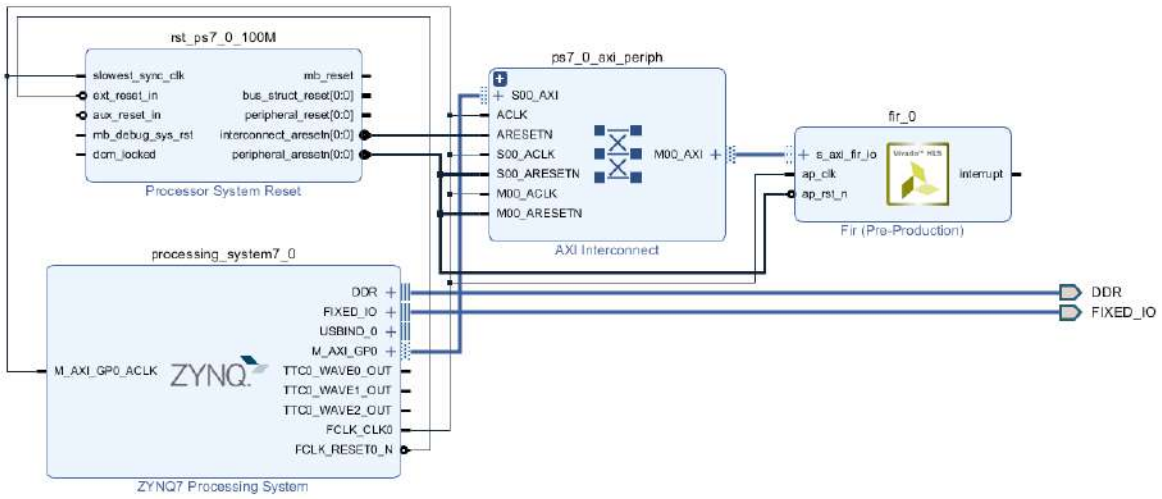


Figure 3.20 La conception matérielle complète

3.3 Implémentation sur SDK

Nous obtenons à la fin un IP-core (nommé FIR_0 dans notre exemple) prêt à être implémenté dans un projet dans le logiciel Xilinx Software Developement Kit

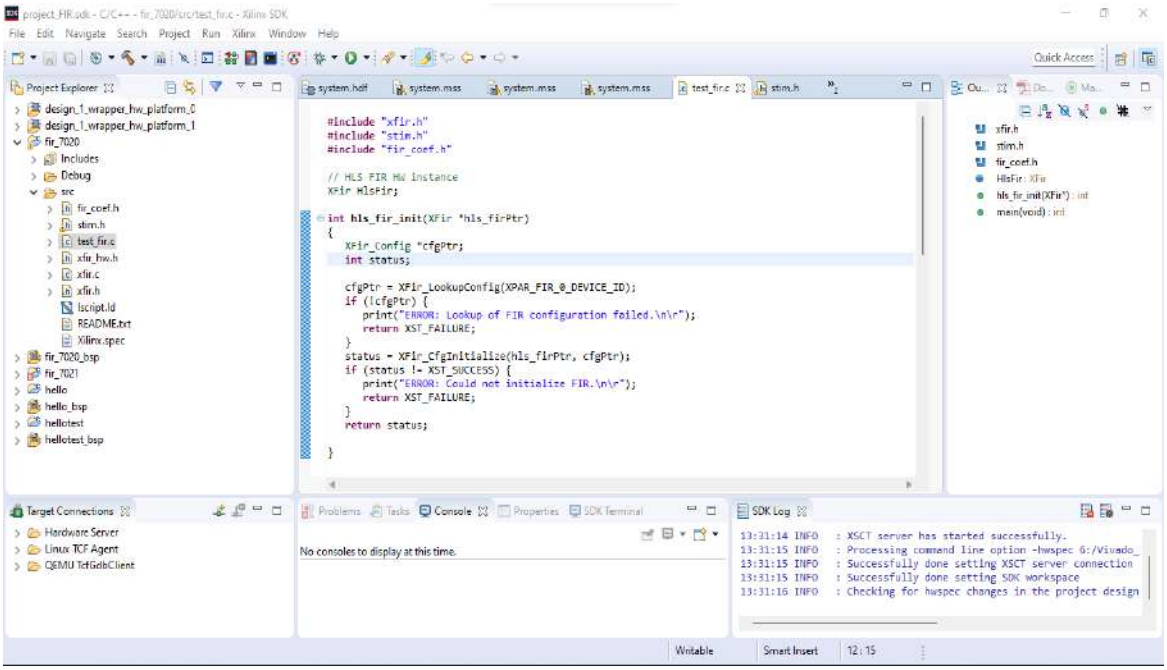


Figure 3.21 Test de l'IP-core « add » sur la carte FPGA

3.4 Résultat de la simulation

Pour la validation de notre implémentation, , nous avons choisi de travailler sur les donneés des experiences de « Machinery Failure Prevention Technology (MFPT) Challenge Data ; ou une simulation sur MATLAB est faite, après une comparaison des résultats avec l'implémentation sur carte FPGA

3.4.1 Simulation avec MATLAB et implimentation avec Vivado

Pour la simulation ,la figure suivante représente le signal brute, le signal après filtrage avec Matlab, et le signal après filtrage avec notre implémentation.

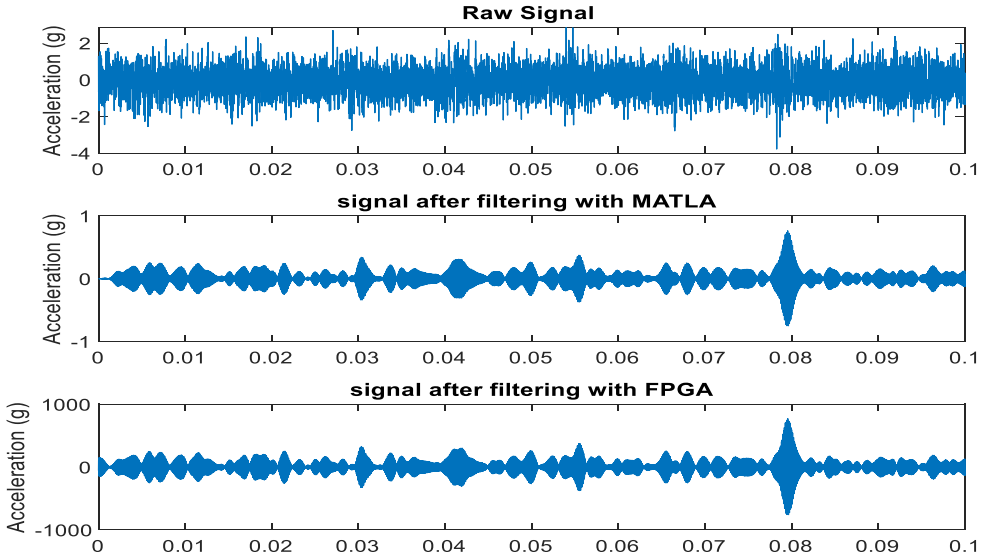


Figure 3.22 le signal avant et après filtrage

3.4.2 Comparaison entre les résultat obtenus sous MATLAB et Vivado

On peut voire dans la figure suivantes les résultats sont presque identique

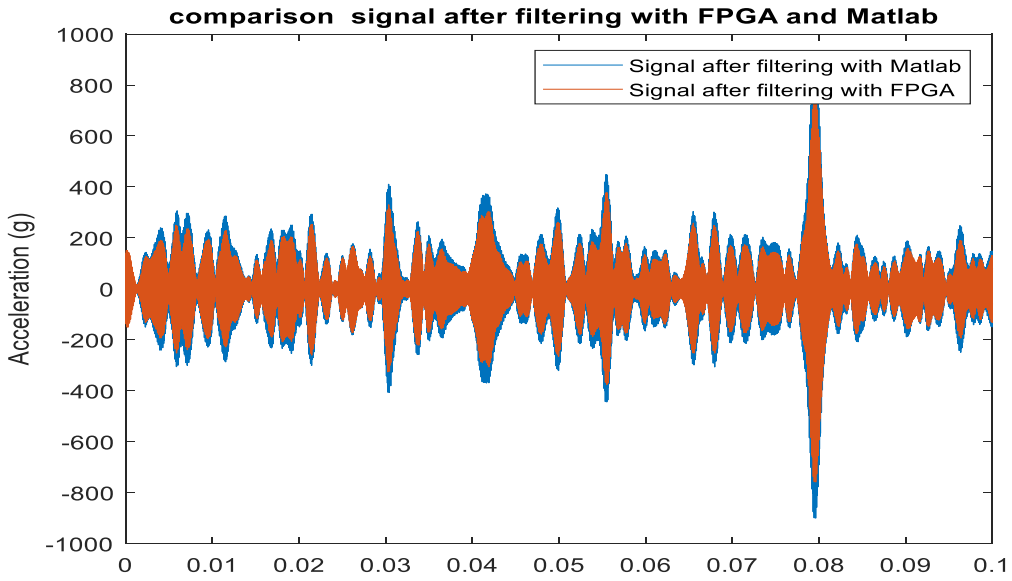


Figure 3.23 comparison signal apres filtrage avec MATLAB et FPGA

3.5 Conclusion

Dans ce chapitre, vous avez ajouté la directive RESOURCE pour créer un adaptateur IP-XACT. Vous avez généré l'adaptateur IP-XACT pendant la phase d'implémentation. Vous avez ensuite créé un système de processeur en utilisant IP Integrator, intégré l'adaptateur IP-XACT généré, et testé le système avec l'application fournie.

Conclusion générale

L'objectif de ce travail était d'implémenter sur une carte FPGA, d'un filtre à paramètre variable pour une meilleure détection des défauts de roulements à billes en utilisant un système de filtrage du signal brut.

Pour la procédure de détections des défauts des roulements à billes, l'étape consiste à filtrer le signal brut mesuré à l'aide d'un filtre à réponse impulsionnelle finie(RIF) à coefficients rechargeables. Dans ce travail nous avons utilisé les circuits FPGA comme organe d'implémentation.

Le travail a été fait dans logiciel Vivado HLS pour transforme une spécification système en C en une implémentation (RTL Register Transfer Level) que tu synthétises dans un FPGA, et réaliser une conception sous Vivado HLS avec des interfaces AXI travers IP intégrateurs.

Une implémentation est faits sur une base des données connue pour la validation de nos solutions, les résultats de l'implémentation sont satisfaisants et donnent un bon filtrage de données conforme à la simulation par Matlab.

Références

[1] Djaballah Saïd, Mémoire de Magister, « Etude et optimisation de la transformée en ondelettes pour la détection des défauts dans les roulements ». Université Mohamed Chérif Messaâdia de Souk-Ahras.pdf

[2] Dechicha farid,ghrabli rafik, Mémoire de Master « Etude et implémentation d'un system pour la détection des défauts dans les roulement, université yahiya fares de medea.pdf

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

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

[5] Moncef Khitas. Mémoire de Magister « Implantation d'application de traitement de signal sur système mono-puce reconfigurable 'SoPc' » universite ferhat abbas de setif.pdf

[6] Oppenheim, Alan V., Willsky, Alan S., et Young, Ian T., 1983: Signaux et systèmes, p. 256 (Englewood Cliffs, New Jersey: Prentice-Hall, Inc.) ISBN 0-13-809731-3

[7] Rabiner, Lawrence R. et Gold, Bernard, 1975: Théorie et application du traitement numérique du signal (Englewood Cliffs, New Jersey: Prentice-Hall, Inc.) ISBN 0-13-914101-4

[8] AE Cetin, ON Gerek, Y. Yardimci, "Conception de filtre FIR Equiripple par l'algorithme FFT", IEEE Signal Processing Magazine, pp. 60–64, mars 1997.

[9] https://stringfixer.com/fr/Finite_Impulse_Response.

[10] L.R. Rabiner & B.Gold., Theory and application of digital signal processing. Prentice Hall. New Jersey. 1975

[11] Douglas F.Elliott ., Handbook of digital signal processing engeneering application. Academic press, INC. Sandiego. California 92101. 1987.a

[12] Djaballah Saïd, Mémoire de Magister, « Etude et optimisation de la transformée en ondelettes pour la détection des défauts dans les roulements ». Université Mohamed Chérif Messaâdia de Souk-Ahras.pdf

[13] Dechicha farid,ghrabli rafik, Mémoire de Master « Etude et implémentation d'un system pour la détection des défauts dans les roulement, université yahiya fares de medea.pdf

[14] LAIFAOUI Wafa. « Implémentation des filtres adaptatifs sur une carte FPGA pour l'annulation d'écho acoustique » Université Mohamed Larbi Ben M'hidi - Oum El bouaghi.pdf

[15] TARANI GAUR« Étude et conception de la transformée de Fourier rapide (FFT) et du modulateur OFDM » recherche pratique ; INSTITUT DE TECHNOLOGIE ET DE SCIENCE DE BIRLA, PILANI, CAMPUS HYDERABAD.pdf

[16] Rouibah Abdelghani ,rahmi Mahfoudh, Mémoire de Master « Etude et implémentation sur FPGA d'un système pour la détection des défauts dans les roulements à billes» Université yahiya fares de medea.pdf

[17] Trickey, H. (1987). Flamel: A high-level hardware compiler. IEEE Transactions on Computer- Aided Design of Integrated Circuits and Systems, 6(2), 259-269. doi: 10.1109/TCAD.1987.1270270

[18] Kowalski, T. J., & Thomas, D. E. (1983). The VLSI design automation assistant: Prototype system. Dans 20th Design Automation Conference Proceedings (pp. 479-483). doi: 10.1109/MDT.1985.294721

[19] Kowalski, T. J., & Thomas, D. E. (1985). The VLSI design automation assistant: What's in a knowledge base. Dans 22nd ACM/IEEE Design Automation Conference (pp. 252-258). doi: 10.1109/DAC.1985.1585949

[20] Kowalski, T., Geiger, D., Wolf, W., & Fichtner, W. (1985). The VLSI design automation assistant: From algorithms to silicon. IEEE Design & Test of Computers, 2(4), 33-43.

[21] Hsieh, J. Y. J., Mahoney, J. E., Ngo, L. T., & Shelly, L. (1986). A User Programmable Reconfigurable Logic Array.

[22] Flake, P., Moorby, P., Golson, S., Salz, A., & Davidmann, S. (2020). Verilog HDL and its ancestors and descendants. Proc. ACM Program. Lang., 4(HOPL), Article 87. doi: 10.1145/3386337. Repéré à https://doi.org/10.1145/3386337.

[23] Lipsett, R., Marschner, E., & Shahdad, M. (1986). VHDL-the language. IEEE Design & Test of Computers, 3(2), 28-41.

[24] Lis, J. S., & Gajski, D. D. (1988). Synthesis from VHDL. Dans Proceedings 1988 IEEE International Conference on Computer Design: VLSI (pp. 378,379,380,381-378,379,380,381). IEEE Computer Society.

[25] Coussy, P., Gajski, D. D., Meredith, M., & Takach, A. (2009). An Introduction to High-Level Synthesis. IEEE Design & Test of Computers, 26(4), 8-17. doi: 10.1109/MDT.2009.69

[26] Canis, A., Choi, J., Aldham, M., Zhang, V., Kammoona, A., Czajkowski, T., Anderson, J. (2013). LegUp: An open-source high-level synthesis tool for FPGA-based processor/accelerator systems. ACM Transactions on Embedded Computing Systems (TECS), 13(2), 1-27. doi: 10.1145/2514740.

[27] Xilinx. (2021). Vivado Design Suite User Guide, High-Level Synthesis , UG902 (v2020.1). Repéré à https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_3/ug902-vivado-high-level-synthesis.pdf.

[28] Intel. (2021b). Intel High Level Synthesis Compiler Pro Edition : Reference Manual (MNL- 1083).

[29] Intel. (2013). Implementing FPGA Design with the OpenCL Standard. White paper.

[30] S. Sjoholm and L. Lindh. VHDL for Designers. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1997. ISBN 0134734149.

[31] P. Coussy and A. Morawiec. High-Level Synthesis : From Algorithm to Digital Circuit. Springer Publishing Company, Incorporated, 1st edition, 2008. ISBN 1402085877, 9781402085871.

[32] A. Nicolau and R. Potasmann. Incremental tree height reduction for high level synthesis. In Proceedings of the 28th ACM/IEEE Design Automation Conference, DAC '91, pages 770–774, New York, NY, USA, 1991. ACM. ISBN 0-89791-395- 7. doi : 10.1145/127601.127767. URL http://doi.acm.org/10.1145/127601.127767.

[33] S. Bakshi, D. D. Gajski, and H. Juan. Component selection in resource shared and pipelined dsp applications. In Design Automation Conference, 1996, with EURO-VHDL '96 and Exhibition, Proceedings EURO-DAC '96, European, pages 370–375, Sep 1996. doi : 10.1109/EURDAC.1996.558231.

[34] P. Gutberlet, J. Muller, H. Kramer, and W. Rosenstiel. Automatic module allocation in high level synthesis. In Design Automation Conference, 1992., EURO-VHDL '92,

EURO-DAC ’92. European, pages 328–333, Sep 1992.
doi : 10.1109/EURDAC.1992.246223.

[35] R. A. Walker and S. Chaudhuri. Introduction to the scheduling problem. IEEE Design Test of Computers, 12(2) :60–69, Summer 1995. ISSN 0740-7475.
doi : 10.1109/54.386007.

[36] J. Cong and J. Xu. Simultaneous fu and register binding based on network flow method. In Proceedings of the Conference on Design, Automation and Test in Europe, DATE ’08, pages 1057–1062, New York, NY, USA, 2008. ACM. ISBN 978-3-9810801-3-1. doi : 10.1145/1403375.1403629.
URL http://doi.acm.org/10.1145/1403375.1403629.

[37] P. Coussy, D. D. Gajski, M. Meredith, and A. Takach. An introduction to highlevel synthesis. IEEE Design Test of Computers, 26(4) :8–17, July 2009. ISSN 0740-7475.
doi : 10.1109/MDT.2009.69.

[38] Handle-C. https://www.mentor.com/products/fpga/handel-c/.

[39] M. Gokhale, J. Stone, J. Arnold, and M. Kalinowski. Stream-oriented fpga computing in the streams-c high level language. In Proceedings 2000 IEEE Symposium on Field-Programmable Custom Computing Machines (Cat. No.PR00871), pages 49–56, 2000.
doi : 10.1109/FPGA.2000.903392.

[40] C. A. R. Hoare. Communicating sequential processes. Commun. ACM, 26(1) : 100–106, January 1983. ISSN 0001-0782. doi : 10.1145/357980.358021. URL http://doi.acm.org/10.1145/357980.358021.

[41] S. Gupta, N. Dutt, R. Gupta, and A. Nicolau. Spark : a high-level synthesis framework for applying parallelizing compiler transformations. In 16th International Conference on VLSI Design, 2003. Proceedings., pages 461–466, Jan 2003.
doi : 10.1109/ICVD.2003.1183177.

[42] Catapult. https://www.mentor.com/hls-lp/catapult-high-level-synthesis/.

[43] Vivado-HLS. https://www.xilinx.com/products/design-tools/vivado/ integration/esl-design.html/.

[44] J. Teich. Hardware/software codesign : The past, the present, and predicting the

future. Proceedings of the IEEE, 100(Special Centennial Issue) :1411–1430, May 2012 . ISSN 0018-9219. doi : 10.1109/JPROC.2011.2182009.

[45] Xilinx. https://www.xilinx.com/support/documentation/sw_manuals/ug998-vivado-intro-fpga-design-hls.pdf, a.

[46] Xilinx. https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug1037-vivado-axi-reference-guide.pdf, c.

[47] Xilinx. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_1/ug902-vivado-high-level-synthesis.pdf, b.

[48] B. Said, B. Assia, T. Mohamed, S. Florentin, and V. Luige, “Applying dijkstra algorithm for solving neutrosophic shortest path problem,” in Advanced Mechatronic Systems (ICAMechS), 2016 International Conference on, 2016, pp. 412–416.

[49] M. Brady, “Artificial intelligence and robotics,” Artif. Intell., vol. 26, no. 1, pp.79–121, 1985.

[50] O. Khatib, “Real-time obstacle avoidance for manipulators and mobile robots,” in Proceedings of the 1985 IEEE International Conference on Robotics and Automation, St. Louis, Missouri, USA, March 25-28, 1985, vol. 2, 1985, pp. 500–505.

[51] K. Ali, “Processus de conception conjointe logiciel matériel dirigés par les modèles,” Ph.D. dissertation, Lille 1, 2010.

[52] L. Sifan, H. Lin, L. Yang, and S. Dong, “Integrated modular avionics anomaly detection based on symbolic time series analysis,” in Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), 2017 IEEE 2nd, 2017, pp. 2095–2099.

[53] C. Shi-Gang, W. Hui, and Y. Li, “A simulation study of a-star algorithm for robot path planning,” in 16th international conference on mechatronics technology, PP, 2012, pp. 506–510.

[54] H. I. Kang, B. Lee, and K. Kim, “Path planning algorithm using the particle swarm optimization and the improved dijkstra algorithm,” in PACIIA 2008, Volume 2, 2008 IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application,

19-20 December 2008, Wuhan, China, 2008, pp. 1002–1004.

[55] Windh, S., Ma, X., Halstead, R. J., Budhkar, P., Luna, Z., Hussaini, O., & Najjar, W. A. (2015). High-Level Language Tools for Reconfigurable Computing. Proceedings of the IEEE, 103(3), 390-408. doi: 10.1109/JPROC.2015.2399275.

[56] Xilinx. (2018b). Vivado Design Suite User Guide: High-Level Synthesis, UG902 V2017.4.

[57] Intel. (2013). Implementing FPGA Design with the OpenCL Standard. White paper.

[58] Arvind, R. S. N. (2009). What is Bluespec? SIGDA Newsl., 39(1), 1. doi:10.1145/ 1862876.1862877. Repéré à https://doi.org/10.1145/1862876.1862877.

[59] Najjar, W. A., Villarreal, J., & Halstead, R. J. (2016). ROCCC 2.0. Dans FPGAs for Software Programmers (pp. 191-204). Springer.

[60] Xilinx. (2012b). ChipScope Pro Software and Cores: User Guide UG029. Repéré à https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/chipscope_pro_sw_cores_ug029.pdf.

[61] Nane, R., Sima, V., Pilato, C., Choi, J., Fort, B., Canis, A., Bertels, K. (2016). A Survey and Evaluation of FPGA High-Level Synthesis Tools. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 35(10), 1591-1604. doi: 10.1109/TCAD.2015.2513673.

[62] Nane, R., Sima, V., Olivier, B., Meeuws, R., Yankova, Y., & Bertels, K. (2012). DWARV 2.0: A CoSy-based C-to-VHDL hardware compiler. Dans 22nd International Conference on Field Programmable Logic and Applications (FPL) (pp. 619-622). doi:10.1109/ FPL.2012.6339221.

[63] Pilato, C., & Ferrandi, F. (2013). Bambu: A modular framework for the high level synthesis of memory-intensive applications. Dans 2013 23rd International Conference on Field programmable Logic and Applications (pp. 1-4). doi: 10.1109/FPL.2013.6645550.

[64] Lahti, S., Sjövall, P., Vanne, J., & Hämäläinen, T. D. (2019). Are We There Yet? A Study on the State of High-Level Synthesis. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 38(5), 898-911.

doi: 10.1109/TCAD.2018.2834439.

[65] Bechhoefer, Eric. "Condition Based Maintenance Fault Database for Testing Diagnostics and Prognostic Algorithms." 2013. https://www.mfpt.org/fault-data-sets/