CHAPITRE 4

IMPLÉMENTATION DU FILTRE EN VHDL ET SA RÉALISATION SUR FPGA

4.1 Introduction

Dans ce chapitre, l'implémentation du filtre CIC décimateur en utilisant le langage VHDL est présentée. Cette implémentation est basée sur une représentation à point fixe. Le diagramme bloc du filtre est subdivisé en plusieurs sous-blocs et le codage au niveau RTL est effectué pour chaque bloc. Après le codage RTL, les différents sous-blocs sont interconnectés et une vérification du CIC décimateur au complet, pour différentes configurations, est effectuée. Finalement, quelques considérations pour la synthèse et l'optimisation sont présentées.

4.2 Architecture du CIC décimateur polyphasé

L'architecture du CIC décimateur polyphasé à implémenter a déjà été présentée à la section 2.3.2.2. Pour faciliter la compréhension des différentes opérations qu'effectue le filtre CIC décimateur et les différents aspects à prendre en considération pour le codage en VHDL, cette architecture est reprise à la figure 31. À partir d'ici, on utilisera l'expression « noyau programmable » pour designer le filtre CIC décimateur polyphasé.

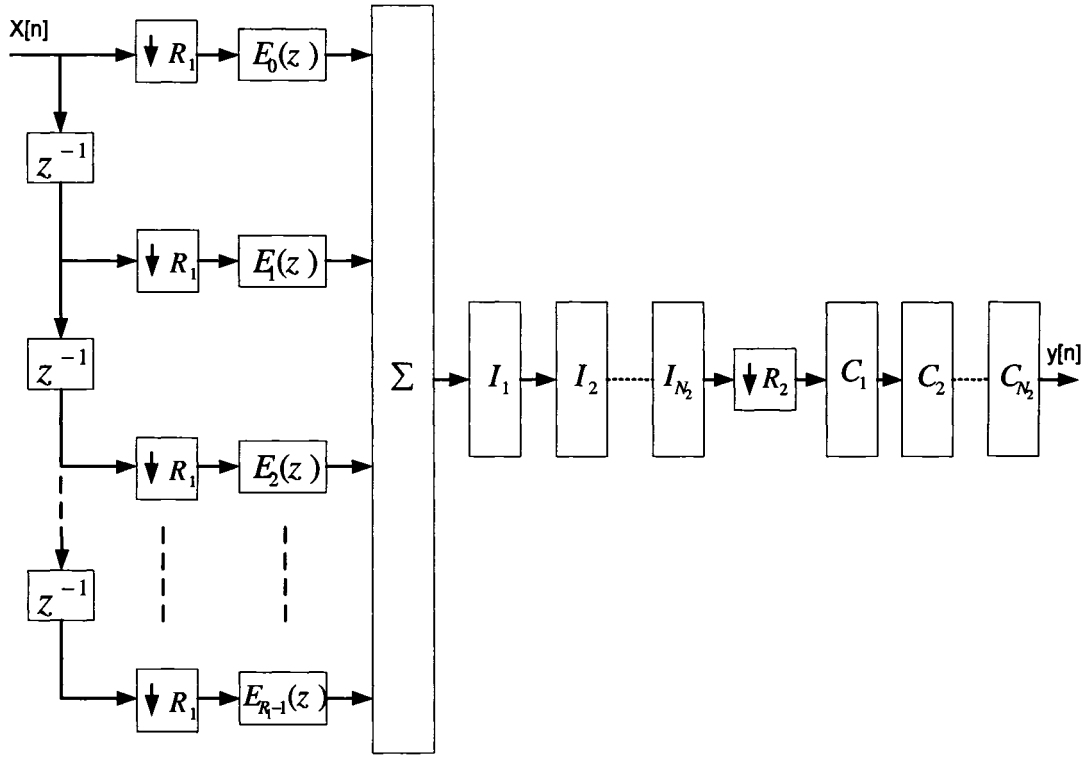


Figure 31 Architecture parallèle du CIC décimateur polyphasé

Avec cette architecture, le noyau programmable est constitué de deux principaux étages de décimation interconnectés de manière sérielle. Le premier étage, placé à l'entrée du noyau programmable, est un filtre FIR polyphasé. Le deuxième étage est un filtre CIC décimateur régulier. Les composantes polyphasées du filtre FIR et les intégrateurs du filtre CIC décimateur régulier fonctionnent à la même fréquence. Par contre, les différentiateurs du CIC décimateur régulier fonctionnent à une fréquence plus basse d'un facteur de R_2 .

4.3 Particularités des différentes configurations

L'utilisateur du noyau programmable peut choisir les valeurs des différents paramètres. Cependant, son choix a une influence directe sur l'architecture matérielle qui sera

implémentée. Tous les paramètres sont programmables avant synthèse à l'exception du facteur de décimation qui peut l'être après synthèse. Ainsi, ce facteur de décimation peut être fixe ou programmable.

4.3.1 Cas d'un facteur de décimation fixe

Si le facteur de décimation est fixe, alors sa valeur ne peut être choisie qu'avant synthèse. Ainsi, le calcul de la largeur maximale des registres sera basé sur cette valeur. L'avantage de cette configuration réside dans le fait que toute la largeur binaire des registres est utilisée pour les différents calculs à faire. Cependant, cette configuration est trop limitée du point de vue de la reconfigurabilité.

4.3.2 Cas d'un facteur de décimation programmable

Si le facteur de décimation est programmable, alors sa valeur peut être choisie avant et après synthèse. Dans ce cas, l'utilisateur doit spécifier la valeur maximale que peut prendre le facteur de décimation après synthèse. Cette valeur maximale sera utilisée pour calculer la largeur maximale des registres. De ce fait, si un facteur de décimation, de valeur outre que la valeur maximale, est utilisé, alors toute la largeur binaire des registres n'est pas utilisée. Ceci veut dire qu'on peut utiliser des largeurs de 40 bits alors qu'on en a besoin que des largeurs de 30 bits. Cependant, cette configuration est reconfigurable ce qui est très important dans le design d'un noyau programmable.

4.4 Implémentation en VHDL

La conception du noyau programmable en VHDL a été divisée en trois niveaux. D'abord, on retrouve le niveau supérieur qui permet à l'utilisateur de choisir les valeurs des différents paramètres. Ensuite, le niveau intermédiaire qui est divisé en trois *PROCESS* : un pour le filtre RIF polyphasé, un pour le CIC décimateur régulier et un

pour le module de contrôle. Enfin, on retrouve au plus bas le niveau, les sous-modules qui sont constitués d'additionneurs, de soustracteurs et d'un compteur.

4.4.1 Niveau supérieur

Le niveau supérieur du noyau programmable établit l'interface avec l'usager. C'est à ce niveau que tous les paramètres programmables peuvent être choisis à l'aide de *generics*. Le nombre d'entrées parallèles et l'ordre du CIC décimateur régulier dictent respectivement le nombre de composantes polyphasées et le nombre d'étages d'intégrateurs et de différentiateurs à utiliser. Étant donné que les coefficients des composantes polyphasées sont fixes pour un nombre d'entrées parallèles donné, alors un *case-is* est utilisé pour choisir les coefficients qu'il faut pour le nombre d'entrées parallèles choisi. En ce qui concerne les étages d'intégrateurs et de différentiateurs, c'est une boucle *for-loop* qui s'exécute.

De nombreuses fonctions s'activent à la compilation avant la synthèse et permettent de fixer adéquatement les largeurs des mots binaires ainsi que les sous-modules créés au niveau inférieur. Toutes ces fonctions sont regroupées dans un package appelé fonctions_utiles et est présenté à la figure 32. La figure 33 montre le symbole du noyau programmable et la figure 34 les paramètres programmables

```
PACKAGE fonctions_utiles IS

FUNCTION int_2_std_logic_vector (value, bitwidth : INTEGER )
   RETURN std_logic_vector;
FUNCTION bitsneededtorepresent( a_value : INTEGER )
   RETURN INTEGER;

FUNCTION extend (vector: std_logic_vector; bits: INTEGER)
```

```
    RETURN std_logic_vector;

FUNCTION std_logic_vector_2_var (vect: std_logic_vector)
    RETURN std_logic_vector;

END fonctions_utiles;
```

Figure 32 Package fonctions_utiles

Fichier : CIC_poly.vhd

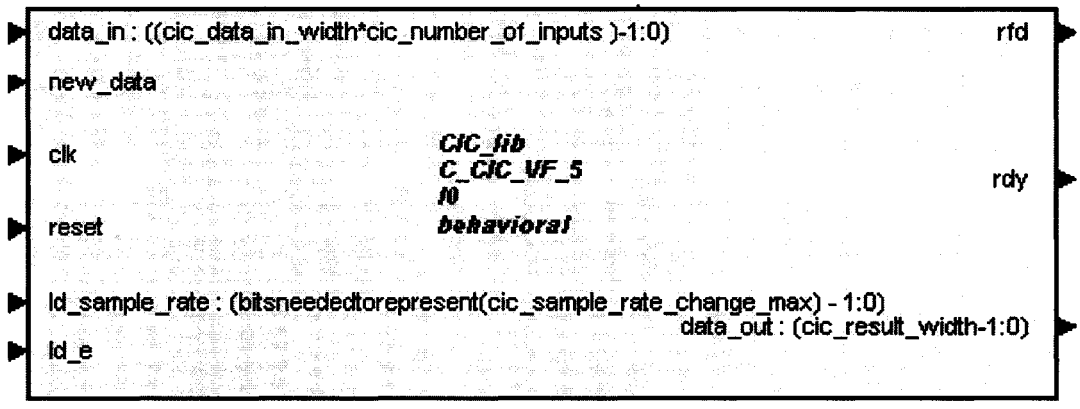


Figure 33 Symbole du noyau programmable

GENERIC:

cic_data_in_width	INTEGER	Largeur binaire de l'entrée du noyau programmable
cic_number_of_inputs	INTEGER	Nombre d'entrées parallèles(R_1)
cic_width_fir_poly	INTEGER	Largeur binaire maximale des bus du FIR polyphasé

cic_stages_cic_reg	INTEGER	L'ordre du CIC régulier (nombre d'intégrateurs et de différentiateurs)
cic_sample_rate_change_type	INTEGER	Type de décimation (1 pour programmable et 2 pour fixe)
cic_sample_rate_change	INTEGER	Facteur de décimation du CIC régulier (R_2)
cic_sample_rate_change_max	INTEGER	Facteur de décimation maximale du CIC régulier dans le cas d'un type de décimation programmable
cic_differential_delay	INTEGER	Délai différentiel des différentiateurs
cic_width_max	INTEGER	Largeur binaire maximale des bus du CIC régulier
cic_result_width	INTEGER	Largeur binaire de la sortie du noyau
cic_latency	INTEGER	Latence de la sortie (permet de retarder la sortie du noyau

Figure 34 Paramètres programmables

Les ports d'entrée et de sortie du noyau programmable ainsi que leur définition sont présentés dans la table 1.

Tableau VII

Ports d'entrée/sortie du noyau et leur définition

Nom du port	Direction	Description
clk	entrée	Horloge
data_in	entrée	Port d'entrée de données du noyau
new_data	Entrée	Signal d'activation de l'horloge *clock enable.* Quand ce signal est actif, les échantillons de donnés présents sur le port data_in sont chargés dans le filtre
reset	entrée	Permet d'initialiser le noyau
ld_sample_rate	entrée	Permet de charger la nouvelle valeur du facteur de décimation si l'option 1 est choisie pour le cic_sample_rate_change_type. Ce port est optionnel et est disponible seulement pour l'option programmable du cic_sample_rate_change_type.
ld_e	entrée	Ce signal est associé avec le port ld_sample_rate. La valeur du port ld_sample_rate est prise en compte par le noyau quand ld_e actif coincide avec un front montant du clock. Comme le port ld_sample_rate, ce port est disponible seulement pour l'option programmable du cic_sample_rate_change_type.
rfd	sortie	Indique si le noyau peut accepter une nouvelle donnée sur Sur le port d'entrée data_in
rdy	sortie	Indique la disponibilité d'une sortie sur le port de sortie data_out
data_out	sortie	Port de sortie du noyau

Les ports *nd (new data), rfd (ready for data) et rdy (ready)* sont utilisés pour contrôler les entrées et les sorties du noyau. Le signal *rfd*, qui est actif haut, indique au système

que le filtre est prêt à accepter des données en entrée. Quand le signal *nd* est actif, il signale au noyau la disponibilité d'une nouvelle entrée sur le port d'entrée *data_in*. Le signal de sortie *rdy* indique la disponibilité d'une nouvelle sortie sur le port *data_out*.

Les signaux de l'interface du noyau peuvent être utilisés de la manière suivante : L'utilisateur du système doit d'abord attendre que *rfd* = 1. Ceci signale qu'une nouvelle entrée peut être traitée par le filtre. La nouvelle entrée est placée sur le port d'entrée *data_in* et *nd* devrait être en mode actif (*nd* = 1) pour un cycle d'horloge. *nd* actif indique au noyau que les données sur le port *data_in* doivent être traitées. Ainsi, le filtre traite les données au front montant de l'horloge qui coïncide avec *nd* = 1. Les données à la sortie du filtre peuvent être récupérées si le noyau assigne la valeur 1 à *rdy*. Ce signal *rdy* peut être utilisé comme *clock enable* par le bloc en bande de base qui récupère les sorties du filtre.

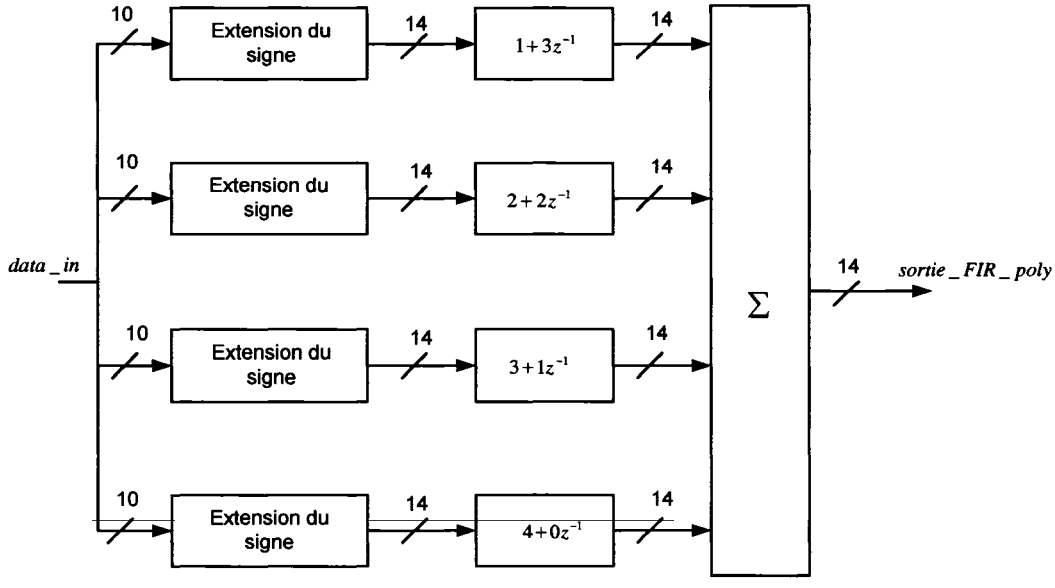
Supposons les deux cas de configurations suivantes :

Cas 1 :

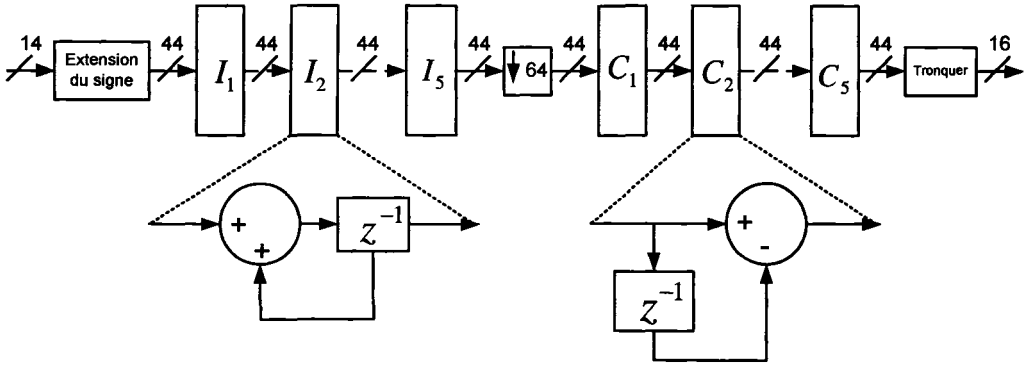
Paramètres	Valeur
cic_data_in_width	10
cic_stage_cic_reg	5
cic_number_of_inputs (R_1)	4
cic_width_fir_poly	$2\log_2(R_1)+10 = 14$ bits
cic_sample_rate_change_type	2 (c'est-à-dire fixe)
cic_sample_rate_change (R_2)	64
cic_differential_delay	1
cic_width_max	$5\log_2(R_2M)+14 = 44$ bits
cic_result_width	16 bits
cic_latency	1

```
ENTITY C_CIC_VF IS
 GENERIC (
        cic_data_in_width              : INTEGER: = 10;
        cic_number_of_inputs           : INTEGER: = 4;
        cic_width_fir_poly             : INTEGER: = 14;
        cic_stages_cic_reg             : INTEGER: = 5;
        cic_sample_rate_change_type    : INTEGER: = 1;
        cic_sample_rate_change         : INTEGER: = 64;
        cic_sample_rate_change_max     : INTEGER: = 64;
        cic_differential_delay         : INTEGER: = 1;
        cic_width_max                  : INTEGER: = 44;
        cic_result_width               : INTEGER: = 16;
        cic_latency                    : INTEGER: = 1);
```

La figure 35 montre l'architecture générée pour cette configuration:



(a) FIR polyphasé



(b) CIC régulier

Figure 35 Architecture générée pour la configuration du cas 1

Une fois cette architecture implémentée, une valeur du facteur de décimation supérieure à celle spécifiée dans le *generic* ne peut pas être utilisée car le calcul de la largeur binaire des bus de données est basé sur cette dernière.

Cas 2 :

Le deuxième cas de configuration est la même chose que le premier, sauf que le type de décimation est programmable. Supposons maintenant que le nombre d'entrées parallèles est égale à 4 et que le CIC régulier doit supporter un facteur de décimation maximale de $R_2 = 256$. Ceci veut dire que le filtre doit supporter un facteur de décimation total $R = R_1 R_2 = 4 * 256 = 1024$. Le filtre FIR polyphasé généré pour cette configuration est le même que celui du cas 1 mais, par contre, le CIC régulier généré est différent et est présenté à la figure 36.

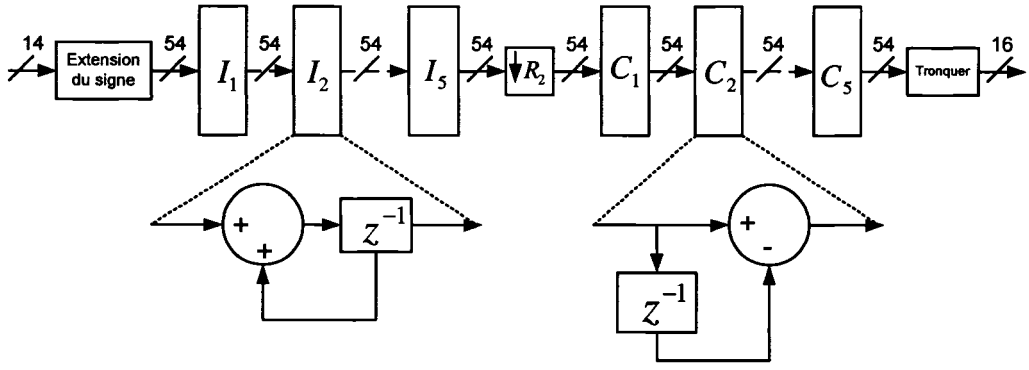


Figure 36 CIC régulier généré pour le deuxième cas de configuration

4.4.2 Niveau intermédiaire

Le niveau intermédiaire contient trois *PROCESS* différents pour ses trois différents blocs qui sont : le FIR polyphasé, le CIC régulier et le module de contrôle.

4.4.2.1 FIR polyphasé

Le nombre d'entrées parallèles du FIR polyphasé peut être 2, 4 ou 8. Comme il est impossible d'avoir des entrées qui changent dynamiquement, le port $data\_in$, qui est l'entrée du noyau programmable et du FIR polyphasé, a une largeur binaire de $(R_1 \times CIC\_data\_in\_width)$ bits qui permet d'accommoder le nombre d'entrées parallèles. Cette largeur binaire sera séparée en R_1 entrées parallèles de largeurs binaires $CIC\_data\_in\_width$ bits chaque. Ces R_1 entrées parallèles de largeurs binaires $CIC\_data\_in\_width$ bits chaque sont les entrées parallèles du FIR polyphasé. Le module du FIR polyphasé possède des signaux de contrôle local qui sont *fir_poly_new_data* et *fir_poly_rdy*. Le signal *fir_poly_new_data* joue le rôle de *clock enable* de l'horloge du module FIR polyphasé. Le signal *fir_poly_rdy*, quand à lui,

indique la disponibilité d'un échantillon sur la sortie du FIR polyphasé. Ainsi, en résumé, le FIR polyphasé fonctionne comme suit : Quand une nouvelle entrée est placée sur le port d'entrée *data_in*, le signal *fir_poly_new_data* est en mode actif (*fir_poly_new_data* = 1) pour un cycle d'horloge. *fir_poly_new_data* actif indique au FIR polyphasé que les données sur le port *data_in* doivent être traitées. Ainsi, le filtre traite les données au front montant de l'horloge qui coïncide avec *fir_poly_new_data* = 1. Les données à la sortie du FIR polyphasé peuvent être récupérées si la valeur 1 est assignée au signal *fir_poly_rdy*. Ce signal est utilisé comme *clock enable* par le bloc d'intégrateurs qui récupère les sorties du FIR polyphasé.

4.4.2.2 CIC régulier

La sortie du FIR polyphasé est présentée à l'entrée du CIC régulier qui est constitué d'un bloc d'intégrateur, d'un facteur de décimation et d'un bloc de différentiateurs. Ainsi, deux *PROCESS* sont utilisés pour le traitement des données dans les blocs d'intégrateurs et de différentiateurs.

Le bloc d'intégrateurs possède des signaux de contrôle qui sont : *integ_new_data* et *integ_rdy*. Étant donné que le bloc d'intégrateur fonctionne à la même fréquence que le FIR polyphasé, alors son signal *integ_new_data* est actif quand *fir_poly_rdy* l'est. Ainsi, le signal présent à l'entrée du bloc d'intégrateur est traité et la valeur 1 est assignée au signal *integ_rdy* qui indique la disponibilité de la sortie du bloc d'intégrateurs.

Le facteur de décimation, qui est réalisé à l'aide d'un compteur, permet de choisir la sortie du bloc d'intégrateurs qui sera présentée à l'entrée du bloc de différentiateurs. Ainsi, si une entrée valable est présentée à l'entrée du bloc de différentiateurs, son signal de contrôle *comb_new_data* est actif. De ce fait, le signal est traité et la valeur 1 est assignée au signal de contrôle *comb_rdy* ce qui indique la disponibilité de la sortie du bloc de différentiateurs. Cette sortie est tronquée et retardée en fonction du délai spécifié

dans *cic_latency* du *generic*. Le résultat est placé dans le port de sortie *data_out* pour donner le résultat final du noyau programmable.

4.4.2.3 Module de contrôle

Le module de contrôle permet de faire le contrôle global du noyau. Il permet de séquencer les signaux de contrôle local du FIR polyphasé et du CIC régulier. Il est essentiellement constitué d'un compteur dont la taille dépend de la valeur du facteur de décimation du CIC régulier.

4.4.3 Sous-modules

Le noyau programmable est constitué d'additionneurs, de soustracteurs, de délais, d'intégrateurs, de différentiateurs et d'un compteur. Cependant, les trois sous-modules de base sont l'intégrateur, le différentiateur et le compteur.

4.4.3.1 Intégrateur

Un intégrateur numérique n'est rien d'autre qu'un accumulateur. Il est constitué d'un additionneur et d'une ligne à délai qui permet de réaliser la boucle de rétroaction. La figure 37 montre un bloc d'intégrateur selon *System generator* de Xilinx. b est l'entrée et q la sortie . L'équation réalisée par ce bloc est la suivante :

$$q(n) = q(n-1) + b(n) \tag{4.1}$$

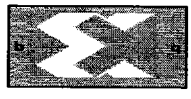


Figure 37 bloc d'intégrateur

L'équation (4.1) montre que la réalisation d'un intégrateur en VHDL est très facile. Ainsi, pour générer N blocs d'intégrateurs en cascade, la boucle *for-loop* est utilisée.

4.4.3.2 Différentiateur

Un différentiateur numérique est réalisé à l'aide d'une ligne délai et d'un soustracteur. La figure 38 montre un bloc de différentiateur avec un délai différentiel $M = 1$ selon *System generator* de Xilinx. Si l'entrée du bloc est $x(n)$ et la sortie $y(n)$, alors l'équation réalisée par ce bloc est la suivante :

$$y(n) = x(n) - x(n-1) \tag{4.2}$$

avec $a = x(n)$ et $b = x(n-1)$. En VHDL, la boucle *for-loop* est utilisée pour générée N blocs de différentiateurs en cascade.

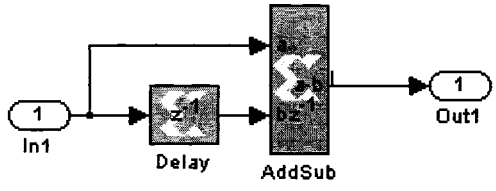


Figure 38 Bloc de différentiateur

4.4.3.3 Compteur

Le facteur de décimation du CIC régulier, R_2, est réalisé à l'aide d'un compteur. Le compteur est incrémenté à chaque fois qu'une sortie est disponible sur le port de sortie du bloc d'intégrateurs. Ainsi, quand le compteur atteint la valeur de R_2, le signal de contrôle *comb_new_data* est mis à 1 pour permettre au bloc de différentiateur de

récupérer la sortie du bloc d'intégrateurs. Étant donné que le facteur de décimation R_2 peut changer de valeur après la synthèse, on peut se poser la question à savoir qu'est ce qui arrive si le facteur de décimation change de valeur alors que le compteur n'a pas encore atteint l'ancienne valeur de R_2. Dans ce cas, le compteur continue à compter jusqu'à ce qu'il atteigne l'ancienne valeur de R_2 et s'initialise à zéro. À partir de ce moment, il compte pour atteindre la nouvelle valeur de R_2.

4.5 Vérification du modèle VHDL

La vérification est extrêmement importante dans un processus de design utilisant les FPGA ou les ASIC. Ainsi, concevoir un circuit, c'est réfléchir à la manière dont il sera décomposé : quelle sera la structure, comment vont se propager les données, où sont les éléments critiques en vitesse, etc, sont autant des questions auxquelles le designer doit répondre pendant la phase de conception. Cette phase est aussi celle durant laquelle est écrit le code VHDL du design. La grande facilité avec laquelle on peut créer des *testbench* (bancs de test) pour les modules VHDL est telle qu'on observe souvent les designers tester les modules au fur et à mesure qu'ils les décrivent. Nous distinguerons donc la simulation de bas niveau, qui permet de tester plus ou moins exhaustivement chaque module ou sous-module, de la simulation de haut niveau, qui a pour objectif de valider le design complet tel qu'il est décrit dans la spécification. Cette méthodologie *bottom-up* (de bas en haut) avec une validation des sous-modules et modules peut diminuer le nombre total de test à faire et engendrer un taux de confiance suffisant [24].

Les modules et les sous-modules ont été validés en utilisant des *testbenchs* spécialisés qui isolent l'entité à tester. De ce fait, la complexité de l'exercice de conception diminue, ce qui permet d'envisager de tester plus complètement la fonction. Les modules ou les sous-modules sont testés de manière presque complète, ce qui augmente la confiance de leur fonctionnalité. Cette technique a été utilisée pour tester le FIR

polyphasé et le CIC régulier. Pour la validation de la fonction à haut niveau, il suffit de vérifier les interconnexions et le timing. La figure 39 montre la vérification à haut niveau du noyau programmable.

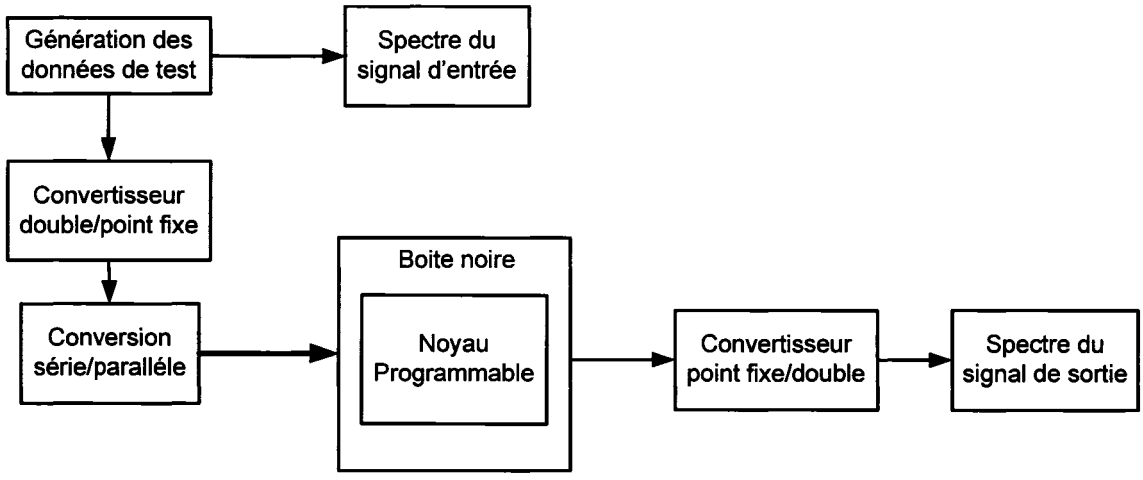
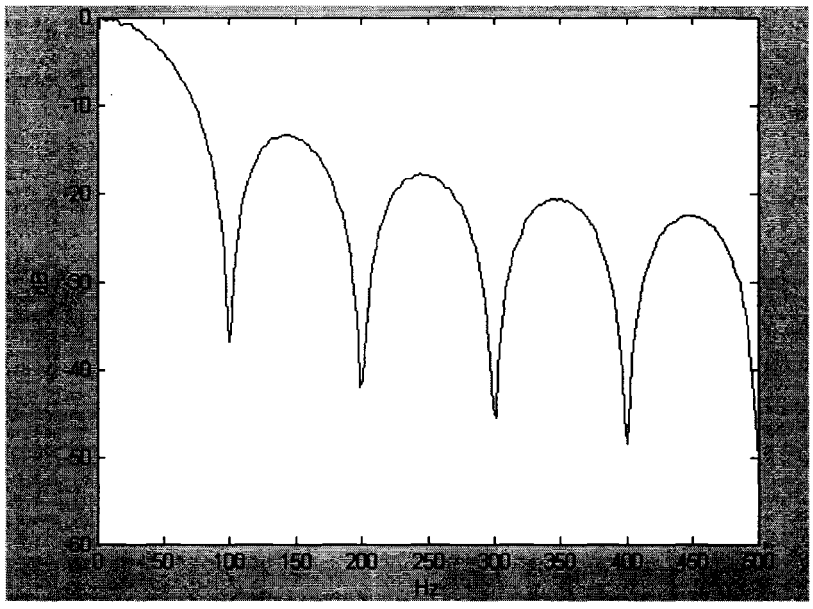
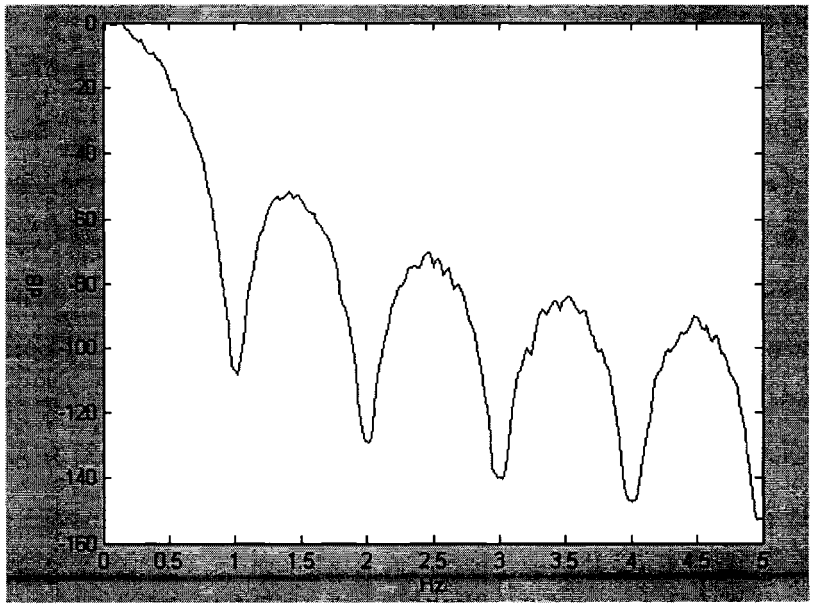


Figure 39 Vérification à haut niveau du noyau programmable

Dans la figure 39, le code VHDL du noyau programmable, à l'aide de Modelsim, a été encapsulé dans une boîte noire pour pouvoir être utilisé dans un modèle *Simulink*. Les données de test générées sont de type double. Le convertisseur double/point fixe permet de convertir les données de test en une arithmétique binaire point fixe complément à deux. Pour interfacer les données avec le bloc du noyau programmable, un convertisseur série parallèle est utilisé. En comparant les spectres du signal d'entrée et de sortie, nous pouvons dire si le noyau fonctionne bien ou pas à haut niveau. La figure 40 montre les spectres des signaux d'entrée et de sortie avec un facteur de décimation $R = 100 (R_1 = 4, R_2 = 25)$ et d'ordre $N = 4$. La figure 40 (a) montre le spectre du signal d'entrée avec une largeur de bande de 100 Hz. La figure 40 (b) montre le spectre normalisé du signal de sortie avec une largeur de bande de 1 Hz. Ceci montre bien que le signal d'entrée a été décimé par 100. Il est intéressant de remarquer l'atténuation de 48 dB (à peu près) sur le spectre du signal décimé.



(a) Spectre d'entrée



(b) Spectre de sortie

Figure 40 Spectres d'entrée et de sortie du noyau programmable avec $R = 100$ et $N = 4$

Pour pouvoir dire que le noyau programmable fonctionne correctement, il faudrait, pour les mêmes spécifications, comparer sa réponse en fréquence avec celle du CIC régulier. Pour ce faire, on utilise, comme référence, le bloc CIC de simulink qui est disponible sous la rubrique *Xilinx blockset.* La structure de vérification est présentée à la figure 41.

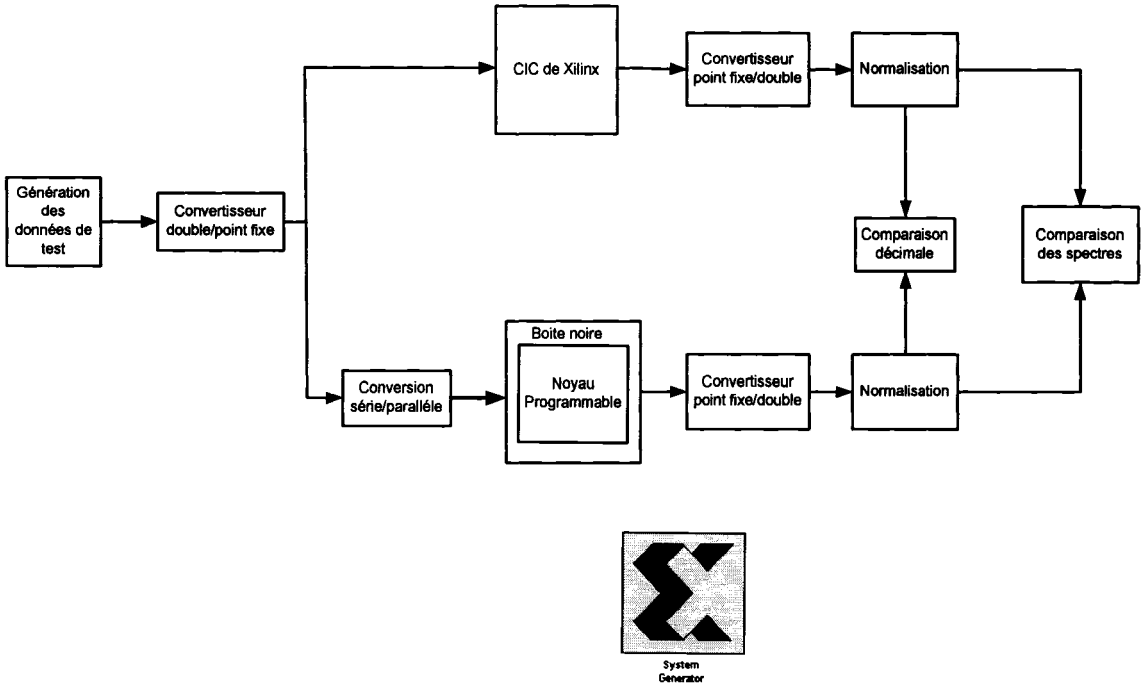
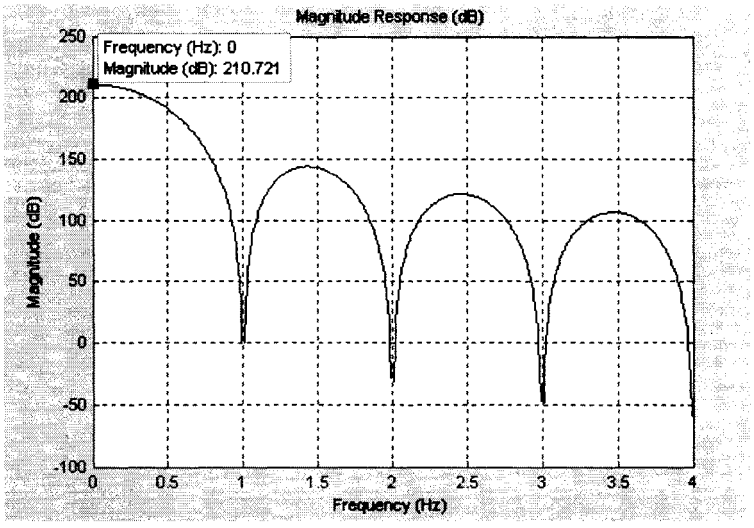


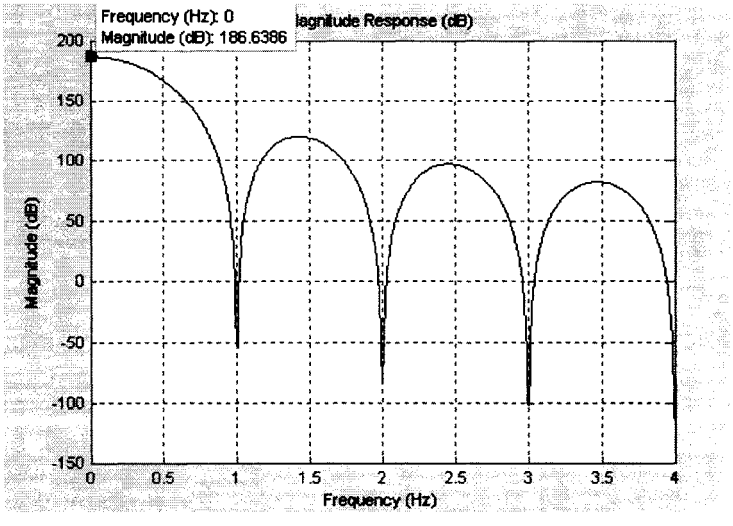
Figure 41 Comparaison du noyau programmable avec le CIC régulier de Xilinx

Dans la figure 41, les données de test sont générées par un générateur de données aléatoire. Le convertisseur double/ point fixe convertit les données de type double en binaire point fixe complément à deux pour que ces dernières puissent être traitées par les filtres. Le convertisseur série/parallèle permet d'interfacer les données qui arrivent de façon sérielle avec les entrées parallèles du noyau. Aux sorties des filtres, les données sont converties en format double par les convertisseurs point fixe/double. À ce stade, vu que les deux filtres n'ont pas le même gain DC, les données sont normalisées pour pouvoir être comparées. Le bloc comparaison des spectres contient du code Matlab qui

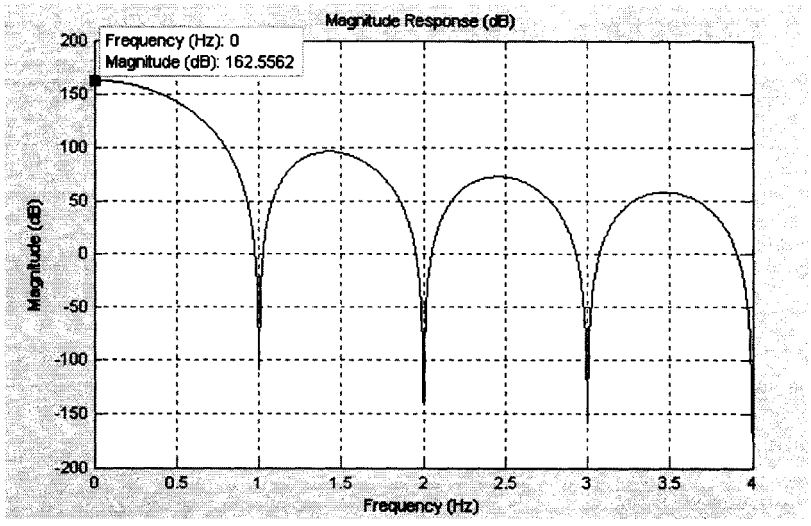
permet de tracer les spectres de sorties sur la même figure. De ce fait, on pourra les comparer. La figure 42 montre, pour $f_s = 128$ Hz, $R = 128$, $N = 5$ et $M = 1$ les spectres du CIC régulier de Xilinx et du noyau programmable avec 2, 4 et 8 entrées parallèles.



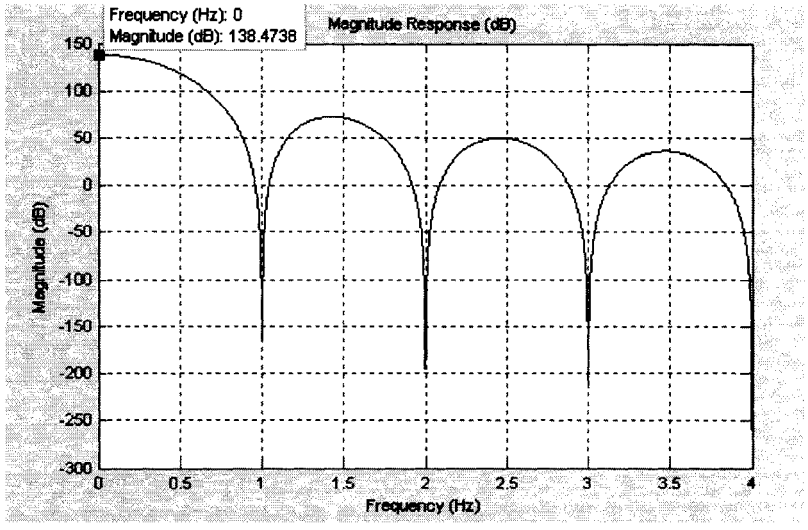
(a) CIC régulier de Xilinx



(b) Noyau programmable avec deux entrées parallèles



(c) Noyau programmable avec quatre entrées parallèles



(d) Noyau programmable avec huit entrées parallèles

Figure 42 Spectres non normalisés du CIC régulier de Xilinx et du noyau programmable

La première remarque à faire sur les spectres présentés à la figure 42 est la différence sur le gain DC. Le gain DC d'un CIC régulier est donné par l'équation suivante :

$$G(dB) = 20\log\left((RM)^N\right) \tag{4.3}$$

En remplaçant les inconnus de l'équation (4.3) par leur valeur, le gain DC du CIC régulier présenté à la figure 10 (a) donne :

$$G(dB) = 20\log\left((128\times1)^5\right) = 210.72 \text{ dB}$$

En faisant une décomposition polyphasé du CIC régulier, le gain DC se transforme et s'écrit comme suit :

$$G(dB) = 20\log(R_1) + 20\log\left((R_2M)^N\right) \tag{4.4}$$

Ainsi, pour $R_1 = 2$ c'est-à-dire deux entrées parallèles, $R_2 = 64$ et le gain DC du CIC polyphasé présenté à la figure 10 (b) est donné par :

$$G(dB) = 20\log(2) + 20\log\left((64\times1)^5\right) = 186.63 \text{ dB}$$

Pour $R_1 = 4$ c'est-à-dire quatre entrées parallèles, $R_2 = 32$ et le gain DC du CIC polyphasé présenté à la figure 10 (c) est donné par :

$$G(dB) = 20\log(4) + 20\log\left((32\times1)^5\right) = 162.55 \text{ dB}$$

Pour $R_1 = 8$ c'est-à-dire huit entrées parallèles, $R_2 = 16$ et le gain DC du CIC polyphasé présenté à la figure 10 (d) est donné par :

$$G(dB) = 20\log(8) + 20\log\left((16\times1)^5\right) = 138.47 \text{ dB}$$

Les résultats théoriques coïncident bien avec les résultats expérimentaux présentés à la figure 10. Cependant, il faut noter que le gain DC du CIC polyphasé est plus petit que celui du CIC régulier. Ceci est dû au fait que la décimation du CIC polyphasé se fait sur deux étages alors que celle du CIC régulier se fait sur un seul étage. L'équation (3.12) montrait que la largeur binaire maximale des bus de données du CIC décimateur polyphasé est plus courte que celle du CIC régulier de $([N_2 \log_2 R_1] - [N_1 \log_2 R_1])$ bits. Cette diminution de la largeur binaire maximale du CIC polyphasé est due à la diminution de son gain DC. En normalisant le gain DC à 1, les spectres du CIC régulier et du noyau programmable sont présentés à la figure 43.

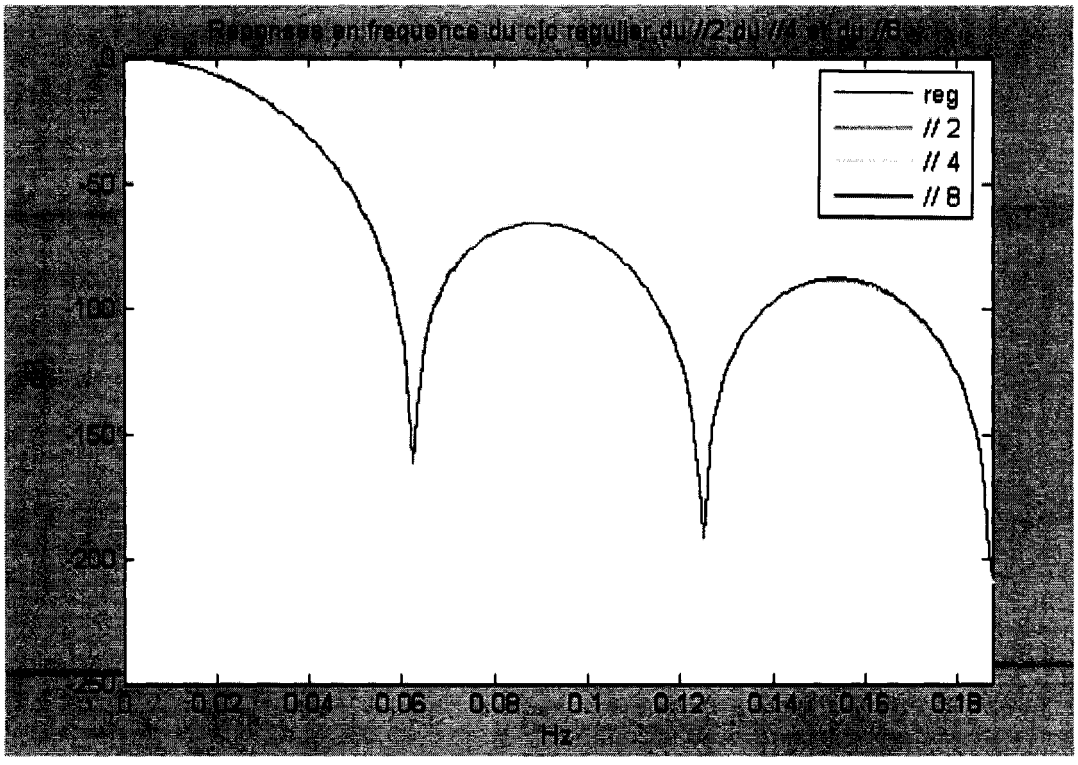


Figure 43 Comparaison des spectres normalisés du CIC régulier et du noyau programmable avec 2, 4 et 8 entrées parallèles.

Finalement, pour terminer la vérification du noyau programmable, les caractéristiques des filtres CIC décimateurs tels que présentés dans l'article de base de Hogenauer sont comparés à ceux du noyau programmable. Le tableau VIII présente la comparaison au niveau de l'atténuation dans la bande passante et le tableau IX la comparaison au niveau de l'atténuation dans la bande image/recouvrement.

Tableau VIII

Atténuation dans la bande passante du CIC régulier et du CIC polyphasé avec 2, 4, et 8 entrées parallèles

Largeur de bande relative à la basse fréquence (f_c)	Filtre CIC	Atténuation (en dB) dans la bande passante à la fréquence (f_c) en fonction de l'ordre du filtre N					
		1	2	3	4	5	6
1/128	CIC_régulier	0.00	0.00	0.00	0.00	0.00	0.01
	CIC_2-phase	0.00	0.00	0.00	0.00	0.00	0.00
	CIC_4-phase	0.00	0.00	0.00	0.00	0.00	0.00
	CIC_8-phase	0.00	0.00	0.00	0.00	0.00	0.00
1/64	CIC_régulier	0.00	0.01	0.01	0.01	0.02	0.02
	CIC_2-phase	0.00	0.00	0.01	0.01	0.01	0.02
	CIC_4-phase	0.00	0.00	0.01	0.01	0.01	0.02
	CIC_8-phase	0.00	0.00	0.01	0.01	0.01	0.02
1/32	CIC_régulier	0.01	0.03	0.04	0.06	0.07	0.08
	CIC_2-phase	0.01	0.02	0.04	0.05	0.07	0.08
	CIC_4-phase	0.01	0.02	0.04	0.05	0.07	0.08
	CIC_8-phase	0.01	0.02	0.04	0.05	0.07	0.08

Tableau VIII (suite)

Largeur de bande relative à la basse fréquence (f_c)	Filtre CIC	Atténuation (en dB) dans la bande passante à la fréquence (f_c) en fonction de l'ordre du filtre N					
		1	2	3	4	5	6
1/16	CIC_régulier	0.06	0.11	0.17	0.22	0.28	0.34
	CIC_2-phase	0.05	0.11	0.16	0.22	0.28	0.33
	CIC_4-phase	0.05	0.11	0.16	0.22	0.28	0.33
	CIC_8-phase	0.05	0.11	0.16	0.22	0.28	0.33
1/8	CIC_régulier	0.22	0.45	0.67	0.90	1.12	1.35
	CIC_2-phase	0.22	0.44	0.67	0.89	1.12	1.34
	CIC_4-phase	0.22	0.45	0.67	0.89	1.12	1.34
	CIC_8-phase	0.22	0.44	0.67	0.89	1.12	1.34
1/4	CIC_régulier	0.91	1.82	2.74	3.65	4.56	5.47
	CIC_2-phase	0.91	1.82	2.73	3.64	4.56	5.47
	CIC_4-phase	0.91	1.82	2.73	3.64	4.56	5.47
	CIC_8-phase	0.91	1.82	2.73	3.64	4.55	5.46

Tableau IX

Atténuation dans la bande image/recouvrement du CIC régulier et du CIC polyphasé avec 2, 4, et 8 entrées parallèles

Délai différentiel (M)	Bande passante relative	Filtre CIC	Atténuation (en dB) dans la bande image/recouvrement à la fréquence f_A					
			1	2	3	4	5	6
1	1/128	CIC_régulier	42.1	84.2	126.2	168.3	210.4	252.5
		CIC_2-phase	42.1	84.2	126.2	168.3	210.4	252.5
		CIC_4-phase	42.1	84.2	126.2	168.3	210.4	252.5
		CIC_8-phase	42.1	84.2	126.2	168.3	210.4	252.5
1	1/64	CIC_régulier	36.0	72.0	108.0	144.0	180.0	215.9
		CIC_2-phase	36.0	72.0	108.0	144.0	180.0	215.9
		CIC_4-phase	36.0	72.0	108.0	144.0	180.0	215.9
		CIC_8-phase	36.0	72.0	108.0	144.0	180.0	215.9
1	1/32	CIC_régulier	29.8	59.7	89.5	119.4	149.2	179.0
		CIC_2-phase	29.8	59.7	89.5	119.4	149.2	179.0
		CIC_4-phase	29.8	59.7	89.5	119.4	149.2	179.0
		CIC_8-phase	29.8	59.7	89.5	119.4	149.2	179.0
1	1/16	CIC_régulier	23.6	47.2	70.7	94.3	117.9	141.5
		CIC_2-phase	23.6	47.2	70.7	94.3	117.9	141.5
		CIC_4-phase	23.6	47.2	70.7	94.3	117.9	141.5
		CIC_8-phase	23.6	47.2	70.7	94.3	117.9	141.5
1	1/8	CIC_régulier	17.1	34.3	51.4	68.5	85.6	102.8
		CIC_2-phase	17.1	34.3	51.4	68.5	85.6	102.8
		CIC_4-phase	17.1	34.3	51.4	68.5	85.6	102.8
		CIC_8-phase	17.1	34.3	51.4	68.5	85.6	102.8

Tableau IX (suite)

Délai différentiel (M)	Bande passante relative	Filtre CIC	Atténuation (en dB) dans la bande image/recouvrement à la fréquence f_A					
			1	2	3	4	5	6
1	1/4	CIC_régulier	10.5	20.9	31.4	41.8	52.3	62.7
		CIC_2-phase	10.5	20.9	31.4	41.8	52.3	62.7
		CIC_4-phase	10.5	20.9	31.4	41.8	52.3	62.7
		CIC_8-phase	10.5	20.9	31.4	41.8	52.3	62.7
2	1/256	CIC_régulier	48.1	96.3	144.4	192.5	240.7	288.8
		CIC_2-phase	48.1	96.3	144.4	192.5	240.7	288.8
		CIC_4-phase	48.1	96.3	144.4	192.5	240.7	288.8
		CIC_8-phase	48.1	96.3	144.4	192.5	240.7	288.8
2	1/128	CIC_régulier	42.1	84.2	126.2	168.3	210.4	252.5
		CIC_2-phase	42.1	84.2	126.2	168.3	210.4	252.5
		CIC_4-phase	42.1	84.2	126.2	168.3	210.4	252.5
		CIC_8-phase	42.1	84.2	126.2	168.3	210.4	252.5
2	1/64	CIC_régulier	36.0	72.0	108.0	144.0	180.0	216.0
		CIC_2-phase	36.0	72.0	108.0	144.0	180.0	216.0
		CIC_4-phase	36.0	72.0	108.0	144.0	180.0	216.0
		CIC_8-phase	36.0	72.0	108.0	144.0	180.0	216.0
2	1/32	CIC_régulier	29.9	59.8	89.6	119.5	149.4	179.3
		CIC_2-phase	29.9	59.8	89.6	119.5	149.4	179.3
		CIC_4-phase	29.9	59.8	89.6	119.5	149.4	179.3
		CIC_8-phase	29.9	59.8	89.6	119.5	149.4	179.3

Tableau IX (suite)

Délai différentiel (M)	Bande passante relative	Filtre CIC	Atténuation (en dB) dans la bande image/recouvrement à la fréquence f_A					
			1	2	3	4	5	6
2	1/16	CIC_régulier	23.7	47.5	71.2	95.0	118.7	142.5
		CIC_2-phase	23.7	47.5	71.2	95.0	118.7	142.5
		CIC_4-phase	23.7	47.5	71.2	95.0	118.7	142.5
		CIC_8-phase	23.7	47.5	71.2	95.0	118.7	142.5
2	1/8	CIC_régulier	17.8	35.6	53.4	71.3	89.1	106.9
		CIC_2-phase	17.8	35.6	53.4	71.3	89.1	106.9
		CIC_4-phase	17.8	35.6	53.4	71.3	89.1	106.9
		CIC_8-phase	17.8	35.6	53.4	71.3	89.1	106.9

Les tableaux VIII et IX montrent que, pour les mêmes spécifications, le noyau programmable possède les mêmes caractéristiques que le CIC régulier. Ceci nous permet de conclure que le noyau programmable fonctionne bien. La comparaison, aux niveaux des ressources utilisées et de la fréquence maximale de fonctionnement du noyau programmable avec les noyaux qui existent sur le marché sera l'objet du chapitre 5.

4.6 Considération pour la synthèse et l'optimisation

Synthétiser un design consiste à traduire une description textuelle d'une fonction en une interconnexion de modules physiques. Décrire une fonction en VHDL pour la rendre correctement synthétisable n'est pas trivial et doit répondre à une certaine rigueur si on veut garantir une certaine qualité de résultat car le design synthétisable va se transformer en silicium et soumis à toutes sortes d'influences externes (températures, tension,

humidité .....). Il est évident que ce n'est pas parce qu'une fonction peut être décrite en VHDL qu'elle sera synthétisable.

4.6.1 Choix du Package

Le VHDL permet de définir un nombre illimité de types de signaux, avec leur arithmétique et leur logique propre. Cependant, dans l'intérêt de la clarté du code, la logique standard, celle définie dans le package IEEE 1164 est priorisée. Ce package contient la description du type élémentaire des signaux logiques.

4.6.2 Opérateurs

Les synthétiseurs fournissent un certain nombre de packages qui contiennent souvent une description de fonctions arithmétiques plus ou moins complexes. Ces modules sont généralement optimisés pour une technologie particulière avec des critères tels que la surface, la consommation ou la vitesse. Ainsi, pour garder la portabilité du code VHDL, qui est un des buts de notre travail, les fonctions linéaires (additions, soustractions) sont implémentées avec le package *IEEE.std_logic_signed*.

4.6.3 Généricité

La généricité permet de décrire un code paramétrable dont la configuration peut être choisie soit au moment de la synthèse, soit dans l'instanciation des composants. Cette possibilité, bien que formidable en théorie, crée souvent des problèmes lors de la synthèse. Les synthétiseurs ne comprenant souvent qu'un sous-ensemble du VHDL, il arrive parfois, surtout pour les outils bon marché, que ceux-ci refusent le code contenant des paramètres génériques. Les outils plus évolués, quand à eux, acceptent normalement la généricité mais celle-ci demande des manipulations supplémentaires. Ainsi, pour

limiter les paramètres génériques dans ce travail, les fonctionnalités simples sont décrites de manière directe (c'est-à-dire sans généricité).

4.6.4 Code concurrent ou séquentiel

Le VHDL permet de décrire le code de deux manières distinctes : de manière concurrente ou séquentielle. Le code séquentiel correspond aux *PROCESS* et le code concurrent à la logique hors des *PROCESS*. Le code séquentiel a tendance à créer des *PROCESS* de plus en plus gros avec des *if* et des *case* imbriqués, rendant le code moins robuste et illisible. De plus, cette complexité logique engendre souvent des difficultés lors de la synthèse car l'outil étant incapable de regrouper efficacement les fonctions. Étant donné que les fonctionnalités décrites de manière concurrente sont souvent découpées en petites fonctions élémentaires plus proches du matériel [Thierry Schneider], le code concurrent est utilisé là où c'est possible.

4.6.5 Optimisation du timing

L'optimisation du *timing* constitue souvent le premier pas pour augmenter les performances du circuit final. Pour cela, on décrit les contraintes temporelles du circuit pour que le synthétiseur sache où concentrer son effort. Étant donné que le design est synchrone, en spécifiant une contrainte de fréquence, le synthétiseur dispose du temps maximum à allouer à la logique combinatoire.

4.6.6 Partage de ressources

Un design VHDL comprend généralement trois parties différentes : des mémoires, des fonctions combinatoires et des fonctions arithmétiques. Dans ce dernier cas, on peut économiser beaucoup de surface et de consommation si on parvient à partager des ressources matérielles entre plusieurs processus. Le gain en surface obtenu en optimisant

manuellement une fonction, par rapport à une optimisation automatique, ne justifie pas une telle dépense de temps. Les synthétiseurs effectuent efficacement cette tâche. Par contre, les générations actuelles d'outils de synthèse rencontrent des difficultés pour détecter certaines duplications de fonction. Pour éviter ce cas de figure, toutes les fonctions arithmétiques sont décrites de façon explicite.

4.7 Conclusion

La réalisation et la vérification du noyau programmable ont été l'objet de ce chapitre. Ainsi, le code VHDL est divisé en trois grands *PROCESS* : un pour le FIR polyphasé, un pour le CIC régulier et un pour le contrôle. Chaque *PROCESS* est vérifié par un *testbench* spécial qui permet de l'isoler. Le noyau programmable au complet est vérifié, premièrement, au haut niveau en comparant ses spectres d'entrée et de sortie pour voir si le code VHDL fonctionne bien. En prenant le CIC régulier de Xilinx qui existe sur Matlab/Simulink comme référence, sa réponse en fréquence est comparée à celles du noyau programmable pour 2, 4 et 8 entrées parallèles. Pour terminer la vérification, les caractéristiques de base du CIC régulier présenté par Hogenauer sont comparés à ceux du noyau programmable. Cette dernière vérification nous a permis de conclure que le noyau programmable fonctionne bien. Pour terminer, étant donné que l'un des buts finaux de ce travail est d'avoir un noyau synthétisable, quelques considérations pour la synthèse ont été présentées.