Mastère de sciences et technologies

MENTION INFORMATIQUE
2014 – 2015

Spécialité

SYSTÈMES ELECTRONIQUES ET SYSTÈMES INFORMATIQUES

SYNTHÈSE DE CIRCUITS UTILISANT L'ARITHMÉTIQUE VIRGULE FIXE

RAPPORT DE STAGE

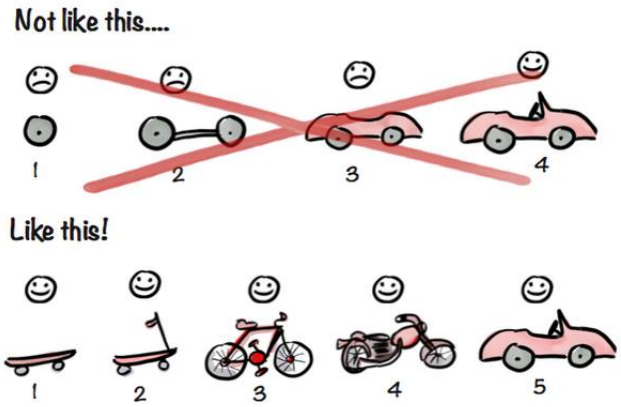
date de soutenance : 29 septembre 2015

Accueil : **Laboratoire d'Informatique de Paris 6**

PRÉSENTÉ PAR
MAMINIONJA RAVOSON

ENCADRANTS
ROSELYNE CHOTIN-AVOT (CIAN)
THIBAULT HILAIRE (PEQUAN)

"Build a working system first, then refine it with more knowledge later."



Remerciements

Je tiens d'abord à remercier Marie-Minerve Louërat, Responsable du département "Systèmes Embarqués sur Puce" et Habib Mehrez, Responsable de l'équipe CIAN - Circuits Intégrés Numériques et Analogiques, de m'avoir permis de faire ce stage dans leurs équipes.

Je voudrai remercier Jean-Lou Desbarbieux et Julien Denoulet, Résponsables du Master SESI à l'UPMC, pour ces 2 ans de formation dans le domaine de la conception des systèmes embarqués. Elle a été très profitable et m'aiderai beaucoup pour la suite.

J'adresse mes vifs Remerciements à mes encadrants de stage, Roselyne Chotin-Avot (Eq CIAN /dpt SOC) et Thibault Hilaire (Eq PEQUAN / Dpt CalSci) pour leur grande confiance, leurs conseils, les discussions scientifiques très enrichissantes ainsi que les temps passés à essayer de résoudre ensemble les problèmes. J'ai énormément appris. Merci.

Et comment oublier Benoit Lopez pour son explication concise et claire sur FiPoGen et Anastasia Volkova notamment pour son aide sur le SIF, le filtre LWDF et l'algorithme DFS [1].

Un grand Merci à ma famille et mes amis pour leur soutien inconditionnel.

1. Deep First Search, un algorithme de recherche dans un graphe qui permet aussi de faire un tri topologique

Table des matières

Table des figures

Introduction

On développe de plus en plus de système électronique embarqué, devenus nécessaires dans différents domaines comme l'automobile, la robotique, la santé, le traitement de signal, etc. Ceci pour augmenter la performance et la sécurité. L'un des éléments essentiels dans ces systèmes est le contrôleur qui génère la loi de commande à partir de la consigne et du comportement réel du système. A l'intérieur et autour de ce contrôleur, on trouve les filtres qui font des opérations mathématiques bien définies sur les signaux.

Dans notre cas, on s'intéresse à un type particulier de filtre : les filtres numériques linéaires invariants dans le temps aka L.T.I [2]. Ce sont des filtres ou le comportement reste le même à chaque échantillon, et on effectue seulement des opérations linéaires (addition et multiplication par des constantes).

L'implémentation des filtres nécessitent de considérer le cout (taille des circuits), la fiabilité (respect des spécifications), le temps de développement et surtout la représentation des coefficients et les variables dans les calculs. Il y a 2 manières de représenter ces nombres : soit en virgule flottante aka FIP (FloatIng Point) et soit en virgule fixe aka FxP (Fixed Point). L'utilisation du FxP est généralisée dans les systèmes embarqués car elle utilise uniquement des unités de calcul en entiers, donc plus petite et plus rapide. Mais elle offre moins de précision et nécessite un travail plus important de la part du développeur (Dans le FIP, tous le calcul est géré par les opérateurs).

Dans ce travail, on désire favoriser l'utilisation des FxP, dans l'implémentation des filtres numériques, en améliorant ou du moins en estimant la précision, et en réduisant le travail du développeur. L'objectif du stage est donc d'avoir un outil - Fxp_synthesizer - qui génère la description matérielle du circuit à partir d'un algorithme de haut niveau tel un diagramme de blocs Simulink. Pour cela, on utilise FiPoGen [3] pour fixer la taille des variables du filtre tout en évaluant l'erreur résultante. La bibliothèque Stratus [4], quant à elle , permet de générer le circuit correspondant à l'algorithme FxP.

2. Linear Time Invariant Filter
3. Fixed Point code Generator : un outil développé par l'équipe PEQUAN du laboratoire LIP6
4. Générateur de code VHDL, développé par l'équipe CIAN du laboratoire LIP6

Chapitre 1

Présentation du stage de recherche

1.1 Le laboratoire LIP6 dans l'Université Pierre et Marie Curie

le LIP6 [1] est un UMR [2] entre l'UPMC [3] et le CNRS [4]. C'est un laboratoire de recherche en informatique se consacrant à la modélisation et la résolution de problèmes fondamentaux motivés par les applications, ainsi qu'à la mise en oeuvre et la validation des solutions au travers de partenariats académiques et industriels. [5]

Les activité du laboratoire sont groupées en 6 départements :

— Calcul Scientifique
— DEcision, Systèmes Intelligents Recherche opérationnelle
— Données et Apprentissage Artificiel
— Réseaux et Systèmes
— Systèmes Complexes
— Systèmes Embarqués sur Puce

Chaque département a plusieurs équipes.

1.1.1 L'équipe CIAN du département Systèmes Embarqués sur Puce

CIAN signifie Circuits Intégrés Numériques et Analogiques. Leur axe de recherche porte sur les méthodes de conception, et la conception elle-même, des composants numériques et analogiques pouvant être intégrés dans des systèmes sur puce. [6]
Mr. Habib Mehrez est le responsable, Site Jussieu 24-25/310.

1. Laboratoire d'informatique de Paris 6
2. Unité Mixte de Recherche
3. Université Pierre et Marie Curie
4. Centre National de la Recherche Scientifique
5. www.lip6.fr
6. http ://www.lip6.fr/recherche/team.php ?id=940

1.1.2 L'équipe PEQUAN du département Calcul scientifique

PEQUAN signifie PErformance et QUalité des Algorithmes Numériques. Leurs domaines de recherche sont liés sur l'arithmétique des ordinateurs et au calcul scientifique. [7]

Le développement d'algorithmes numériques en virgule flottante ou en virgule fixe avec leur validation fait partie de leur domaine de compétences.
Mr. Stef Graillat est le responsable, Site Jussieu 26-00/313.

1.2 Le stage Recherche

Ce stage permet de mettre en œuvre les acquis pendant la formation de Master2 et contribue à sa validation. Le stage se déroule dans le local du laboratoire. Ce document constitue le rapport de réalisation qui détaille les travaux effectués pendant le stage ainsi que les résultats obtenus et leur validation. Il reprend aussi la spécification du sujet.

7. http ://www.lip6.fr/recherche/team.php ?id=120

Chapitre 2

Le sujet

2.1 Contexte de l'étude

Ce stage est lié au projet FxPSynthesis, qui est un projet Action initiative du LIP6. Ce projet est le fruit d'une collaboration entre les 2 équipes CIAN et PEQUAN du laboratoire. Les 2 équipes ont chacun un outil (FiPoGen et Stratus)qui peut être mis en commun pour former un flot complet de conception et de réalisation d'algorithme de filtrage et/ou de contrôle numérique.

L'objectif du stage, qui fait partie du projet, est d'avoir un outil de synthèse de circuit à partir d'un algorithme de haut niveau.

2.2 Définition et analyse du problème

2.2.1 Le sujet

Pendant le stage, on doit développer un outil de synthèse de haut-niveau pour les filtres linéaires, en utilisant l'arithmétique virgule fixe. On part d'une description très haut niveau, un diagramme Matlab/Simulink pour arriver à une implémentation VHDL du filtre. Les différentes étapes, et les formats pendant cette synthèse sont représentés par le schéma de la figure 2.1.

Essentiellement, le stage a 2 objectifs :

— **Concevoir un module** qui permet à FiPoGen d'importer des diagrammes de blocs Simulink

— Développer un module à l'aide de Stratus pour **génerer l'architecture matérielle** correspondant au graphe de calcul généré par FiPoGen

— Il en découle donc des travaux d'**intégration** des modules avec FipoGen et Stratus pour former un flot de conception "complet"

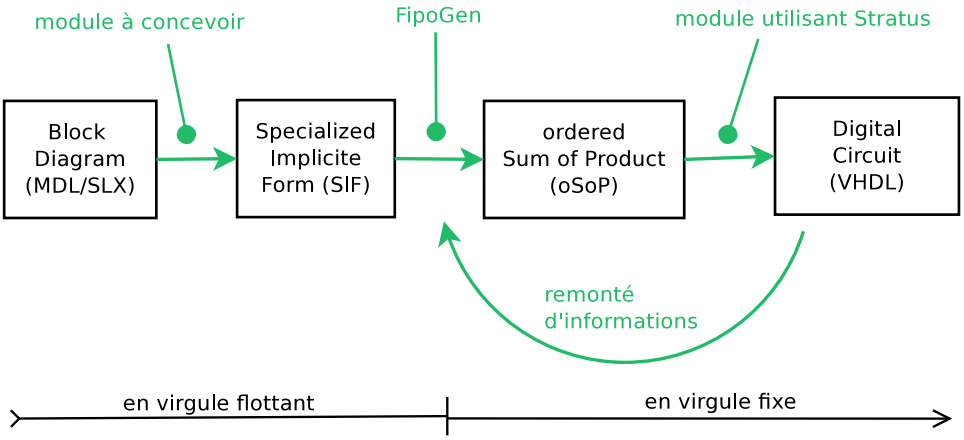


FIGURE 2.1 – les étapes de la synthèse

2.2.2 Etat de l'art

Il existe de nombreux outils de conception et de synthèse de filtre. Un exemple très connu est le FDAtool de Matlab. Ils fonctionnent par simulation/raffinements, donc demandent beaucoup de temps et d'expertise au concepteur. En plus, le résultat n'est pas toujours très satisfaisant.

On a à notre disposition des bibliothèques pour développer un nouvel outil :

— **FiPoGen** (Fixed Point Generator) : un outil qui permet de transformer un algorithme en du code virgule fixe déjà spécifié

— **Stratus** : qui permet de décrire des composants matériels à l'aide de ces générateurs paramétrables

Les travaux pendant le stage peuvent être vus comme un développement et une intégration dans FiPoGen de 2 modules en entrée et en sortie. Le premier est un nouveau module à concevoir, le dernier se base sur Stratus. La figure 2.2 montre l'enchaînement des différents modules.

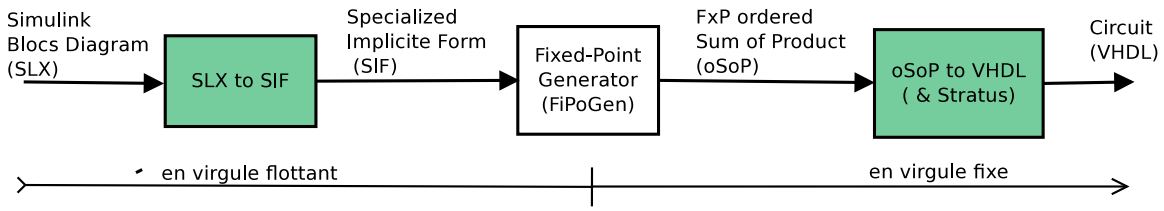


FIGURE 2.2 – les étapes de la synthèse

2.2.3 Les problèmes à résoudre

1. Au niveau du module d'entrée à FiPoGen (SLX –> SIF), Il faut :
   — Faire un peu de **reverse-engineering** car les fichiers de diagramme simulink (MDL, SLX) ne sont pas bien documentés, donc il faut
   — Acquérir un minimum de compréhension sur **l'algèbre linéaire, la théorie des graphes et les filtres numériques** pour comprendre et pouvoir faire la transformation en SIF du filtre.
   — Savoir **extraire les paramètres** utiles à partir du fichier Simulink. Notons que le nouveau format de fichier SLX de simulink est une un fichier compressé, au format OPC [1], qui contient plusieurs fichiers XML [2].
   — Comprendre la représentation en **S.I.F** des systèmes linéaires
   — Dévéloppement d'un **algorithme** qui permet d'établir la représentation d'état spécialisé (SIF) d'un système décrit par son diagramme de bloc.
   — Eventuellement, pouvoir génerer un Diagramme de Block à partir d'un S.I.F.

2. Au niveau du module de sortie, utilisant Stratus (oSoP –> VHDL), Il est nécessaire de :
   — Avoir une notion sur l' **arithmétique virgule fixe**
   — Comprendre un peu l'outil **FiPoGen**, en particulier la **classe oSoP**
   — Savoir utiliser efficacement les **Générateurs de Stratus**
   — Trouver un mécanisme pour **valider** que le code vhdl généré correspond bien à l'oSoP

3. Concernant **Intégration** de la chaine complète :
   — Les modules doivent être **compatible** et forment une chaine uniforme avec l'existant, **en Python**
   — Il sera nécessaire de développer des **"glue logic"** (des bouts de code qui permet d'interfacer) pour mettre ensemble les modules
   — On doit **bien documenter** le travail pour faciliter la reprise par d'autre à la fin du stage. Cela comprend l'écriture d'un code lisible et réutilisable.

1. Open Package Convention
2. eXtensible Markup Language

Chapitre 3

Principe de la solution envisagée

Dans le développement de la solution, on se garde de faire une conception facilement extensible à d'autres cas. Donc il faut bien choisir la manière de structurer les données et de faire les traitements.

Pour le module **Simulink vers SIF** On va :
— Faire du "reverse enginering" pour déduire la structure du fichier simulink. Il existe 2 format de fichier en Simulink : MDL (fichier texte) et SLX (fichier compressé). Pour notre implémentation, on a choisit le nouveau format SLX.
— Choisir Python comme language de développement
— Suivre Un développement incrémental
— Partir d'un diagramme très simple pour tester les différentes partie du code
— Faire à la main puis comparer avec le résultat obtenu avec le code
— Etablir des moyens de test pour les fonctionnalités essentiels
— Tester avec des diagrammes plus conséquents

Pour le module **oSoP vers VHDL** On va :
— S'inspirer des codes existant pour avoir une idée
— Python comme language de développement
— Comprendre en détails les générateurs de Startus utile pour notre cas
— Comprendre l'outil Stratus (le modèle objet associé) pour développer
— Faire des tests de génération de petits circuits
— choisir et implémenter un algo efficace pour la généartion oSoP vers VHDL
— Générer un test bench pour vérifier avec FiPoGen le vhdl obtenu

3.1 Identification des tâches à accomplir

La solution retenue qui va être mise en oeuvre
— **Prendre en main** le projet, le sujet du stage
— Maitriser l'utilisation des **outils**, python en particulier
— Etudier la structure des **fichiers** matlab (SLX)
— Bien comprendre le **SIF** et les filtres numériques
— **Concevoir** le module SLX vers SIF : c'est à dire créer un module qui permet à FiPooGen de prendre en entrée un diagramme en blocs Simulink
— Développer le module oSoP vers VHDL : Explorer les possibilités offert par stratus, puis utiliser l'API Stratus pour générer la description d'un circuit à partir du graphe de calcul généré par FiPoGen.
— Construire un jeux de tests (ensemble d'algorithme à réaliser) qui permet de valider les différentes fonctionnalités de notre chaine, puis tester avec des exemples, **déboguer** et optimiser. Après, il faut fixer les limites des solutions developpés.
— **Intégration** des modules
— **Rédaction** du rapport et des autres documentations. Celà inclus bien sur l' amélioration de la lisibilité et de la réutilisation des codes sources.

Chaque développement se compose des 3 étapes suivantes :
— Etude de l'existant
— Prototypage
— Test et validation
— Analyse des résultats et correction
— Intégration de la fonctionnalité

3.2 Définition de la procédure de recette

Les procédures décrits dans ce chapitre permet de valider l'aptitude de notre solution à répondre aux problèmes posés.

3.2.1 Au niveau de la chaine complète

A partir d'un diagramme de blocs en Simulink, on doit avoir un code VHDL correspondant qui réalise la même opération en virgule fixe. On vérifiera par simulation, si tout est Ok, on devrait avoir le même résultat de simulation à une petite erreur prêt, fixée par la contrainte en entrée de FiPogen.

3.2.2 Au niveau de chaque fonctionnalité

On testera soit avec des exemples déjà traité par les 2 équipes ou des exemples bien connus, soit par simulation avec des stimulus aléatoire. Notre module devrait donner le même résultat.

Pour la partie diagramme Simulink vers SIF

- essentiellement **par comparaison** au résultat d'interprétation manuelle
- comparaison au résultat pour le filtre LWDF déjà été fait "manuellement" par un membre de l'équipe PEQUAN
  On doit avoir le **même résultat**
- petits diagrammes simulink pour tester les fonctionnalités internes

Pour la partie utilisant Stratus : oSoP vers VHDL

- FiPoGen peut générer un **code C** virgule fixe correspondant à l'oSoP
- Nous, on génère du code **VHDL** à partir de l'oSoP
- on verifiera par **double simulation**

On génère aléatoirement les mêmes stimulus pour les 2 codes puis on lance les 2 simulations, ce qui devrait nous donner le même résultat.

3.2.3 Au niveau du développement

Écrire des tests unitaires pour les méthodes de classe essentielles ainsi que pour les fonctions. Faire des tests au fur et à mesure du développement et d'ajout de fonctionnalités. Tous les tests devront passer.

Chapitre 4

Mise en œuvre

4.1 Génération du S.I.F à partir du schéma-bloc Simulink

4.1.1 L'idée

L'objectif ici est de déduire les matrices (J, K, L, M, N, P, Q, R, S) du S.I.F à partir d'un schéma-blocs représentant un système linéaire, de manière automatique.

4.1.2 Le diagramme de bloc Matlab/Simulink

Un diagramme de bloc ou schéma-bloc est une représentation graphique simplifié d'un système plus ou moins complexe. Il est composé de blocs connectés par des lignes qui montre l'interaction entre-eux. Il permet d'exprimer la structure et le flux d'un système pour juste montrer le concept sans entrer dans les détails d'implémentation.

Le fichier Simulink

L'avantage d'utiliser Simulink est que c'est un standard dans l'industrie. Par contre, c'est un format très peu documenté. Dans Simulink, on a 2 formats pour stocker les schéma-blocs :

— MDL : ancienne format
— SLX : nouvelle format de fichier, à partir de la r2012a de Matlab

Dans ce travail, on a choisi de traiter le format de fichier SLX.

Structure d'un fichier Simulink SLX Le fichier SLX est le nouveau format de fichier pour les diagrammes de blocs dans Matlab/Simulink. C'est un conteneur de fichiers XML et non XML, conforme à la norme OPC [1].

La figure 4.1 montre un extrait des informations intéressant sur le contenu du fichier.

1. Open Package Convetion

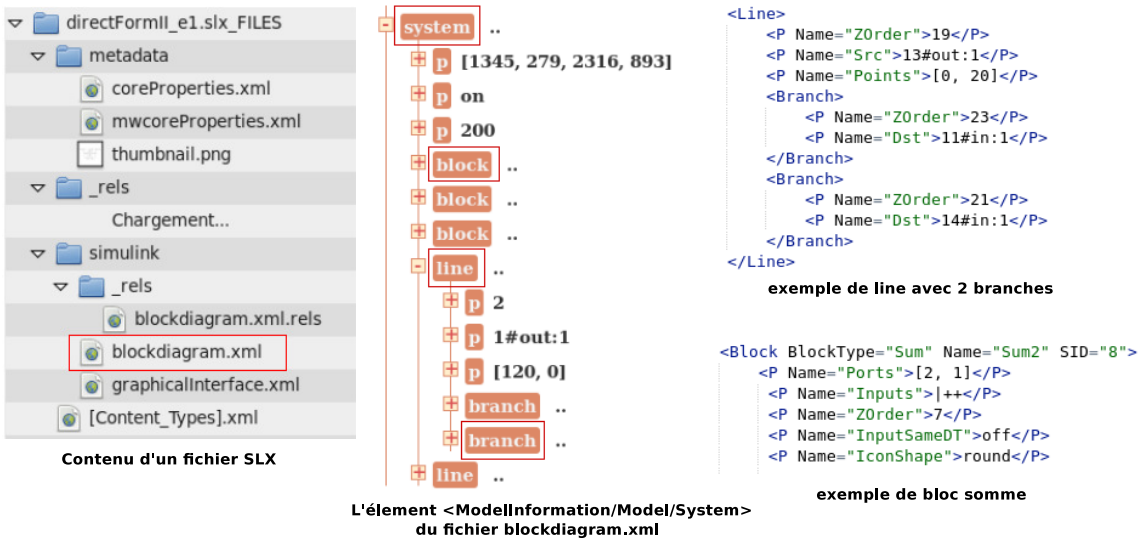


FIGURE 4.1 – extrait du contenu d'un fichier Simulink SLX

4.1.3 La Forme implicite - S.I.F

[1, 2, 3] Il faut bien comprendre le format de représentation SIF pour bien faire la déduction à partir d'un diagramme de blocs.
La représentation d'état est très utilisée pour étudier un filtre, mais elle n'est pas complète et a quelques limitations :

— difficile d'analyser l'effet d'arrondi sur un coefficient particulier
— Absence de variable intermédiaire utile pour l'expression de certaines réalisations

La forme implicite spécialisé (SIF) est une réalisation directe, plus précis que la représentation d'état, sous la forme représenté par la figure4.2[1][4] :

Elle a comme avantages :

— L'analyse facile des effets de la précision finie
— La description de haut niveau plus généralisé et plus précis

4.1.4 Méthode pour déduire le SIF à partir d'un Diagramme de Blocs

On a donc établit l'algorithme suivant pour convertir le schéma-bloc en S.I.F :

— Décompresser le fichier SLX pour extraire le fichier *blocdiagram.xml*
— Parser le fichier pour extraire la liste des blocs et la liste des lignes (qui indiques les inter-connexions)
— Aplatir le design s'il y a des sous-systèmes
— Labelliser chaque blocs
— Identifier les blocs en entrée de chaque bloc, cela donne l'équation au niveau de chaque bloc
— Regrouper si possible les blocs sommes

$$\begin{pmatrix} J & 0 & 0 \\ -K & I_n & 0 \\ -L & 0 & I_p \end{pmatrix} \begin{pmatrix} T(k+1) \\ X(k+1) \\ Y(k) \end{pmatrix} = \begin{pmatrix} 0 & M & N \\ 0 & P & Q \\ 0 & R & S \end{pmatrix} \begin{pmatrix} T(k) \\ X(k) \\ U(k) \end{pmatrix}$$

$$\begin{cases} \boldsymbol{Jt}(k+1) = & \boldsymbol{Mx}(k) + \boldsymbol{N}u(k) \\ \boldsymbol{x}(k+1) = \boldsymbol{Kt}(k+1) + & \boldsymbol{Px}(k) + \boldsymbol{Q}u(k) \\ y(k) = \boldsymbol{Lt}(k+1) + & \boldsymbol{Rx}(k) + \boldsymbol{S}u(k) \end{cases}$$

t(k+1) : résultat intermédiaire
y(k) : sortie
x(k+1) : état suivant
u(k) : entrée

$$\boldsymbol{Z} \triangleq \begin{pmatrix} -\boldsymbol{J} & \boldsymbol{M} & \boldsymbol{N} \\ \boldsymbol{K} & \boldsymbol{P} & \boldsymbol{Q} \\ \boldsymbol{L} & \boldsymbol{R} & \boldsymbol{S} \end{pmatrix}$$

FIGURE 4.2 – Représentation d'un S.I.F.

— Identifier les équations avec celles du SIF, donc les coefficients formeront les matrices du SIF
— Rendre triangulaire inférieure la matrice J et réorganiser les matrices concernées : M, N, K, L

Concernant le schéma-bloc qu'on traite, il faut noter que :
— Chaque bloc simulink est identifié par un numéro unique appelé ***SID***
— On a que trois types de blocs : *Sum, Gain, Delay*
— Pas de retour, seulement par les blocs 'Delay'

4.1.5 Implémentation - le module SLX2SIF

Il suffit de lui donner un fichier Simulink au format SLX, contenant le schéma-bloc pour un système linéaire, et il donne en sortie les matrices du SIF avec les variables liées (t, x, u, y).

[schéma]

Les caractéristiques du module

Pour l'état actuel de notre module, on peut traiter 6 types de blocs Simulinnk :
— Gain
— Delay
— Sum
— SubSystem

— Inport
— Outport

Pour chaque bloc, les paramètres essentiels sont :
— BlockType
— Name
— SID Chaque bloc a ses paramètres spécifiques représentés par le tableau 4.1 :

BlockType	**Paramètres**	**Description**
Delay	DelayLength	Description
Sum	Inputs	Mode : + ou
Sum	Ports	[in, out] : nombre E/S
Gain	Gain	
SubSystem	Ports	[in, out] : nombre E/S
SubSystem	System	

TABLE 4.1 – Les paramètres des blocs Simulink

Notons que notre module *SLX2SIF* peut gérer un système MIMO. [2]

4.2 Génération de la liste des oSoP à partir du SIF

La forme implicite spécialisée aka S.I.F [3] permet de représenter efficacement un filtre linéaire. Pour l'implémenter réellement, il faut choisir la nature de la cible : logicielle ou matérielle. [2]

Dans le cas de l'implémentation logicielle, la taille des variables, constantes et opérateurs est fixée par la cible. Par contre, pour l'implémentation matérielle, les largeurs peuvent varier selon la plage des valeurs que peuvent contenir chaque variable ainsi que la précision voulue.

4.2.1 Les calculs nécessaires au préalables

Déduction du State-Space équivalente

Pour mieux caractériser le système, on souhaite donc avoir la représentation d'état équivalent au S.IF. On peut aisément calculer ça à partir des équations du SIF.

$$\begin{aligned} J.t(k+1) &= M.x(k) + N.u(k) \\ x(k+1) &= K.t(k+1) + P.x(k) + Q.u(k) \\ y(k) &= L.t(k+1) + R.x(k) + S.u(k) \end{aligned}$$

2. Multi-Inputs Multi-Outputs
3. Specialized Implicite Form

Le principe consiste à intégrer les calculs intermédiaires dans l'équation d'état et de sortie. Ainsi, on obtient :

$$A = K.J^{-1}.M + P$$
$$B = K.J^{-1}.N + Q$$
$$C = L.J^{-1}.M + R$$
$$D = L.J^{-1}.N + S$$

Calcul de Hu

La fonction de transfert Hu sert à évaluer quel effet a l'entrée sur les variables .

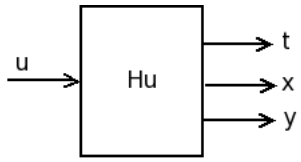


FIGURE 4.3 – La fonction de transfert Hu

Pour le calculer, on part toujours du S.I.F qui décrit notre système. On peut déduire facilement que :

$$HuA = A$$
$$HuB = B$$
$$HuC = [J^{-1}.MAC]$$
$$HuD = [J^{-1}.NAD]$$

Calcul de He

He sert à voir l'effet des erreurs accumulées par les variables sur la sortie.

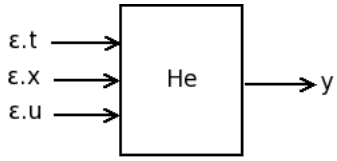


FIGURE 4.4 – La fonction de transfert He

On démontre facilement que :

$$HeA = A$$
$$HeB = [K.J^{-1}Id0]$$
$$HeC = C$$
$$HeD = [L.J^{-1}0Id]$$

Calcul de DC-Gain pour un State-Space

Le DC-G est le gain en mode statique ou la fréquence du signal est nulle. Le gain DC-G en temps discret est la valeur de la fonction de transfert lorsque $z = 1$
Pour les modèles en représentation d'états avec les matrices (A, B, C, D) :

$$G_{DC} = D + C.(I - A)^{-1}.B$$

Calcul du WCPG - Worst Case Peak Gain - pour un State-Space

Le W.C.P.G est la plus grande valeur de la sortie pour toute valeur possible de l'entrée.[5] Pour un système modélisé dans la représentation d'états, on a le définit comme suit :

$$WCPG = |D| + \sum_{k=0}^{\infty} |C.A^k.B|$$

4.2.2 FiPoGen

FiPoGen est un outil développé dans le cadre de la thèse de Benoit Lopez et du projet ANR DEFIS (Design of fixed-point embedded systems, 2011-2014) permettant de transformer un algorithme en un graphe d'opérations virgule fixe, avec calcul des erreurs commises, optimisations des largeurs des opérateurs (sous contrainte d'erreur de sortie). Dans sa version actuelle, FiPoGen est principalement dédié à l'implantation des filtres linéaires récursifs.

Les problèmes Liés à la précision Finie

[2, 3]Les filtres numériques sont toujours implémentés dans une précision finie car les calculateurs ont un nombre de bits limités pour représenter les nombres et faire les calculs.
Cela introduit deux effets :

- le bruit d'arrondi sur les variables - round-off noise
- la dégradation de performance due à l'arrondi des coefficients - coefficients sensitivity

Il est donc nécessaire de s'assurer que ces effets n'entraîne pas une dégradation considérable sur la performance du filtre.
Ces effets dépendent du :

- format d'arithmétique choisi (FIP ou FxP)
- la longueur de mots pour représenter les nombres
- la réalisation choisie (structure du filtre)

Les fonctionnalités essentiels de FiPoGen

FiPoGen est une sorte de boite à outils pour générer du code en virgule fixe. Il permet de :

- choisir la meilleur réalisation
- Optimiser la largeur de chaque variable en fonction de l'erreur acceptée sur la sortie
- Générer un graphe de calcul en FxP pour l'algorithme en entrée en S.I.F

4.2.3 Le flot pour la génération

La figure 4.5 représente les étapes et les entrées nécessaires pour générer pour générer la liste des oSoP pour un schéma bloc donné.

Dans un fichier Bonmin.opt, il faut spécifier le temps limite de résolution ainsi que l'algorithme à utiliser.

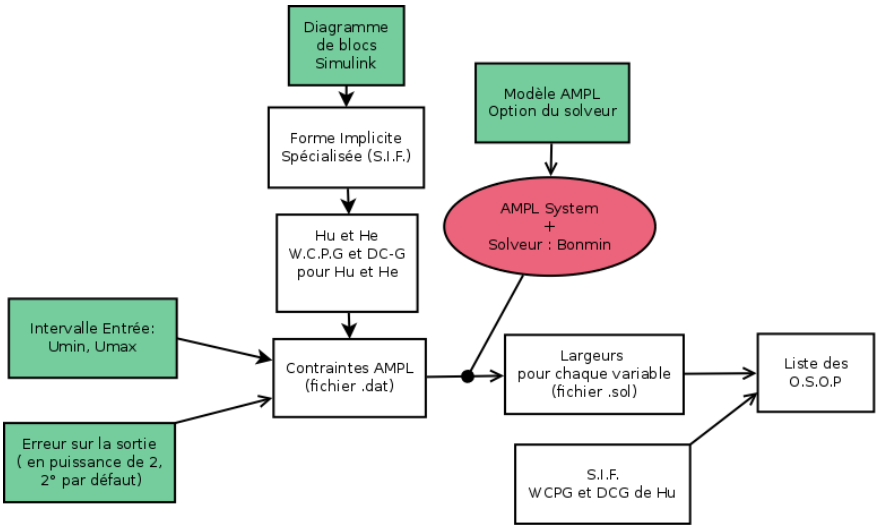


FIGURE 4.5 – Le flot pour la génération de la liste d'oSoP

4.3 Le module oSoP vers VHDL

Avant tout développement il est nécessaire de comprendre l'implémentation des oSoP dans FiPoGen ainsi que l'outil Stratus[6][7] avec le modèle objet associé.

4.3.1 l'oSoP

L'oSoP(ordered Sum of Product) est un arbre de calcul qui précise dans quel ordre faire les sommes de produits. Notons que pour les filtres linéaires, pour une passe, on a que 2 types opérations : la somme et le produit par une constante.

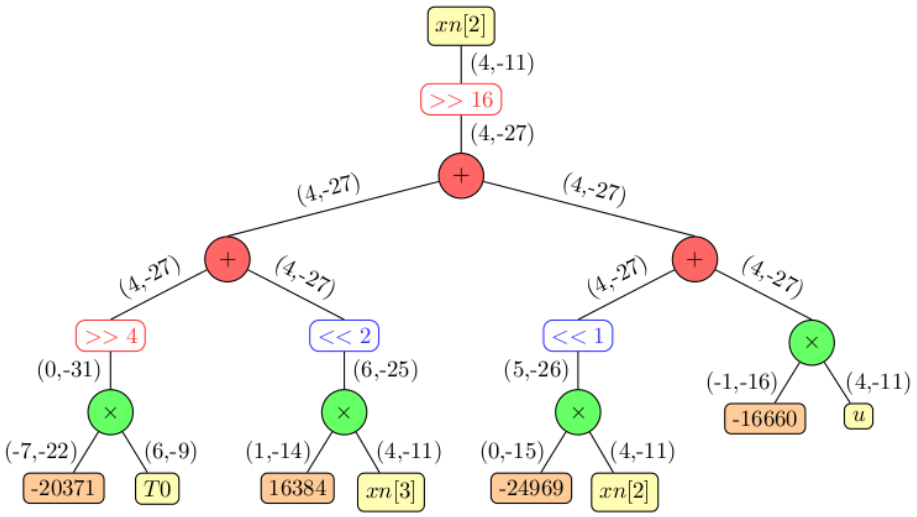


FIGURE 4.6 – représentation graphique d'un oSoP

L'outil FipoGen, permet d'exporter en latex, une visualisation graphique du graphe des opérations.

4.3.2 Stratus et Les générateurs utilisés

Stratus peut être vue comme une bibliothèque de Générateur de composants paramétrable, basé sur Python, qui génère à la sortie du code VHDL synthétisable. Il a des méthodes et fonctions qui permet de décrire les éléments du circuit. On peut donc utiliser les structures de contrôles (boucle, test), les structures de données (variables, listes, dictionnaires, etc) et les fonctions de Python pour gérer les objets VLSI.

Notons que les filtres linéaires n'utilisent que 3 opérateurs, à savoir :

— l'addition
— la multiplication par une constante
— le délai

Ceux-là correspond respectivement dans le circuit à des additionneurs, multiplicateurs par une constante et des registres.

Dans nos expérimentation, on utiliser le modèle ***Slansky*** pour l'additionneur et le modèle par défaut ***Wallace*** pour le multiplieur.
On aurait peut utiliser les fonctions *Evaltime()* et *EvalArea()* pour évaluer le temps de transition et la surface occupé par chaque modèle.

4.3.3 Méthodes

Le principe est de parcourir l'oSoP et d'instancier le générateur Stratus correspondant avec les signaux de connexions nécessaires. On traite donc le graphe récursivement

Pour le circuit d'un oSoP, les étapes sont :

— Créer un modèle Stratus
— Créer le signal de sortie
— Créer un signal intermédiaire qui va recevoir le résultat de l'additionneur final
— Appeler la fonction qui traite l'additionneur
— Connecter la sortie avec le décalage nécessaire

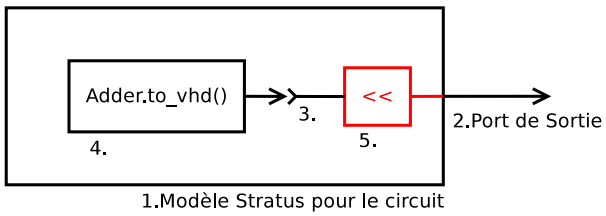


FIGURE 4.7 – Méthodes pour générer le circuit correspondant à un oSoP

A chaque fois qu'on rencontre un additionneur, il faut :

— Créer 3 signaux
— Instancier le générateur d'additionneur Stratus
— Créer 2 signaux intermédiaire qui va recevoir le résultat des opérandes
— Appeler la fonction qui traite les opérandes
— Connecter les 2 signaux avec le décalage nécessaire
— Retourner la sortie

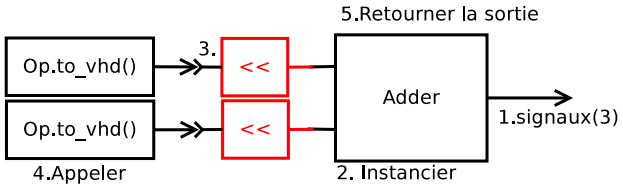


FIGURE 4.8 – Méthodes pour générer le circuit pour un additionneur

A chaque fois qu'on rencontre un additionneur, il faut :
— Créer un signal d'entrée
— Instancier le générateur de multiplicateur de Stratus
— instancier un signal pour connecter la sortie
— Retourner la sortie

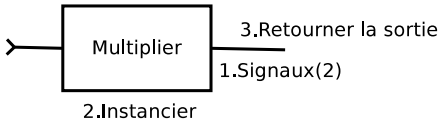


FIGURE 4.9 – Méthodes pour générer le circuit pour un multiplicateur

4.3.4 Implémentation

La figure reffig :osop2vhdl montre le diagramme de classe simplifié du module oSoP vers VHDL.

Stratus simplifie beaucoup la génération de code VHDL. On a déjà des générateurs pour chaque opération. La production de code VHDL se fait en parcourant l'oSoP).
Et pendant le parcours, à chaque fois qu'on rencontre une opération, on instancie le générateur correspondant avec les nets qui permet de le connecter.

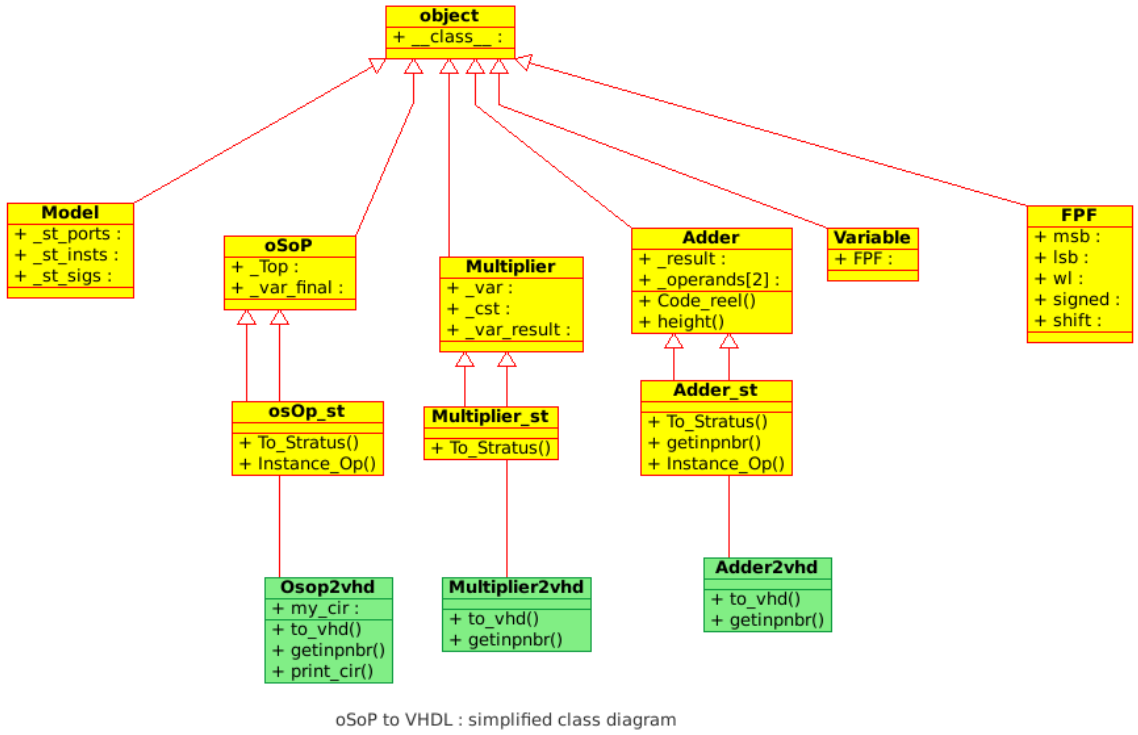


FIGURE 4.10 – Diagramme de classe simplifié oSoP2VHDL.

4.4 Connexion entre les oSoP pour avoir le circuit du filtre

Un fois qu'on a le code VHDL correspondant à chaque somme de produits (oSoP) dans le filtre, on peut les connecter entre eux avec des registres à l'aides des équations d'états, pour avoir le circuit entier qui correspond au filtre.

Chapitre 5

Les résultats

5.1 Environnement de développement et Conditions d'expérimentation

Pour notre travail, on a utilisé les outils suivants :
— **Python 2.7** avec zipfile, lxml.etree, math, scipy.io, numpy, pickle, shutil
— FiPoGen, AMPL/Bonmin
— Stratus (lib arith, modules stratus, packages stratus)
— **Git** pour la gestion de version
— Matlab/Simulink
— GHDL pour la simulation VHDL

Avant chaque expérimentation, il faut toujours :
— configurer la variable d'environnement **PYTHONPATH** pour pointer sur les outils : FiPoGen, Stratus et nos modules mêmes.
— Configurer Stratus en utilisant la commande *eval /soc/coriolis2/etc/coriolis2/coriolisEnv.py*
— Compléter correctement le fichier *config.xml* qui contient les paramètres pour nos outils, à savoir : le chemin vers AMPL et Bonmin, le fichier Simulink à utiliser, le plage d'entrée, l'erreur sur la sortie.

On a comme **résultats** :
— un module "slx2sif" qui permet de générer la Forme Implicite Spécialisée correspondant à un diagramme de Blocs Matlab/Simulink
— une petite correction apportée à FiPoGen
— un module qui utilise Stratus pour générer le code VHDL à partir d'un oSoP
— une application qui permet de générer un code VHDL correspondant à un filtre décrit dans un diagramme de blocs

5.2 Exemple de génération de circuit pour un filtre

Pour un diagramme Simulink donné, les outils sont capables de généré automatiquement le circuit correspondant.
Les deux figures suivantes représentent un schéma-bloc d'un filtre numérique linéaire du second ordre et son circuit correspondant généré automatiquement par les outils.

Notons que pour seulement cette structure là différents types de filtre.[8]

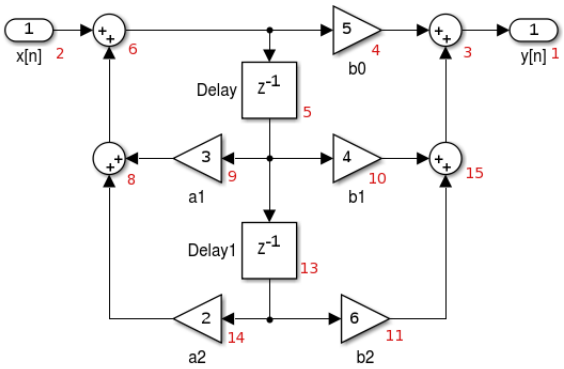


FIGURE 5.1 – exemple de schéma-bloc pour un filtre

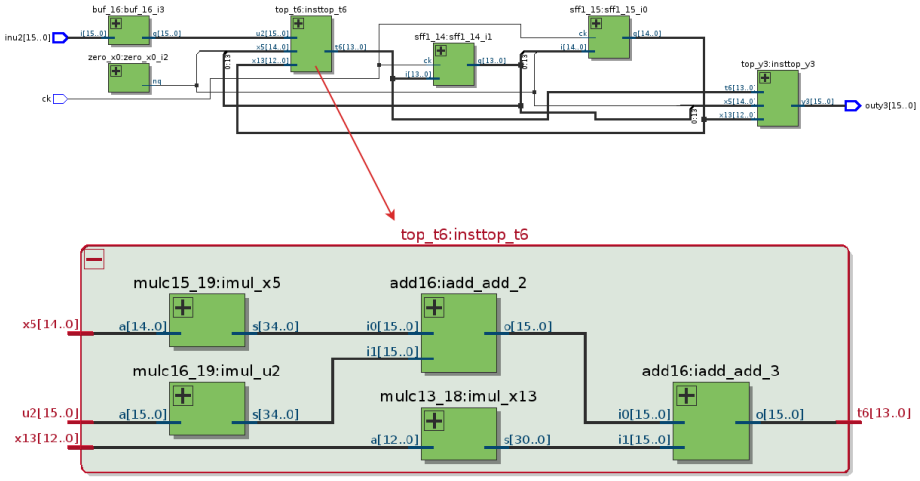


FIGURE 5.2 – exemple de circuit généré pour un filtre

Chapitre 6

Test, Validation et Analyse des résultats

Il faut considérer qu'en tant que développement d'outils et non une réalisation d'un système, la validation de notre travail de manière rigoureux est un peu difficile. Mais on a fait de notre mieux.

6.1 Pour le module SLX2SIF en entrée

Pour ce module, on a fait les tests suivants :
— Validation sur un exemple fait manuellement
— Validation sur quelques structures de filtre
— Validation sur les 2 types de structures de d'adaptateur pour LWDF
— validation sur un exemple de LWDF de 5è ordre

Les fonctionnalités internes vérifiés sont :
— Récupération des paramètres des blocs
  — Gain (Gain)
  — DelayLength (Delay)
  — Inputs (Sum)
— Traitement des sous-systèmes (SubSystem)
— Groupement des additions
— Établissement de l'équation pour chaque bloc

Notons que la liste des oSoP peut être créée automatiquement à partir du schéma-bloc

6.2 Génération du VHDL pour un oSoP

— Au début : est avec un oSoP créé manuellement
— Chaque VHDL de l'oSoP est généré dans un dossier
— Le test est fait manuellement
— test avec la liste des oSoP générée
— Test sur quelques exemples de filtre

6.3 Vérification du circuit correspondant au filtre

La vérification du circuit entier correspondant au filtre a été fait visuellement en regardant le netlist RTL généré. La simulation du VHDL du filtre n'a pas pu être fait par manque de temps.
Pour le faire, il faut :

- Connecter les codes C de chaque oSoP de la même manière qu'on a fait avec le VHDL
- utiliser GHDL pour faire une double simulation

Chapitre 7

Conclusion Générale

Ce travail de stage traite le développement d'outils qui permet d'implémenter en matérielle un algorithme de filtrage numérique. Plus précisément, ils permettent d'avoir le code VHDL du circuit correspondant à un filtre numérique décrit dans un schéma bloc Simmulink. On a eu un prototype qui fonctionne mais des travaux restent à faire pour avoir une solution plus robuste.

D'un point de vue personnel, ce stage a été très enrichissant. Le sujet nécessite de se familiariser avec de nombreux concepts comme la forme implicite spécialisée, l'algèbre linéaire, le format de fichier Simulink, le graphe de calcul oSoP, la programmation en Python, l'outil FiPoGen et Stratus.

De manière explicite, ce travail de stage a contribuer à :
— la mise à l'épreuve de FiPoGen et correction de quelques bugs, surtout sur la manipulation des SIF et d'oSoP
— l'ajout d'import Simulink dans FiPogen : le module slx2sif
— au développement d'un module de génération de circuit à partir d'un oSoP avec Stratus
— la création d'un preuve de concept pour un outil de synthèse de filtre numérique : de la modélisation en diagramme de blocs vers la description matérielle

Néanmoins, des développement reste encore à faire, comme :
— Améliorer le temps d'optimisation des largeurs, essayer d'autres solveurs
— Développer FiPoGen pour traiter complètement les systèmes MIMO
— Étendre le module d'import Simulink pour supporter plus de type de blocs
— Créer un module inverse SIF vers Simulink
— Établir un "TestBench" plus performant

Par ce stage, on a pu montrer que l'intégration de FiPoGen et Stratus dans un flot de conception automatique permet de réduire le temps de développement et de synthèse de filtres numériques, tout en respectant la spécification. Cela a été fait en encapsulant une grande partie du travail d'implémentation du filtre dans un outil. Mais des travaux de développement et de tests intensives sont nécessaires pour avoir un "vrai" outil.

Annexe A

Le calendrier

Afin d'organiser au mieux le stage, on a établit ci-après le planning des différentes tâches à accomplir.

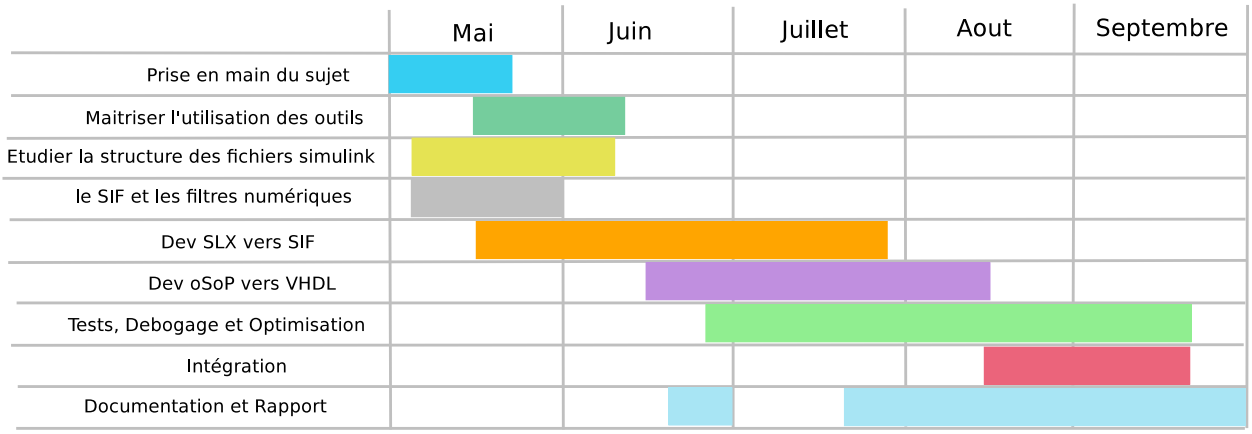


FIGURE A.1 – planning des tâches

Annexe B

Quelques notions essentiels de ce stage

B.1 Les filtres numériques

Le filtrage est un traitement qui permet de transformer un signal. Les filtres numériques agissent sur des signaux à temps discrets c.a.d une séquence de nombres noté $\{x(n)\}$ ou $x(n)$ est la valeur du signal réel au temps $t = n.T$, T étant la période d'échantillonnage.

FIGURE B.1 – Un filtre numérique

Un filtre numérique est entièrement définie par une équation au différence liant l'entrée $x(n)$ et la sortie $y(n)$:

$$y(n) = \sum_{i=0}^{N} b_i.x(n-i) - \sum_{i=1}^{N} a_i.y(n-i) \tag{B.1}$$

A partir de l'équation au différence, on obtient, par transformation en Z, la fonction de transfert du filtre :

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{i=0}^{N} b_i.z^{-i}}{1 + \sum_{i=1}^{N} a_i.z^{-i}} \tag{B.2}$$

Un filtre peut être spécifié par sa réponse impulsionnelle, ou son gain en fréquence ou mieux par son gabarit. Notons que, dans le vocabulaire du traitement des signaux, la "synthèse" consiste à trouver la valeur des coefficients a_n et b_n qui répond à la spécification du filtre.
Dans ce travail, on considère que ce calcul a déjà fait au préalable. Donc, on prend en entrée un filtre déjà définit avec les coefficients exprimés en précision "infinie". On essaiera d'obtenir une description matérielle du filtre.

B.2 Réalisation ou Structure d'un filtre

La manière de faire les calculs dans le filtre définit la structure du filtre. Il existe différente structure d'implémentation d'un filtre, à savoir :

- Forme Directe I
- Forme Directe II : économie de stockage
- Forme Transposée
- Représentation d'État : utilisée en automatique

Si les calculs dans le filtre sont faits de manière exacte, il y a aucune différence sur les structures. Mais en précision finie, le résultat dépend beaucoup de la structure utilisé.

- les coefficients ne sont pas exactes
- les calculs ne sont pas exactes

B.3 Voir la netlist sous Quartus

Pour voir la netlist RTL [1], Il faut :

— Créer un projet Quartus
Spécifier le ficheir top-level
Ajouter tous les fichiers VHD du design au projet
— Ajouter SXLIB au librairie du projet
Assignements > Settings > Libraries : /soc/alliance/cells/sxlib
— Élaborer le design :
Processing > Start > Analysis&Elaboration
— Voir le netlist :
Tools > Netlist Viewers > RTL viewer

On voit sur la figureB.2 le Netlist d'un circuit correspondant à un oSoP,visualisé sous RTLViewer de Quartus.

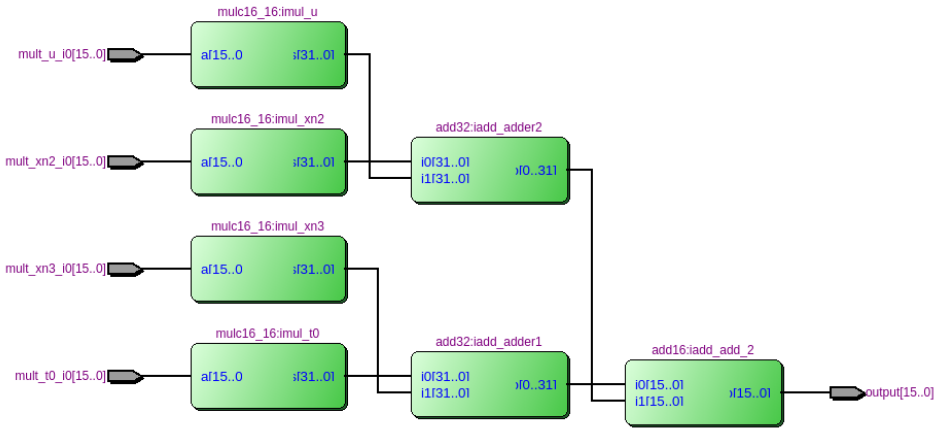


FIGURE B.2 – exemple de Netlist généré par osop2vhd

1. Register Transfert Level

B.4 Listing des fichiers utiles pour l'optimisation

fichier d'option de Bonmin :

```
bonmin.time_limit 5
bonmin.algorithm B-BB
```

Le fichier de modèle pour AMPL :

```
#----------------------------------------
# Exponetial Nonlinear Knapsack problem
#    . Version for several constraints
#----------------------------------------

option solver bonmin;

param N;  # Nbr wx variables
param M;  # Nbr wy variables
param Q;  # Nbr of constraints

param Ux; # Upper bound on Wx
param Uy; # Upper bound on Wy

set Colx := 1 .. N; # Wx variable indices
set Coly := 1 .. M; # Wy variable indices
set Rows := 1 .. Q; # Constraints indices

param c{Colx};
param d{Coly};

param coefx{Rows, Colx};
param coefy{Rows, Coly};

var Wx{Colx} >=0, integer;
var Wy{Coly} >=0, integer;

minimize Obj :
   sum{j in Colx} c[j]*Wx[j] + sum{j in Coly} d[j]*Wy[j];

subject to Const{i in Rows}:
    sum{j in Colx} coefx[i,j]*2**(-Wx[j]) + sum{j in Coly} coefy[i,j]*2**(-Wy[j]) <= 1;
```

Annexe C

Les fichiers de mon répertoire au laboratoire

Cette partie décrit les fichiers produit pendant mon stage. La plupart des dossiers contient un readme et un Makefile qui contient les démarches à faire pour utiliser les modules développés. Le chemin sont par rapport à mon répertoire **/users/enseig/ravoson**.

- **stage-m2/** : le dossier de mon stage
- **stage-m2/slx2sif_python_impl** : le dossier contenant une première implémentation du module SLX2SIF, simple, permettant une compréhension facile du principe
- **stage-maminionja** : le dossier contenant le code source du module SLX2SIF
- **stage-m2/osop/mfixed_point/example** : le dossier contenant le code source du module osop vers vhdl
- **stage-m2/rapport** : le dossier contenant le rapport de spécification et de réalisation
- **stage-m2/fxp_synth** : dossier contenant les outils rassemblés dans un flot
- **fipogen** : un clone local de FiPoGen

Notons que le dossier **stage-m2/fxp_synth** contient la version la plus récente de tous les outils.

Les outils externes utilisés :

- sur la machine "*cabezon*" : /dsk/l1/misc/ravoson/amplide-demo/**ampl**
- sur la machine "*cabezon*" : /dsk/l1/misc/ravoson/**Bonmin**-1.8.3/build/bin
- /soc/coriolis2/lib/python2.6/site-packages/**stratus**
- /users/outil/**arith**/latest/lib/
- /users/outil/arith/latest/**modules_stratus**
- /soc/alliance/cells/**sxlib**

Bibliographie

[1] Thibault Hilaire. *Analyse et Sunthèse de l'implémentation de lois de contrôle-commande en précision finie*. Nantes, France, 2006.

[2] Benoit LOPEZ. *Implémentation optimale de filtre linéaire en arithmétique virgule fixe*. PhD thesis, UPMC, 2014.

[3] Thibault HILAIRE. Finite Wordlength Realizations Toolbox User's Guide. `https://gforge.inria.fr/projects/fwrtoolbox/`, 2009.

[4] Thibault Hilaire Anastasia Volkova. "Fixed-point implementation of Lattice Wave Digital Filter : Comparison and error analysis". ( ), 2015.

[5] Christophe Lauter Anastasia Volkova, thibault Hilaire. Reliable Evaluation of Worst-Case Peak Gain Matrix in Multiple Precision. 2nd IEEE Symposium on computer Arithmetic, 2015.

[6] Sophie Belloeil. Stratus User's Manual. `https://www-soc.lip6.fr/sesi-docs/coriolis2-docs/coriolis2/en/html/stratus/index.html`, 2012.

[7] Sophie Belloeil. Stratus Developper's Manual. `file:///soc/coriolis2/share/doc/coriolis2/en/html/stratus_developper/index.html`, 2012.

[8] Richard Lyons and Amy Bell. The swiss army knife of digital networks. •, -.