ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE LA
MAITRISE EN GÉNIE ÉLECTRIQUE
M.Ing.

PAR
PLOURDE, Frédéric

DÉVELOPPEMENT D'UNE MÉTHODOLOGIE D'ESTIMATION DE L'UTILISATION DES RESSOURCES MÉMOIRES SUR UNE PUCE DSP MULTI-NOYAUX

MONTRÉAL, LE 8 MAI 2009

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE

M. Claude Thibeault, directeur de mémoire
Département de génie électrique à l'École de technologie supérieure

M. François Gagnon, membre du jury
Département de génie électrique à l'École de technologie supérieure

M. Jean-François Boland, président du jury
Département de génie électrique à l'École de technologie supérieure

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 23 AVRIL 2009

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

REMERCIEMENTS

En premier lieu, je tiens à remercier mes collègues du laboratoire LACIME pour leurs conseils lors de la rédaction de ce mémoire de même que pour leur support moral. Grâce à eux j'ai pu mener à bien le laborieux processus de rédaction et garder le cap malgré certaines distractions.

Ensuite, je remercie sincèrement M. Claude Thibeault pour sa patience et ses conseils tout au long de mes recherches et lors de la rédaction. Nos nombreuses discussions et rencontres furent constructives, elles permirent à mes recherches d'arriver à terme.

Finalement, merci à Louise Savard pour son aide lors de la correction de ce mémoire.

Frédéric Plourde

DÉVELOPPEMENT D'UNE MÉTHODOLOGIE D'ESTIMATION DE L'UTILISATION DES RESSOURCES MÉMOIRES SUR UNE PUCE DSP MULTI-NOYAUX

PLOURDE, Frédéric

RÉSUMÉ

Ce mémoire présente le développement d'une méthodologie d'estimation des ressources mémoires utilisées par des applications de traitement du signal sur un circuit DSP multi-noyaux. L'estimation est effectuée à partir de représentations haut niveau des applications dans l'environnement de modélisation Matlab/Simulink. Le domaine d'application visé par la méthodologie est celui des communications numériques sans fil. Ainsi, la méthodologie a été vérifiée sur plusieurs fonctions de traitement du signal numérique couramment utilisé dans ce domaine d'application. Il est démontré qu'il est possible, à partir de représentations Embedded Matlab Function, d'estimer la quantité de mémoire requise par les fonctions sur un noyau de processeur DSP.

Mot-Clés : DSP, estimation, multi-noyaux, Matlab/Simulink, mémoire

DEVELOPMENT OF A METHODOLOGY TO ESTIMATE MEMORY USAGE ON A MULTI-CORES DSP

PLOURDE, Frédéric

ABSTRACT

This master's thesis presents the development of a methodology for estimating the memory resources used by signal processing applications running on a multi-cores DSP. This estimation is based on a high level representation of the application in the Matlab/Simulink environment. The chosen domain of application is the wireless numeric communications. The methodology was tested on frequently used signal processing functions in wireless communications. It is shown, that memory consumption can be estimated from an Embedded Matlab Function for a particular application on a DSP core.

Key words : DSP, estimation, multi-cores, Matlab/Simulink, memory

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

LACIME	Laboratoire de communication et d'intégration de la microélectronique
OPERA	Octasic, Polytechnique, ÉTS, Radio Application
SOC	Système sur puce
RAM	Random access memory
ROM	Read-only memory
EEPROM	Electrically Erasable Programmable Read Only Memory
DSP	Processeur de signal numérique
UMTS	Universal Mobile Telecommunication System
OFDM	Orthogonal Frequency Division Multiplexing
OFDMA	Orthogonal Frequency Division Multiple Access
MIMO	Multiple-input Multiple-output
Mbps	Mégabit par seconde
3G	Troisième génération
LTE	Long Term Evolution
MC-CDMA	Multi-Carrier Code Division Multiple Access
EMF	Embedded Matlab Function
FIR	Filtre à réponse impulsionnelle finie
LMS	Least-mean-square
IIR	Filtre à réponse impulsionnelle infinie
TFR	Transformée de Fourier rapide
MAC	Multiplicateur-accumulateur

SIMD	Single Instruction Multiple Data
SQNR	Signal-to-Quantization Noise Ratio
ALU	Arithmetic Logic Unit
FIFO	First input, first output
ISI	Interférence inter-symbole
QAM	Modulation d'amplitude en quadrature
VA	Algorithme de Viterbi
ACS	Addition-comparaison-selection
IP	Protocole internet
E/S	Entrées et Sorties
MIMD	Multiple Instructions Multiple Data
UT	Unité de traitement
DMA	Contrôleur d'accès direct à la mémoire
FPGA	Field Programmable Gate Array
DDR	Double Data Rate
SDRAM	Synchronous Dynamic RAM
IOR	Registre de sortie d'instruction
IDE	Environnement de développement intégré
ISS	Simulation d'instructions
MIPS	Million d'instructions par secondes
MOPS	Millions d'instructions par seconde

MACS	Multiplications-additions par seconde
CDFG	Graphe de contrôle et de flux de données
RTW	Real-Time Workshop
TPA	Test de performances algorithmiques
TI	Texas Instruments Inc.

INTRODUCTION

L'évolution des technologies de télécommunication sans fil numériques vers des vitesses de transfert à haut débit présente de nombreux défis pour les concepteurs de systèmes. En effet, afin de pouvoir rejoindre la limite de capacité définie par la Loi de Shannon, les concepteurs doivent avoir recours à des algorithmes de traitement du signal complexes. La modulation OFDM, le codage de canal Turbo et les systèmes à entrée multiple sortie multiple (MIMO) sont des exemples de techniques qui s'approchent de la limite de Shannon. Or, ces techniques sont exigeantes en terme de puissance de calcul nécessaire pour les exécuter. De plus, des technologies comme l'évolution à long terme (LTE) des systèmes de communications mobiles de troisième génération (3G) combinent les trois techniques.

Les circuits intégrés utilisés par concepteurs de systèmes sont plus puissants et ont une architecture parallèle. Ces circuits peuvent être des systèmes sur puces (SOC) ou des processeurs de traitement du signal (DSP) à multi-noyaux. Les SOC sont des circuits intégrés conçus à partir de plusieurs éléments hétérogènes tels que des noyaux DSP, des accélérateurs matériels, interfaces d'entrée sortie et mémoires. Les DSP multi-noyaux sont conçus à partir d'une architecture homogène de plusieurs noyaux DSP identiques. Malgré leur complexité, ces circuits ont des ressources matérielles limitées en termes de puissance de calcul et quantité de mémoire disponible.

Ainsi, en plus d'avoir à gérer avec la complexité des algorithmes et des circuits, les concepteurs doivent respecter des contraintes de temps et de coût sévères pour obtenir un maximum de rentabilité de leur système. Il s'avère crucial de déterminer quelles sont les ressources nécessaires pour l'exécution d'une application afin de maximiser le rendement de la plateforme matérielle utilisée. Par contre, les applications de télécommunications sont d'abord modélisées à l'aide de représentation haut niveau où il est difficile de prédire avec précision les ressources utilisées par l'application sur la plateforme matérielle. En effet, une représentation avec un haut niveau d'abstraction ne contient pas d'informations relatives au

matériel. Une représentation haut niveau est définie par un ensemble de blocs fonctionnels caractérisant l'application à implémenter au niveau algorithmique. Par ailleurs, il est important de déterminer si la quantité de mémoire disponible sur le circuit est suffisante afin de permettre l'exécution de l'application dans les contraintes de temps requises.

La recherche présentée dans ce mémoire s'inscrit dans ce contexte et propose une méthodologie d'estimation des ressources mémoires à partir de modèles Matlab/Simulink. La méthodologie a été développée en considérant le circuit DSP Vocallo, de la compagnie Octasic inc, comme plateforme matérielle . Le Vocallo est un nouveau DSP multi-noyaux à architecture homogène qui cible les applications de télécommunications numériques sans-fil. La présente recherche fait partie du projet OPÉRA qui tente de définir une méthodologie globale de conception des applications de communications numériques sur des architectures multi-noyaux.

Les objectifs de la recherche décrit dans ce mémoire sont de fournir un ou des outils permettant l'extraction des contraintes de performances liés à l'utilisation de la mémoire à partir d'une description haut-niveau d'une application. Le but recherché est d'automatiser d'une certaine manière l'estimation des performances de l'application sur la plateforme matérielle. Bien que plusieurs méthodologies d'estimation existent, celles-ci requièrent de nombreux paramètres d'entrées qui doivent être fournis manuellement par l'usager. D'autres méthodologies proposées par (Florin, Hongwei et Ilie, 2007) nécessitent un format intermédiaire de caractérisation de l'application et qui n'est pas réutilisé par la suite dans le processus de développement. Donc, la contribution principale de ce projet est de fournir un outil automatisant au maximum une méthodologie d'estimation à partir d'un modèle Matlab/Simulink représentant l'application. Une deuxième contribution du projet est d'indiquer au développeur les parties d'une application modélisée à haut-niveau qui nécessiteront un effort d'optimisation lors de l'implémentation. Afin d'y parvenir, l'outil procurera des informations sur l'utilisation des ressources mémoires par les primitives

constituant l'application. Ces primitives pourront ensuite être distribuées sur plusieurs noyaux du circuit Vocallo.

Le présent document est donc divisé selon la structure suivante. Premièrement, la mise en contexte de la recherche et sa situation dans le projet OPÉRA est présentée. Deuxièmement, les applications visées par le projet OPÉRA sont étudiées. Ensuite, l'Architecture du circuit Vocallo est présentée. Dans le quatrième chapitre, les méthodologies d'estimation existantes dans la littérature sont analysées. Finalement, le développement de la méthodologie proposée est décrit en détails.

CHAPITRE I

MISE EN CONTEXTE

La recherche dont fait l'objet le présent document s'inscrit dans le cadre d'un projet coopératif de recherche et développement entre le LACIME de l'École de technologie supérieure, l'École Polytechnique et la compagnie Octasic inc. Ce projet vise le développement de nouvelles méthodologies de conception pour des applications de communications numériques ciblant une plateforme à multiples processeurs. Ce chapitre se divise en deux sections, la première traitera du projet global OPERA et la deuxième du projet dont fait l'objet ce document.

1.1 Projet OPERA

Le but du projet OPERA « Octasic, Polytechnique, ÉTS, Radio Application » est de rechercher de nouvelles stratégies de conception dans le domaine des télécommunications sans fils pour une plateforme programmable. À partir d'une plateforme donnée, nous cherchons à savoir si une application ciblée peut y être implémentée. La méthodologie propose des techniques et des outils qui permettent l'analyse de diverses applications afin de déterminer leurs besoins en termes de ressources et leurs performances potentielles. La plate-forme utilisée comme base par le projet est le processeur Vocallo de la compagnie Octasic.

1.1.1 Processeur Vocallo

Le processeur Vocallo d'Octasic est un nouveau circuit multiprocesseur spécialisé dans le traitement des signaux numériques. Il représente bien les nouvelles générations de circuits numériques dits systèmes sur puce (SOC) et multiprocesseurs. L'arrivée de ce type de processeurs permet l'intégration d'applications complexes sur une seule puce. Par contre, cela augmente la difficulté à prévoir les performances d'une application sur le processeur

avant son implémentation finale. Le Vocallo s'avère aussi une plate-forme intéressante en termes de partitionnement des composantes de l'application sur les différents processeurs.

1.1.2 Design basé sur une plate-forme et design dérivé

Dans les dernières années, l'avènement de procédés de fabrication des circuits intégrés plus sophistiqués a permis aux concepteurs de suivre la loi Moore quant à l'augmentation des performances. Par contre, cela a eu pour effet d'augmenter les coûts de production non récurrents et l'intervalle de temps entre la conception et la mise en marché. Dans un marché où les contraintes de coût et de temps déterminent le succès d'un nouveau circuit, il s'avère important de développer des méthodologies de conception plus efficaces. Ainsi, sont apparues de nouvelles tendances dans l'industrie comme le design basée sur la plate-forme (platform-based design) et les designs dérivés (design re-use). Certains (Keutzer et al., 2000) croient que les prochaines générations de circuits qui connaîtront du succès auront été conçues avec l'aide de méthodologies basées sur une plate-forme. Ces méthodologies devront être aussi en mesure de fournir aux développeurs des indications sur les performances de l'application sur la plate-forme dans les premières phases de l'implémentation. Il est donc important de disposer de métriques de performances pertinentes et d'une description haut niveau de l'application qui soit accordable avec les spécifications de la plate-forme matérielle. Une plate-forme matérielle est définie par (Ferrari et Sangiovanni-Vincentelli, 1999) comme étant une famille d'architectures qui satisfait un ensemble de contraintes architecturales qui sont imposées pour permettre la réutilisation de composants matériels et logiciels. Dans le cas du projet OPERA, la plate-forme visée est le Vocallo d'Octasic, les buts du projet étant de fournir une méthodologie capable de guider les concepteurs d'applications sur le Vocallo à partir d'un niveau d'abstraction élevé et ce dans des délais restreints. Le projet tentera de faire le lien entre la description d'une application et les spécifications du matériel afin de générer un ensemble de métriques de performances. Ces métriques devraient évaluer les performances des

principaux composants d'un système embarqué qui sont selon (Ferrari et Sangiovanni-Vincentelli, 1999) :

- Les performances des unités de traitement ou processeurs;
- La vitesse et la taille du système de mémoire (RAM, ROM, Flash, EEPROM);
- Les canaux analogiques, le minutage, les canaux numériques et la vitesse de communication des entrées/sorties du système.

De plus, la méthodologie devrait être en mesure d'identifier les goulots d'étranglement de la plate-forme pour permettre d'éventuels design dérivés.

1.1.3 Les applications visées par le projet OPERA

Comme le nom du projet l'indique, les applications cibles du projet sont les communications par ondes radio. En effet, le domaine des communications par ondes radio (sans-fil), en particulier les communications numériques s'avère être un secteur important du marché des circuits numériques. Grâce à l'incessante réduction à l'échelle des transistors qui permet de fabriquer des circuits toujours plus rapides, plus petits et moins énergivores, le nombre d'appareils envahissant le marché n'a cessé de croître. Par exemple, «en 2002, approximativement 405 millions de téléphones cellulaires ont été vendus dans le monde »(Neuvo, 2004). Avec un nombre estimé de 3 milliards d'abonnés en 2008 (Neuvo, 2004), le marché des télécommunications mobiles représente une part intéressante des applications utilisant un processeur de signal numérique (DSP). C'est donc dans ce contexte que le projet OPERA se concentrera sur les applications de communications sans-fil. Les trois principales applications étudiées au cours du projet seront les systèmes UMTS, OFDM/OFDMA et MIMO. Ces systèmes ou technologies de communications ont été choisis car ils proposent des défis intéressants en termes de complexité et aussi parce que la prochaine génération de systèmes de communications sans-fil utilisera ces principes. En effet, la demande pour une bande passante accrue force les développeurs à adopter des

algorithmes plus complexes. Avec des taux de transferts spécifiés de 100 Mbps pour une liaison descendante de la prochaine évolution à long terme de la 3ième génération de téléphonie cellulaire (3G LTE). (Ashok, 2007) prévoit une complexification des techniques de traitement des signaux. Selon (Ashok, 2007) ces techniques incluent les systèmes multientrées multisorties (MIMO) et les modulations comme l'accès multiple par répartition orthogonale de la fréquence (OFDMA) et accès multiple par répartition en code avec multiporteuses (MC-CDMA). Ces techniques sont aussi utilisées dans les familles de technologies WIMAX et Wifi. Il est donc pertinent de s'intéresser de près aux algorithmes de ces techniques dans le but de percer le marché avec un nouveau circuit comme le Vocallo. Les différentes applications seront examinées plus en détails dans le prochain chapitre.

1.2 Estimation des contraintes de performances liées à l'utilisation de la mémoire

Le projet décrit dans ce mémoire s'inscrit dans la problématique globale du projet OPERA. De manière plus spécifique, ce mémoire s'intéresse à l'estimation des contraintes de performances liées à l'utilisation de la mémoire.

Lorsque vient le temps de développer une application de traitement de signal complexe sur un processeur, l'aspect de l'utilisation des ressources mémoires est critique. Comme le souligne (Panda et al., 2001) « les questions de mémoire [...] ont souvent un impact important sur les performances d'un système embarqué, sa puissance dissipée et son coût total d'implémentation ». Dans le cas d'un processeur numérique du signal, les applications sont souvent confinées à une mémoire locale intégrée et dont la quantité disponible est limitée. De plus, l'accès à la mémoire externe s'effectue à vitesse réduite par rapport à la vitesse de traitement du processeur. Il s'avère donc important pour le concepteur d'obtenir des estimations sur l'utilisation des ressources mémoires par l'application afin de réaliser la meilleure implémentation possible. Aussi, comme le suggèrent (Thiele, Wandeler et Chakraborty, 2005), il est très important d'obtenir ces informations dans les premières étapes de la conception afin de prendre les bonnes décisions lors de l'implémentation finale. Or, les

concepteurs modélisent souvent les applications à l'aide de descriptions haut niveau où il y a abstraction des contraintes de la plateforme. Le défi est important quand on considère que les contraintes des systèmes embarqués sont strictes, la taille de mémoire limité et la consommation de puissance minimale. La méthodologie proposée devra fournir une estimation assez précise pour guider le concepteur, mais en même temps extraire les informations à partir d'un modèle haut niveau pour être utilisées dans la phase initiale de conception.

1.2.1 Type de la méthodologie proposée

Certaines des méthodologies d'estimations existantes (Kjeldsberg, Catthoor et Aas, 2003) font totalement abstraction de la plateforme. Ce type de méthodes est très adapté aux projets où le choix de la plateforme reste encore à déterminer, alors les résultats obtenus servent lors du choix de la plateforme. Mais, dans le cas de notre projet, la plateforme est prédéterminée, le Vocallo d'Octasic, l'objectif du projet étant plutôt la réutilisation d'une même plateforme pour plusieurs applications. La méthodologie devra donc fournir aux concepteurs les indications de performances en rapport avec cette plateforme. C'est pourquoi la méthodologie proposée tient compte des caractéristiques de l'architecture du circuit.

1.2.2 Modèles de représentations haut niveau

Le choix de la représentation haut niveau de l'application est un facteur important dans le développement de la méthodologie. En effet, les informations qui servent à effectuer les estimations doivent d'être extraites à partir de cette représentation. Celle-ci doit être assez complète afin de permettre la définition des structures de données de l'application. Ces représentations comprennent aussi les algorithmes de traitement. Dans le domaine du traitement des signaux, la représentation par des boucles imbriquées et des tableaux multidimensionnels est souvent utilisée comme l'indiquent (Kjeldsberg, Catthoor et Aas, 2003). Dans le cadre du projet OPERA le modèle de représentation choisi est

l'environnement Matlab/Simmulink. « Simulink est un logiciel de modélisation, simulation et d'analyse des systèmes dynamiques (The MathWorks, 2007)». Ce choix est dicté par les objectifs du projet OPERA qui sont de développer de nouvelles applications pour la plateforme Vocallo. L'environnement Matlab/Simulink est un outil puissant de développement et de simulation pour les applications de télécommunications puisqu'il incorpore des librairies d'algorithmes communément utilisées dans le domaine. La méthodologie proposée effectue des estimations des ressources mémoires à partir de modèles applications décrits en Simulink, plus particulièrement à partir des blocs « Embedded Matlab Function ». Ces blocs sont utilisés pour insérer des scripts en langage Matlab dans des modèles Simulink. Ils sont généralement utilisés pour générer du code C qui peut être implémenté sur un système embarqué. Cet aspect s'avère important dans le développement de méthodologies parce qu'il facilite le rapprochement entre la plateforme et la représentation haut niveau.

CHAPITRE 2

LES APPLICATIONS ÉTUDIÉES

Le principal objectif du projet OPERA est le développement de méthodologies de conception basées sur une plateforme existante. L'approche de conception préconisée est du type ascendante (bottom-up), c'est-à-dire qu'à partir d'une même architecture matérielle de base, on conçoit différentes applications. La tâche des concepteurs est d'implémenter une application définie par des spécifications sur une plateforme aux caractéristiques fixes. Comme il a été mentionné précédemment la classe d'applications visée par le projet est celle des télécommunications. Cette classe étant assez vaste, l'étude se concentrera sur certaines applications qui représentent le plus d'intérêt au niveau de part de marchés qu'elles occupent et aussi en termes de potentiel de développement.

Dans ce chapitre, nous commenceront par examiner les opérations plus générales au domaine du traitement des signaux pour ensuite examiner les technologies cellulaires de troisième génération (3G), en particulier le système universel de télécommunication avec les mobiles (UMTS) et finalement pour terminer les technologies OFDM(A).

2.1 Opérations de traitements des signaux

Bien que la classe d'applications étudiée dans le cadre du projet OPERA soit celle des télécommunications, la majorité des recherches dont fait l'objet le présent document ont été faites sur les opérations de traitements des signaux fondamentales. Leur étude demeure pertinente puisqu'elles constituent les composantes de base des applications de communications numériques. Le nombre d'opérations fondamentales est élevé et c'est pourquoi seulement une liste restreinte des fonctions les plus communes a été étudiée. Cette liste a été déterminée à partir d'une liste de fonctions qui composent un ensemble

d'algorithmes servant de tests de performance pour DSP. L'ensemble choisi est le « BDTI DSP Kernel Benchmarks » de la compagnie BDTI. Selon (Berkeley Design Technology, 2007): « le BDTI DSP Kernel Benchmarks est un ensemble de 12 algorithmes de tests de performance pour DSP qui représentent les opérations fondamentales constituant la plupart des applications de traitement de signaux ». La liste des fonctions de l'ensemble est présentée dans le Tableau 2.1. C'est donc à partir de cette liste qu'ont été choisies les fonctions étudiées pour établir la méthode d'estimation de l'utilisation des ressources mémoires. Les fonctions suivantes ont servi au développement de la méthode :

- Filtre FIR réel par bloc;
- Filtre FIR complexe;
- Filtre égaliseur adaptatif LMS complexe;
- Décodeur de Viterbi;
- Maximum vectoriel;
- Addition de vecteurs.

Ces fonctions fréquemment rencontrées dans les applications de télécommunication ont été utilisées lors de la vérification de la méthode d'estimation. Elles seront décrites plus en détails dans les sections qui suivent.

Tableau 2.1 Liste des fonctions du test de performances
BDTI DSP Kernel Benchmarks

Tiré de Berkeley Design Technology (2007)

Fonction	Description	Exemple d'application
Filtre à réponse impulsionnelle finie (FIR) réel par bloc	Filtre FIR qui reçoit des blocs de données réelles à traiter	G.7258 encodage de la voix
Filtre FIR complexe par bloc	Filtre FIR qui reçoit des blocs de données complexes	Égalisation de canal modem
Filtre FIR réel à échantillon simple	Filtre FIR qui reçoit un seul échantillon de données réelles à traiter à la fois	Traitement de la voix, filtrage général
Filtre FIR moindres carrés moyens "least-mean-square" (LMS) adaptatif	Filtre FIR LMS adaptatif qui reçoit un seul échantillon de données réelles à traiter	Égalisation de canal, contrôle de servo, encodage linéaire prédictif
Filtre à réponse impulsionnelle infinie (IIR)	IIR qui traite un seul échantillon de données réelles à la fois	Traitement de l'audio, filtrage général
Produit scalaire	Produit scalaire de deux vecteurs	Convolution, corrélation, multiplication de matrices, traitement de signaux multidimensionnels
Addition de vecteurs	Addition de deux vecteurs	Graphique, combinaison de signal audio, recherche de vecteurs
Maximum vectoriel	Localisation de la valeur maximale d'un vecteur	Codage de contrôle d'erreurs, algorithme de bloc de calcul point flottant
Décodeur de Viterbi	Décodage d'un bloc de bits qui a été encodé par un code convolutionnel	Code contrôleur d'erreurs
Machine à états finis	Traitement séquentiel en fonction de certaines conditions	Contrôle d'opérations
TFR de 256 points en base 2	Conversion d'un signal du domaine temporel au domaine fréquentiel	Radar, Compression audio MPEG, analyse spectrale
Décompactage de bits	Décompaction des données de longueur variable d'un flot de bits	Décompression audio, acheminement de protocole

2.1.1 Filtre à réponse impulsionnelle finie forme générale (FIR)

Les filtres FIR représentent une des fonctions les plus couramment implémentées dans les DSP. Ces filtres ont une réponse à une impulsion qui est égale à zéro après un temps

déterminé. Cela est dû au fait que dans un filtre FIR il n'y pas de rétroaction venant de la sortie du filtre. Une autre caractéristique utile est leur phase linéaire qui garantit que tous les composants d'un signal sont retardés selon le même délai. Les fonctions qui peuvent être implémentées avec des filtres FIR sont par exemple: des filtres passe-bas, égaliseur ou bien de mise en forme. Ils peuvent traiter des données réelles ou complexes en fonction de l'application. Les filtres sont définis par l'équation générale suivante :

$$y(n)=\sum_{k=0}^{N-1}h(k)x(n-k) \tag{2.1}$$

Où $x(n)$ représente l'entrée du filtre, $y(n)$ la sortie et $h(k)$ les N coefficients du filtre FIR, comme on peut le voir dans la Figure 2.1 qui présente la structure générale d'un filtre.

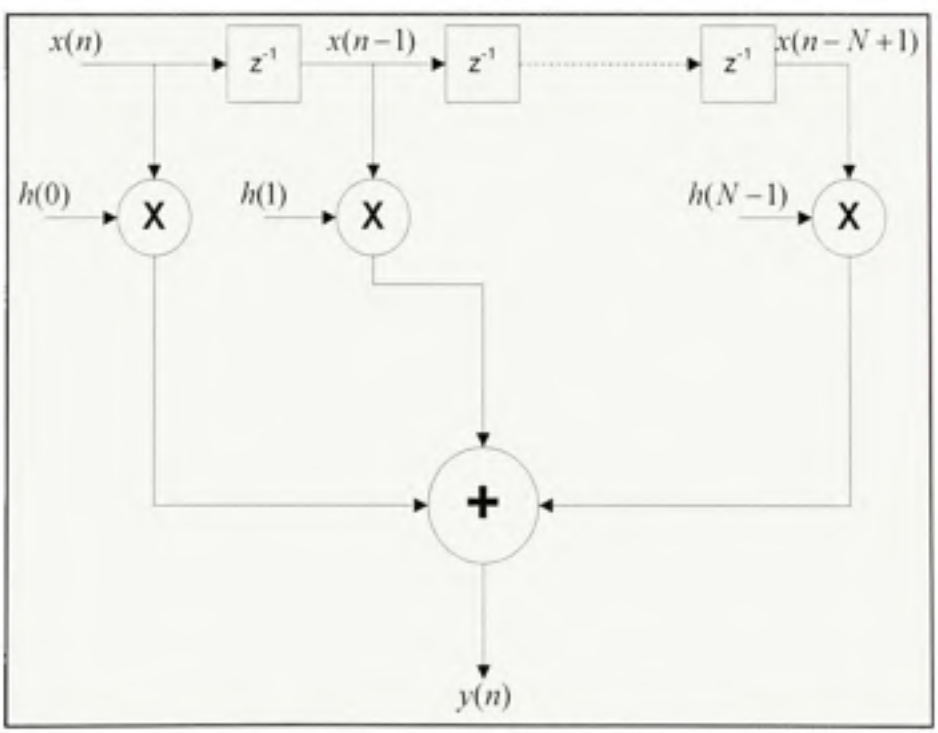

Figure 2.1 Structure générale d'un filtre FIR.

En considérant cette structure générale, on peut déduire plusieurs types d'implémentations. Le type d'implémentation peut varier en fonction de l'application pour lequel est destiné le filtre ou en fonction de la plate-forme. À partir de cette même structure, il est aussi possible d'extraire les principaux composants du filtre qui sont : les multiplicateurs, l'additionneur, une ligne à délais et différents tampons. Si la plate-forme d'implémentation le permet, il est possible de remplacer les multiplicateurs et additionneurs par un multiplicateur-accumulateur (MAC). Dans la présente recherche, les composants intéressants sont les différents tampons et la ligne à délais du filtre puisque ce sont eux qui utilisent les ressources mémoires de la plate-forme. Examinons maintenant les deux types de FIR qui ont été implémentés pour cette recherche, le FIR réel par bloc et le FIR complexe.

2.1.2 FIR réel par bloc

Le filtre FIR réel par bloc est un composant dont la réalisation sur un DSP est assez simple puisqu'elle s'effectue avec une opération MAC. De plus, puisque le traitement s'effectue sur un bloc d'échantillons, la contrainte de temps est la période entre l'arrivée de chaque bloc. Il est à noter que la notion de trame apparait dans certaines littératures, une trame étant généralement un bloc d'échantillons, de longueur définie, transmis entre différent modules dans une application de télécommunication. Le FIR réel par bloc est un cas intéressant pour la vérification de la méthode d'estimation puisqu'il est possible de calculer l'utilisation des ressources à l'aide d'une simple formule mathématique. Un autre point intéressant concerne la quantité de mémoire consommée, elle peut varier grandement en fonction des spécifications du filtre.

2.1.2.1 Spécifications du FIR réel par bloc

En plus d'être défini par l'équation générale du FIR(2.1), le FIR réel par bloc est aussi spécifié par les caractéristiques du Tableau 2.2. Avec celles-ci on peut déduire l'équation d'utilisation de la mémoire du filtre. En effet, il est possible de représenter les différents

tampons du filtre en fonction de ces caractéristiques. Parce qu'il s'agit d'un filtre traitant les données par bloc, il est nécessaire d'avoir des tampons pour recevoir les données non-traitées et les sorties du filtre La taille de ces tampons est définie par le nombre d'échantillons par bloc (N_s) et le nombre de bits de quantification des échantillons (N_q). On nomme ces tampons les tampons d'entrée ($\prod_e$) et de sortie ($\prod_s$) de la fonction.

Tableau 2.2 Caractéristiques
du FIR réel par bloc

N_s	Nombre d'échantillons par bloc à traiter
N	Nombre de coefficients
N_q	Nombre de bits de quantification par échantillon
T_i	Temps d'arrivée des blocs

Il est aussi important de noter qu'étant donné que la taille des bus et des opérandes des circuits DSP est souvent fixe, le nombre de bits de quantification des échantillons doit être remplacé par les tailles des données supportées par le processeur choisi pour l'implémentation. Les tailles de données les plus couramment supportées par les processeurs DSP sont : 8 bits (char), 16 bits (int short) et 32 bits (int). Certains DSP ont aussi des instructions spécialisées pour effectuer un traitement sur plusieurs données de tailles réduites en une seule instruction (SIMD). Il faut donc tenir compte des particularités du DSP lors de l'estimation des tailles des tampons nécessaires. Outre les tampons d'entrées et de sorties, un filtre FIR comporte habituellement un tampon pour les valeurs des coefficients ($\prod_h$) et un tampon pour la ligne à délais des échantillons passés ($\prod_d$). Leurs tailles est en fonction du nombre de coefficients du filtre et du type des données choisies. La Figure 2.2 représente le FIR avec ses principaux tampons décrits précédemment.

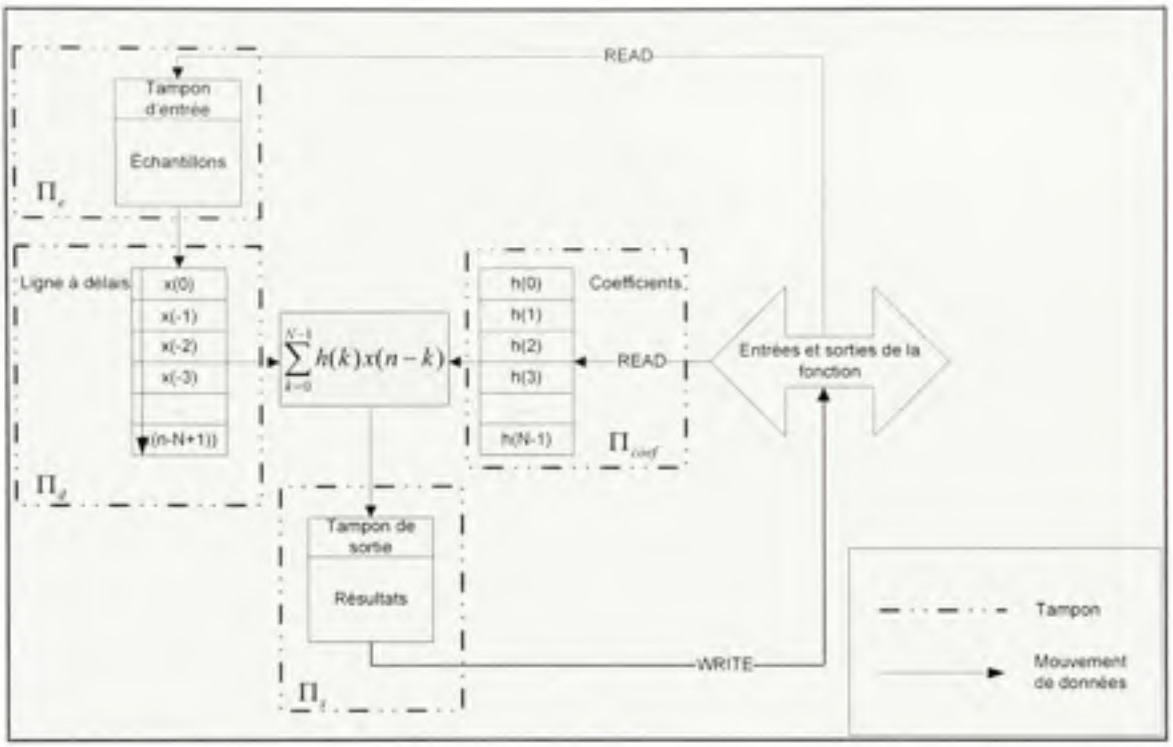

Figure 2.2 Filtre FIR réel par bloc avec ses principaux tampons.

2.1.2.2 Équation d'estimation des ressources du FIR réel par blocs

L'équation d'estimation est une formule mathématique qui permet de calculer la quantité de mémoire utilisée par les tampons du filtre. Elle a été déduite d'après l'étude des caractéristiques du filtre et de l'organisation des tampons. Il peut exister plusieurs versions de cette équation puisque l'organisation des tampons est définie lors de l'implémentation du filtre par le développeur. La forme du FIR présenté ci-haut en est une parmi plusieurs autres. Cette forme est considérée plus simple à implémenter et plus adaptable à différentes plateformes. Une autre forme possible est celle où un seul tampon est utilisé pour les entrées et sorties. Les résultats du filtre écrasent les échantillons d'entrées à mesure que celui traite ces derniers. Donc, en examinant le schéma de l'implémentation choisie, on en déduit que la

quantité de mémoire totale utilisée par les tampons du FIR (Π_t) est donnée par l'équation(2.2).

$$\Pi_t = \Pi_e + \Pi_s + \Pi_d + \Pi_h \tag{2.2}$$

La taille des différents tampons est définie en fonction des paramètres du FIR. Ainsi, le tampon d'entrée (Π_e) nécessite la quantité de mémoire caractérisée par le nombre d'échantillons par bloc (N_s) et le nombre de bits de quantifications par échantillons (N_q) comme présenté dans l'équation(2.3). Ces caractéristiques varient selon l'application à laquelle est destiné le filtre ou selon la plateforme. Par exemple, le nombre de bits de quantification peut être déterminé en fonction du rapport de puissance entre le signal et le bruit de quantification (SQNR) admissible par l'application. Un autre facteur qui influence le choix de N_q est la largeur des opérandes supportées par les ALU du DSP.

$$\Pi_e = N_s \times N_q \tag{2.3}$$

La quantité de mémoire utilisée par le tampon de sortie (Π_s) est aussi en fonction des mêmes paramètres que le tampon d'entrée puisque le FIR produit un résultat par échantillons d'entrée. De plus, la multiplication matérielle de deux nombres binaires de n-bits produit un résultat d'une largeur de 2 n-bits(Stalling, 2003). Alors la taille du tampon Π_s est donnée par l'équation(2.4).

$$\Pi_s = 2 \times N_s \times N_q \tag{2.4}$$

Le troisième tampon nécessaire dans un filtre FIR est celui de la ligne à délais. Une ligne à délais, est un mécanisme qui retarde et emmagasine les échantillons selon un délai et une profondeur préétablie ou configurable. Le fonctionnement d'une ligne à délai peut être

comparé une pile *first input, first output* (FIFO) dont chacun des éléments est accessible en lecture (voir Figure 2.3). La profondeur requise de la ligne pour son utilisation dans un FIR est égale aux nombre de coefficients du filtre (N).

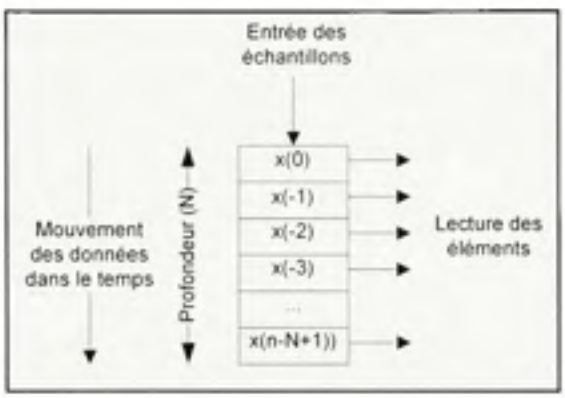

Figure 2.3 Fonctionnement d'une ligne à délais.

Donc, la quantité de mémoire requise par le tampon Π_d est égal au nombre de coefficients (N) multiplié par le nombre de bits de quantification des échantillons d'entrée (N_q) (équation(2.5)).

$$\Pi_d = N \times N_q \tag{2.5}$$

Finalement, le dernier tampon est celui qui sert à emmagasiner les valeurs des coefficients (Π_h). Sa taille dépend du nombre de coefficients et de la quantification (équation(2.6)).

$$\Pi_h = N \times N_q \tag{2.6}$$

2.1.3 Filtre FIR complexe

Le filtre FIR complexe est spécifié par la même équation générale que le filtre FIR réel (équation(2.1)), excepté que les échantillons et les coefficients sont des nombres complexes.

Le principal type d'applications du filtre complexe est celui de l'égalisation de canal de communication, une technique de filtrage conçue pour réduire le brouillage intersymbole (ISI) (Sklar, 2001). Les échantillons et les coefficients sont complexes lorsque les systèmes de transmission de données utilisent une modulation d'amplitude en quadrature (QAM) où l'information est représentée par deux signaux déphasés à 90 degrés (voir Figure 2.4). Ainsi, on se retrouve avec le double d'informations à traiter pour chacun des échantillons ce qui augmente la quantité de mémoire nécessaire au fonctionnement du filtre.

Figure 2.4 Exemple de modulateur/démodulateur QAM.

Il faut aussi noter que les opérations comme la multiplication des nombres complexes requière plus de ressources de traitement qu'une multiplication réelle. En ce qui a trait à la taille des tampons du filtre, elle sera alors le double, puisque le FIR complexe reprend la même équation générale que le FIR réel. Donc, un filtre FIR complexe devrait utiliser au moins deux fois plus de mémoire qu'un filtre réel.

2.1.4 Filtre égaliseur adaptatif LMS

Le filtre FIR complexe présenté dans la section précédente peut être utilisé comme égaliseur lorsque la réponse du canal est invariable dans le temps. Ainsi, dans le cas d'un canal avec une réponse variant lentement dans le temps, on emploie une égalisation adaptative qui suit ces variations (Sklar, 2001). L'adaptation est implémentée par un mécanisme qui ajuste les

valeurs des coefficients du filtre en fonction de l'erreur à la sortie du filtre. Le filtre adaptatif reprend la structure de base du FIR en y ajoutant un algorithme qui ajuste les valeurs des coefficients (Figure 2.5).

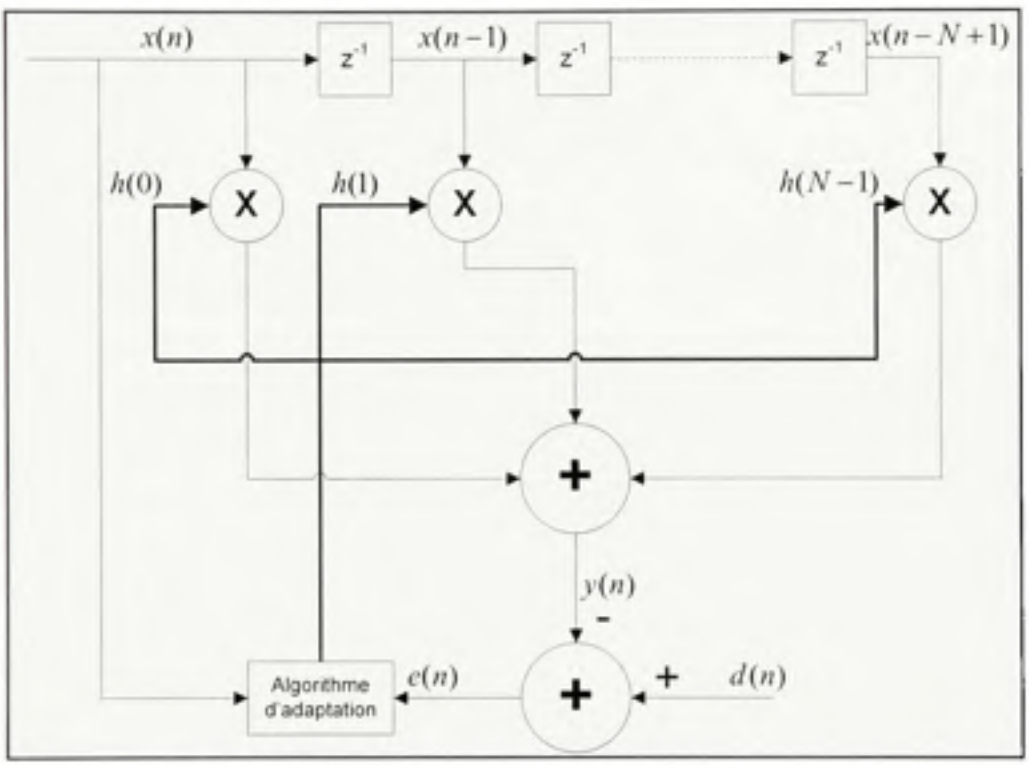

Figure 2.5 Structure générale d'un filtre adaptatif.

L'erreur est la différence entre la sortie du filtre et la sortie désirée ($d(n)$). Les valeurs de sortie désirée peuvent soit provenir d'une courte séquence d'entraînement connue à l'avance par le récepteur, on appelle cela le mode d'entraînement (*training mode).* L'autre méthode se sert d'un détecteur de symbole à la sortie du filtre, elle est appelée *decision directed mode.* Le but de l'algorithme d'adaptation est de réduire l'erreur ($e(n)$) en changeant les poids des coefficients.

2.1.4.1 L'algorithme LMS

Il existe plusieurs algorithmes d'adaptation, celui des moindres carrés moyens (LMS), étudié dans le présent projet, est l'un des plus utilisés en pratique (Farhang-Boroujeny, 1998). Il tire son nom du fait qu'il minimise l'erreur en adaptant les coefficients selon l'optique des carrés moyens. L'algorithme fonctionne en trois phases :

1. Filtrage;
2. Estimation de l'erreur;
3. Adaptation des coefficients.

L'étape du filtrage est identique au traitement d'un filtre FIR. L'estimation de l'erreur s'effectue en faisant la différence entre un symbole désiré et la sortie du filtre (équation(2.7)).

$$e(n) = d(n) - y(n) \tag{2.7}$$

L'adaptation des coefficients s'effectue selon l'équation récursive (2.8), appelée descente du gradient (Farhang-Boroujeny, 1998),

$$\mathrm{h}(n+1) = \mathrm{h}(n) - \mu \nabla e^2(n) \tag{2.8}$$

Où :

- $\mathrm{h}(n+1)$ est le vecteur contenant les nouvelles valeurs des coefficients;
- $\mathrm{h}(n)$ est le vecteur contenant les valeurs actuelles des coefficients;
- $\nabla e^2(n)$ est l'estimation du gradient de l'erreur;
- μ est le pas d'ajustement de l'algorithme.

On calcule l'estimation du gradient de l'erreur selon l'équation(2.9).

$$\nabla e^2(n) = -2e(n)\mathrm{x(n)} \tag{2.9}$$

Où $e(n)$ est l'erreur calculée précédemment et $\mathrm{x}(n)$ le vecteur contenant les échantillons à l'entrée du filtre. Ainsi, on obtient l'équation (2.10) en combinant les équations (2.8) et (2.9)

.

$$\boxed{\mathrm{h}(n+1) = \mathrm{h}(n) + 2\mu e(n)\mathrm{x}(n)} \tag{2.10}$$

2.1.4.2 Filtre LMS complexe

Le filtre LMS complexe est utilisé pour les mêmes types d'applications que le filtre FIR complexe. En effet, dans les systèmes de transmissions sans fil, la réponse du canal peut varier dans le temps et un filtre égaliseur LMS est souvent utilisé dans de tels cas. Donc, le type de filtre adaptatif implémenté pour vérifier la méthodologie d'estimation est un filtre complexe puisque le domaine d'application visé par le projet OPERA est celui des communications sans fil.

2.1.5 Décodeur de Viterbi

2.1.5.1 Les codes convolutionnels

Les techniques de codage ont été développées afin d'améliorer les performances des systèmes de communications opérant à travers un canal qui dégrade le signal. Le codage consiste en un traitement numérique des données qui ajoute des bits de redondance selon un certain patron ou code. Dans les systèmes de communications sans fil, les codes les plus courants sont les codes convolutionnels (Hendrix, 2002). Un encodeur convolutionnel traite les données de manière sérielle : pour chaque bit d'informations en entrée, un certain nombre de bits est produit. Le rapport du nombre de bits en entrée, sur le nombre de bits produits (symboles) d'un encodeur définit son taux de codage R (équation(2.11)).

$$R = \frac{n_{in}}{n_{out}} \tag{2.11}$$

Les symboles codés sont générés à partir d'un symbole à l'entrée et d'une séquence des bits précédents contenus dans un registre à décalage. La contrainte de longueur K de l'encodeur détermine le nombre de bits dans le registre. La configuration de l'encodeur est souvent représentée par des polynômes générateurs qui indiquent lesquels des bits contenus dans le registre sont utilisés. Par exemple, pour l'encodeur représenté par la Figure 2.6, les polynômes générateurs sont $g_1(X)$ pour u_1 et $g_2(X)$ pour u_2 (Sklar, 2001).

$$g_1(X) = 1 + X + X^2 \tag{2.12}$$

$$g_2(X) = 1 + X^2 \tag{2.13}$$

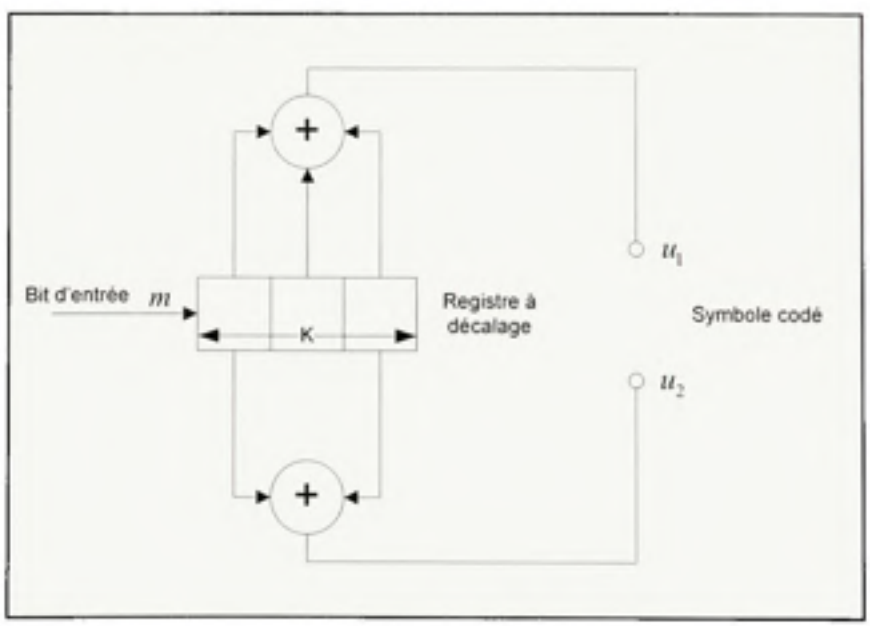

Figure 2.6 Encodeur convolutionnel avec un taux 1/2 une contrainte K = 3.

2.1.5.2 Diagramme en treillis

La structure récursive d'un encodeur convolutionnel permet d'assimiler son fonctionnement à une machine à états finis. En effet, il est possible de prédire la sortie de l'encodeur en fonction des valeurs passées mémorisées dans le registre et de la valeur du bit à l'entrée. Les valeurs passées représentent l'état actuel de la machine et la valeur du bit d'entrée détermine la transition vers l'état suivant. Ainsi, pour un encodeur avec une contrainte K le nombre d'états possibles est défini par l'équation (2.14). Donc, pour l'encodeur de la Figure 2.6, le nombre d'états de la machine équivalente est 4 états (équation (2.15)).

$$n_{états} = 2^{K-1} \tag{2.14}$$

$$2^{3-1} = 4 \tag{2.15}$$

Afin de représenter les transitions possibles en fonction de l'état et de l'entrée de l'encodeur, on peut utiliser un diagramme d'états ou en treillis. Ce dernier possède l'avantage d'ajouter l'évolution dans le temps des états par rapport à l'arrivée de nouveaux symboles à l'entrée du décodeur. Le diagramme en treillis exploite la structure cyclique de l'encodeur convolutionnel où chacune des transitions admissibles est représentée par une flèche. Le diagramme en treillis de l'encodeur décrit précédemment est présenté par la Figure 2.7. Les états correspondent aux nœuds du diagramme et les symboles encodés aux poids des liaisons. Pour utiliser le treillis afin décoder une série de symboles, on le parcourt en choisissant la transition concordant le plus avec l'entrée.

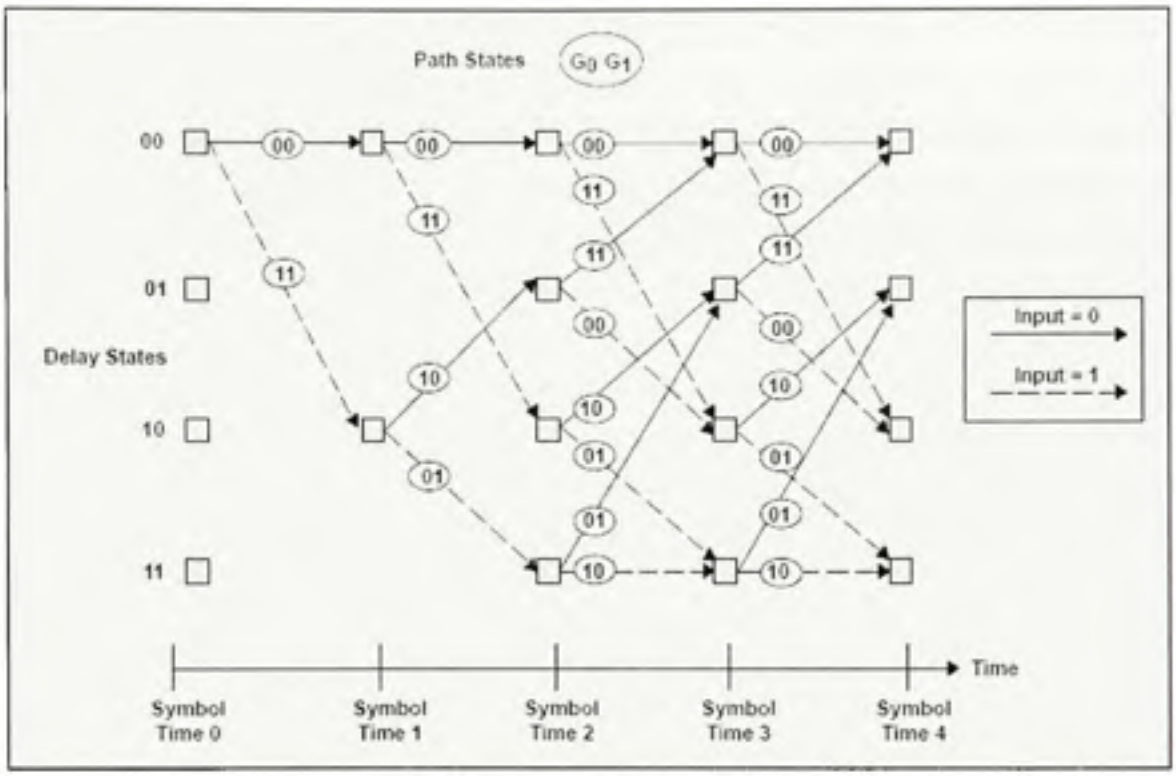

Figure 2.7 Diagramme en treillis d'un encodeur avec K = 3 et un taux de ½.

Tiré de Hendrix (2002, p. 6)

2.1.5.3 Algorithme de Viterbi

Le décodage des codes convolutionnels requiert une grande puissance de calcul lorsque l'on examine toutes les possibilités de chemins à travers le treillis. L'algorithme proposé par Viterbi (VA) (Viterbi, 1967), tire partie de la structure particulière du treillis et identifie lequel des chemins possibles entrant dans un état est le plus similaire au symbole reçu. Le VA réduit donc la charge de calculs en éliminant les chemins les plus improbables. L'algorithme se divise en deux séquences : la mise à jour des métriques de similitudes et la routine de retraçage (Hendrix, 2002).

2.1.5.3.1 Mise à jour des métriques

La première étape du VA est le calcul des métriques de similarité qui évalue la distance locale, pour chacun des états au temps t, entre le symbole reçu et les valeurs attendues pour chaque chemin possible (« path states »). La distance locale est calculée avec la distance de Hamming dans le cas de décodeurs avec des entrées en décision dure ou avec la distance euclidienne pour des entrées en décision pondérée. Habituellement, on s'assure que le décodeur commence dans un état connu à l'avance afin de réduire le nombre d'alternatives. Après avoir calculé les distances des chemins possibles, le VA additionne chacune des distances à la distance accumulée. Ensuite, il compare les distances accumulées et on détermine la plus petite que l'on sauvegarde avec le chemin correspondant dans la matrice de retraçage. La matrice de retraçage contient le meilleur branchement à partir d'un état. Le décodeur procède de cette façon pour chacun des symboles jusqu'à la fin d'une série de symboles reçus ou de la trame.

2.1.5.3.2 Retraçage

Lorsque le décodeur atteint la fin d'une trame, il commence l'étape du retraçage des symboles décodés. Généralement, des zéros sont insérés à la fin d'une trame afin de retourner l'encodeur dans l'état zéro. Il faut insérer $K-1$ zéros pour retourner à l'état zéro, cela permet au décodeur de commencer le retraçage dans un état connu. Le décodeur examine la matrice de retraçage pour déterminer l'état précédent et le symbole décodé associé avec la transition.

2.1.5.4 Implémentation d'un décodeur de Viterbi

L'implémentation du VA sur un DSP nécessite beaucoup de ressources tant au niveau du traitement qu'au niveau de la mémoire. Le traitement s'effectue au niveau d'une trame qui contient un nombre de symboles déterminé en fonction de l'application. De plus, les DSP

incorporent souvent des instructions spécialisées au niveau matériel afin d'accélérer le VA. Le pseudo-code de la Figure 2.8 présente une implémentation générale des principales étapes du VA.

```
for each frame:
   {
   Initialize metrics
   for each symbol:
      {
      Metric Update or Add-Compare-Select (ACS)
      For each delay state:
         {
         Calculate local distance of input to each possible path
         Accumulate total distance for each path
         Select and save minimum distance
         Save indication of path taken
         }
      }
   Traceback
   for each bit in a frame (or for minimum # bits):
      {
      Calculate position in transition data of the current state
      Read selected bit corresponding to state
      Update state value with new bit
      }
   reverse output bit ordering
   }
```

Figure 2.8 Pseudo code pour l'algorithme de Viterbi.

Tiré de Hendrix (2002, p. 6)

2.1.5.4.1 Utilisation des ressources par le VA

Comme on peut le constater en examinant le pseudo code, la mise à jour des métriques est équivalente à une opération d'addition-comparaison-sélection (ACS). Cette opération est souvent implémentée en matériel sur les DSP utilisés dans les applications de télécommunications. Le reste de l'algorithme consiste en des manipulations de matrices de données. Or, l'un des désavantages de l'algorithme de Viterbi est la grande quantité de mémoire nécessaire pour contenir les matrices de métriques et de retraçage. En fait, la quantité de mémoire requise par un décodeur de Viterbi croît exponentiellement avec la

longueur de contrainte K (Sklar, 2001). Certaines modifications de l'algorithme comme les techniques de fenêtrage (« windowing ») réduisent la quantité de mémoire en travaillant sur une portion de la trame à la fois. Par contre, ces techniques augmentent la charge au niveau du traitement. En général un décodeur de Viterbi nécessite les tampons contenant les informations suivantes :

- métriques accumulées;
- matrice de retraçage;
- trame de symboles reçus;
- bits décodés.

La taille des deux premiers tampons dépend du nombre d'états de l'encodeur qui est spécifié par la contrainte K (équation(2.14)). Les métriques accumulées d'un symbole ($\alpha^{e(k+1)}$) pour chaque état e sont calculées à partir des métriques du symbole précédent ($\alpha_b^{e(k)}$ et $\alpha_a^{e(k)}$) et des métriques de branchements ($\gamma_b^{e(k)\to e(k+1)}$ et $\gamma_b^{e(k)\to e(k+1)}$) (équation(2.16)).

$$\alpha^{e(k+1)} = \max\left(\alpha_b^{e(k)} + \gamma_b^{e(k)\to e(k+1)}, \alpha_a^{e(k)} + \gamma_a^{e(k)\to e(k+1)}\right) \tag{2.16}$$

Alors, il n'est pas nécessaire de mémoriser les métriques accumulées pour tous les symboles d'une trame durant tout le processus de décodage. Seulement deux tampons nécessaire soit un pour les anciennes métriques et un pour les nouvelles, leur taille est définie par l'équation(2.17).

$$\boxed{\Pi_\alpha = n_{états} \times N_q} \tag{2.17}$$

Les métriques de branchements sont la distance entre le symbole reçu et la sortie attendue de l'encodeur selon son état. Ainsi, le nombre de métriques de branchements est restreint pour

un symbole reçu puisqu'il dépend du nombre de combinaisons possible à la sortie de l'encodeur convolutionnel. Les métriques de branchements pour un décodeur à décision pondérées sont calculées selon la distance euclidienne (équation (2.18))(S.Cormier, 2006).

$$\gamma^{e(k)\to e(k+1)} = \sum_{r=0}^{R-1} \left(u_r(k) - g_r(k)\right)^2 \tag{2.18}$$

Où :

- $\gamma^{e(k)\to e(k+1)}$: est la métrique de branchement pour une branche allant de l'état $e(k)$ vers l'état $e(k+1)$;
- R : est la constante définie par le taux de codage $1/R$;
- $u_r(k)$: est un symbole pondéré d'un symbole-codé composé de r symboles reçu au temps k ;
- $g_r(k)$: est le symbole à la sortie de l'encodeur contraint par le treillis selon la transition de l'état $e(k)$ vers l'état $e(k+1)$.

Lors du processus de décodage, le VA retrace l'état précédent de l'encodeur en fonction de l'état dans lequel il se situe et de la transition sélectionnée, fonction de maximisation, lors du calcul des métriques accumulées (équation(2.16)). Le décodeur doit donc garder en mémoire la transition sélectionnée pour chacun des états et des symboles d'une trame afin d'effectuer le décodage-retraçage. Ainsi, pendant le calcul des métriques, une matrice de retraçage est créée et contient la transition sélectionnée. Puisqu'il n'y seulement deux transitions possibles par état, alors un bit est nécessaire pour indiquer celle choisie. La taille de la matrice de retraçage est déterminée en fonction du nombre d'états et du nombre de symboles par trame (équation(2.19)).

$$\boxed{\Pi_R = n_{états} \times n_{sym}} \tag{2.19}$$

Le tampon contenant des symboles reçus a une taille égale à la longueur d'une trame et au nombre de bits de quantification d'un symbole (équation(2.20)).

$$\boxed{\Pi_{sym} = n_{sym} \times N_q} \tag{2.20}$$

La taille du tampon des bits décodés est égale on nombre de symbole reçu divisé par la constante du taux de codage $1/R$ (équation(2.21)).

$$\boxed{\Pi_{dec} = \frac{n_{sym}}{R}} \tag{2.21}$$

2.1.6 Maximum et addition vectoriel

Les deux dernières fonctions étudiées dans le cadre de cette recherche sont des opérations vectorielles fondamentales. En effet, ces opérations sont des composantes d'opération plus complexes comme des décodeurs, modulateurs ou d'autres fonctions de traitement du signal. Elles ont été aussi choisies parce qu'elles sont simples à implémenter sur un DSP, ainsi elles permettent d'effectuer une première vérification de la méthodologie. De plus, l'utilisation des ressources mémoires est facilement calculable puisque les deux fonctions utilisent seulement des tampons d'entrées et sorties. Les tailles de ceux-ci sont définies par rapport au nombre d'éléments qui constituent un vecteur.

$$\boxed{\Pi_e = n_{éléments} \times N_q} \tag{2.22}$$

$$\boxed{\Pi_s = n_{éléments} \times N_q} \tag{2.23}$$

2.2 Conclusion

Dans ce chapitre les différents algorithmes qui seront étudiés dans le cadre de ce projet ont été décrits et analysés. Les applications visées appartiennent au domaine des télécommunications car celles-ci représentent une part importante des algorithmes de traitement de signaux. De plus, la complexité inhérente aux applications de

télécommunications constitue un défi qui impose le développement de nouvelles méthodologies conception. Ainsi, toutes ces applications nécessitent un tampon d'entrée et de sortie dont la taille est une fonction du nombre d'échantillons à traiter à l'intérieur d'un intervalle de temps fixe. L'écriture et la lecture de ces tampons à l'intérieur d'un intervalle de temps impose une bande passante, vers la mémoire, suffisante pour respecter cette contrainte. De plus, la majorité des applications utilise aussi des tampons temporaires durant le traitement des données et leur taille est en fonction de l'algorithme implémenté. Par exemple, le décodeur de Viterbi requiert des tampons de grande taille pour ses métriques de branchement et sa matrice de retraçage. La taille des tampons temporaires peut dans certains cas, sans optimisation, dépasser la quantité de mémoire disponible sur la plateforme. Donc, pour les applications de traitement de signal, la gestion des ressources mémoires est critique afin de respecter les spécifications sur la plateforme choisie. Alors, dans le prochain chapitre nous étudierons la plateforme matérielle utilisée lors de ce projet, le circuit Vocallo d'Octasic inc. C'est une plateforme multi-noyaux conçues pour des applications de transfert de la voie ou de signal vidéo sur IP et réseaux de téléphonie sans-fil(Octasic Inc, 2008).

CHAPITRE 3

ARCHITECTURE VOCALLO

Le développement de nouvelles applications numériques du signal, comme le standard UMTS pour la téléphonie cellulaire de 3^{e} génération, a permis l'introduction par les fabricants de plusieurs modèles de DSP. De plus, avec une augmentation des performances en nombres entiers des microprocesseurs de 50-55% par année (Säckinger, 1997), il est maintenant possible de développer des applications très complexes. La première moitié de cette augmentation est due à la réduction de la taille des transistors et l'autre partie aux améliorations des circuits et de l'architecture (Säckinger, 1997). Donc, la connaissance de l'architecture du processeur sur lequel sera implémentée l'application est primordiale pour obtenir une bonne estimation des performances. Ce chapitre présente l'architecture du Vocallo ainsi que l'environnement de développement logiciel qui sont les éléments de la plateforme visée pour l'application de la méthodologie développée.

3.1 Spécifications de l'architecture Vocallo

Le processeur Vocallo à multi-noyaux est conçu pour les applications de traitement de la voix, du vidéo et des données sur IP (Octasic Inc, 2008). Dans cette section nous étudierons les spécifications du Vocallo afin de développer la méthodologie d'estimation la mieux adaptée pour cette plateforme. Puisque le Vocallo est un puce multi-noyaux, nous commencerons par examiner son architecture haut-niveau pour ensuite entrer dans les détails d'un noyau.

3.1.1 Architecture haut-niveau

L'architecture haut-niveau du Vocallo comprend les différentes interfaces d'entrées-sorties du circuit, les communications inter-noyaux, et la mémoire externe. La Figure 3.1 présente

graphiquement les principaux composants de l'architecture. Celle-ci se divise en trois blocs physiquement distincts :

- le circuit de traitement Vocallo;
- le circuit d'interface E/S;
- la mémoire externe.

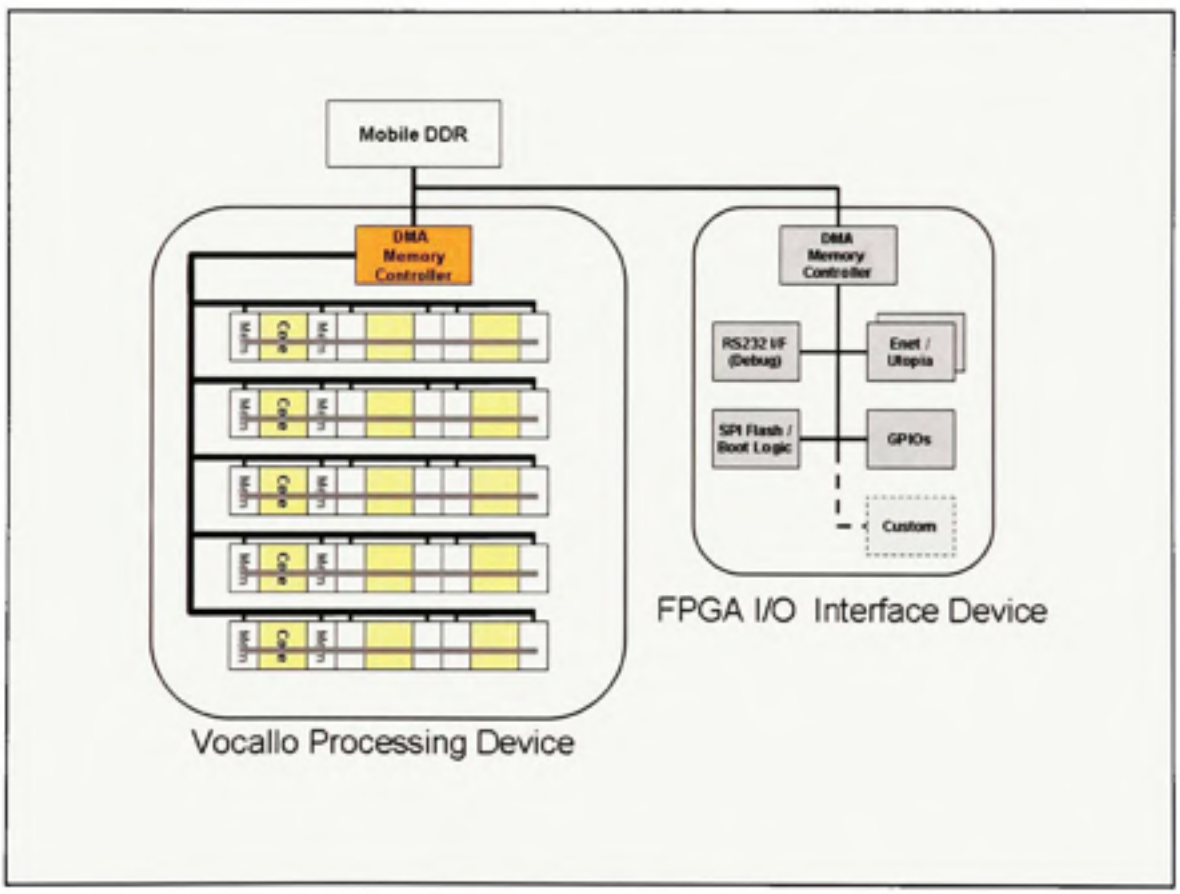

Figure 3.1 Architecture haut niveau du Vocallo.

3.1.1.1 Architecture interne du Vocallo

Le circuit Vocallo est constitué de 15 noyaux identiques et d'un contrôleur d'accès mémoire. C'est un processeur parallèle à flot d'instructions et à flot de données multiples (MIMD) homogène. Un processeur MIMD est un processeur parallèle constitué de plusieurs unités de traitement (UT) interconnectées par un support de communication (Säckinger, 1997). Chaque noyau est un processeur de traitement numérique autonome, l'architecture en détail d'un noyau est présentée dans la section 3.1.2. Les noyaux sont disposés selon une matrice de 5 lignes par 3 colonnes. Il est possible de désactiver les noyaux de la colonne du centre pour partager leur mémoire avec les deux noyaux adjacents. Ainsi les deux noyaux des colonnes extérieures peuvent utiliser la moitié de la mémoire du noyau du centre.

3.1.1.2 Communications inter-noyaux

Les communications inter-noyaux du Vocallo sont composées de signaux de contrôle et d'un bus de données. Les signaux de contrôles servent à gérer les transferts de données entre les noyaux et vers la mémoire externe. Ceux-ci sont nécessaires puisque le bus de données est partagé par tous les noyaux. Le bus est relié à la mémoire externe par l'entremise du contrôleur d'accès direct à la mémoire (DMA). C'est le contrôleur qui prend en charge tous les transferts de données inter-noyaux et vers la mémoire externe. Le contrôleur DMA supporte aussi un mode de transfert spécial qui permet d'envoyer un ensemble de données simultanément à tous les noyaux. L'accès au bus de données est partagé par tous les noyaux et lorsqu'un noyau l'utilise pour envoyer ou recevoir des données, les autres noyaux n'y ont pas accès. La gestion de l'accès est assurée par un système de file d'attente du type premier arrivé, premier servi mais avec un système de priorités. La largeur du bus est de 32 bits et sa vitesse de transfert est de 1.5 GHz. Un autre aspect important de la communication inter-noyaux est que lorsqu'un noyau transfère des données sur le bus, l'exécution des instructions sur le noyau est interrompu.

3.1.1.3 Interfaces d'entrée-sorties

Les interfaces d'entrées-sorties (E/S) du Vocallo ne sont pas intégrées au circuit, elles sont dans un circuit FPGA externe. Elles sont accessibles au Vocallo par le bus de communication de la mémoire externe. Les données qui entrent ou sortent du circuit d'interfaces E/S ne peuvent être directement envoyées vers un noyau de traitement. La mémoire externe sert de tampon entre le Vocallo et les interfaces. Ainsi, un bloc de données provenant d'une entrée doit d'abord être écrit dans la mémoire externe et ensuite être lu par un ou plusieurs noyaux. Le principe est le même pour un noyau transférant des données vers une sortie. Cette particularité doit être prise en compte lors de l'analyse de l'utilisation du bus mémoire puisque cela double le nombre de transactions nécessaires pour effectuer un transfert vers une E/S.

3.1.1.4 Mémoire externe

La mémoire externe est essentielle au fonctionnement du Vocallo puisque celui-ci ne peut accéder directement à l'interface E/S. Le type de mémoire supporté par l'interface du vocallo est la Mobile DDR SDRAM. L'interface supporte une ou deux banques de mémoire ce qui fait varier la largeur du bus de données entre 32 ou 64 bits. La vitesse de transfert est de 166 MHz DDR. L'accès à la mémoire externe est géré par le contrôleur DMA du Vocallo. Le bus de données est partagé avec le circuit d'interface et celui-ci doit demander l'autorisation d'accès au contrôleur DMA du Vocallo.

3.1.2 Spécifications détaillées d'un noyau

Un noyau du circuit Vocallo est équivalent à un processeur DSP avec un jeu d'instructions complet et une mémoire locale. L'architecture d'un noyau est représentée à la Figure 3.2 et est examinée plus en détails dans cette section. C'est une architecture du type Von Neumann, c'est-à-dire que les données partagent le même espace mémoire que le code de l'application. La méthodologie proposée se concentre plus spécifiquement sur l'utilisation

des ressources par une application exécutée sur un noyau. L'architecture comprend les composantes suivantes :

- 16 unités d'exécution (ALU);
- une mémoire locale;
- des registres de données.

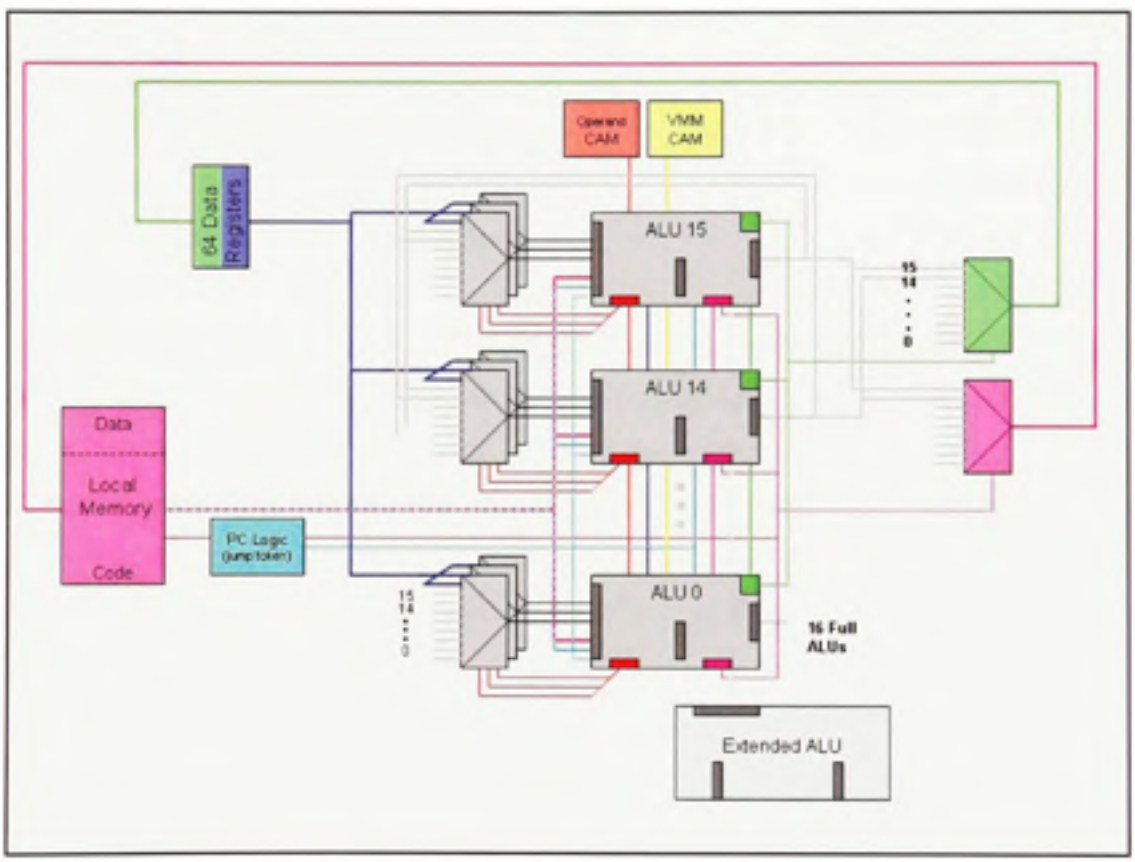

Figure 3.2 Architecture interne d'un noyau.

3.1.2.1 Les registres de données

Le jeu d'instructions du Vocallo est du type chargement-rangement. L'architecture chargement-rangement ou registre-registre, peut accéder à la mémoire uniquement grâce aux instructions de chargement et de rangement (Hennessy et Patterson, 2003). Donc, il est important d'avoir un nombre suffisant de registres de données qui sont accessibles par les ALUs pour exécuter les instructions. Chaque noyau possède 4 types de registres :

- 63 registres à usage général;
- un registre nul;
- 63 registres temporaires;
- un registre de sortie d'instructions (IOR).

Les registres à usage général ont une largeur de 32 bits et sont utilisés pour entreposer des données, des pointeurs d'adresse de données et des conditions. Le registre nul est un registre spécial dont la valeur est toujours nulle. Ensuite, les registres temporaires sont des entités logiques associées avec les bascules de sorties d'un ALU. Une valeur contenue dans un registre temporaire est valide tant qu'une nouvelle instruction n'est pas exécutée dans l'ALU. Finalement, l'IOR est un registre unique qui est utilisé par les instructions qui nécessitent un quatrième opérande.

3.1.2.2 La mémoire locale

Un noyau possède 96K octets de mémoire locale qui est utilisée pour contenir les données et le code. Lorsque cela est nécessaire, il est possible de désactiver un noyau de la colonne centrale de Vocallo pour ajouter 48 K octets aux noyaux adjacents. Les transferts de données entrant et sortant de la mémoire s'effectuent à une vitesse de 1.5 GHz. La mémoire ne possède qu'un seul port, il n'est donc pas possible d'écrire et de lire des données simultanément.

3.1.2.3 Unités d'exécutions (ALU)

Les ALU d'un noyau sont tous identiques et opèrent en parallèle. Ils partagent l'utilisation des ressources mémoires et des registres. La logique interne des ALU est asynchrone et le séquencement des instructions entre les ALU est géré par un système de fanions. L'exécution des instructions est pipelinée à travers les ALU. Cela permet d'augmenter la vitesse d'exécution des instructions lorsqu'il n'y pas de dépendance de données à l'intérieur d'une fenêtre de 16 instructions. Le jeu d'instructions supporté par les ALU est du type RISC mais certaines instructions plus complexes sont disponibles. Il y a des instructions du type « Single Instruction Multiple Data » (SIMD) qui peuvent exécuter un traitement sur plusieurs données à l'intérieur d'une seule instruction. Les registres de données de l'instruction sont sous-divisés en blocs qui constituent des données indépendantes. Ainsi, un seul registre peut contenir la partie réelle et imaginaire d'un nombre complexe. Par contre, cette méthode réduit la plage de précision de données puisque les opérations s'effectuent sur des données de taille de 16 ou 8 bits. Le nombre de chargements de données en mémoire est aussi réduit par deux par rapport à une instruction conventionnelle.

3.2 Environnement de développement

Les applications développées sur le Vocallo doivent être implémentées en utilisant des outils qui constituent l'environnement de développement (IDE) du Vocallo. Ces outils sont l'éditeur du code, le compilateur et le simulateur d'instructions. L'éditeur de code est un logiciel doté d'une interface graphique qui est utilisée pour écrire le code en langage C et assembleur. L'interface permet d'appeler le compilateur et le simulateur. Le compilateur supporte les applications écrites en langage C, assembleur ou mixte. Celui-ci est intégré à l'interface graphique de l'éditeur. C'est le compilateur qui produit le rapport d'utilisation de la mémoire qui est utilisé dans ce projet. Lorsque le code est compilé, il est ensuite possible de simuler le comportement de l'application sur le circuit avec le simulateur. Le simulateur

d'instructions (ISS) de l'IDE est un émulateur virtuel du jeu d'instructions du Vocallo. Il fournit une estimation statique du temps d'exécution d'une application sur un seul noyau en fonction du nombre d'instructions, des branchements et des dépendances de données. L'IDE est aussi doté d'un profileur qui se sert de l'ISS afin de représenter l'exécution des instructions à travers les ALU et les dépendances de données entre ceux-ci.

Dans le cadre du projet décrit dans ce document, deux versions du l'IDE ont été utilisées. La plus ancienne des versions supporte un langage C non standard et un jeu d'instructions limité alors que la nouvelle version supporte du langage C ANSI et le jeu d'instructions évolué. Il y a aussi de petites différences dans les rapports d'utilisation de la mémoire. L'ancienne version produit un rapport simple où la quantité de mémoire nécessaire au code et aux données est indiquée en bits. Cette version ajoute aussi un « overhead » la mémoire utilisée par des constantes prédéfinies par l'IDE. La nouvelle version produit un rapport qui représente le placement en mémoire des différentes instances de données et la quantité de mémoire utilisée doit être calculée manuellement. Les deux versions de l'IDE n'indiquent toutefois pas le temps de transfert des données qui entrent et sortent d'un noyau.

3.3 Impact de l'architecture sur la méthodologie

L'objectif de la méthodologie décrite dans ce mémoire est de fournir une estimation de l'utilisation de la mémoire sur la plateforme à partir d'une représentation haut niveau. Donc, une connaissance de l'architecture s'avère nécessaire. En effet, certaines particularités de la plateforme ont influencé le développement de la méthodologie. Le Vocallo est un circuit multi-noyaux avec un bus mémoire partagé et où chaque noyau possède sa propre mémoire locale. Il faut d'abord déterminer de quelle manière une application complexe est partagée sur le circuit. Ensuite, lorsque la partition de l'application est fixée, on doit pouvoir estimer la quantité de mémoire nécessaire par chaque partie de l'application sur un noyau. La taille de la mémoire sur un noyau est réduite par rapport à la taille de la mémoire externe, il faut s'assurer que la quantité de données qui entre et sort d'un noyau n'est pas excessive. De

plus, la bande passante restreinte du bus mémoire est un autre facteur à considérer dans l'implémentation d'une application. Aussi, présentement les outils fournis par la compagnie Octasic ne permettent pas encore de faire du développement sur plusieurs noyaux. Donc, en tenant compte de la complexité de la plateforme, ce projet propose d'examiner les besoins en ressources mémoires pour des applications dont la taille est restreinte et qui constituent les primitives d'une application plus complexe. Ces primitives devraient avoir une taille suffisamment réduite pour être exécutées sur un seul noyau. La méthodologie se concentrera alors sur l'utilisation des ressources mémoire propre à un noyau. Ces ressources sont donc : la mémoire locale, les registres de données et la bande passante du bus mémoire.

3.4 Conclusion

Dans ce chapitre, l'architecture du Vocallo a été examinée et son impact sur la méthodologie a été étudié. Il en est ressorti que Vocallo est une plateforme sophistiquée et que les applications devront être partitionnées afin de pouvoir exploiter toutes ses performances. De plus, l'utilisation de la mémoire étant l'objet de ce document, les principales contraintes en termes de ressources mémoires pour le Vocallo ont été identifiées à celles appartenant aux noyaux du circuit. Ces ressources sont la mémoire locale d'un noyau, ses registres et la bande passante du bus.

CHAPITRE 4

MÉTHODOLOGIES EXISTANTES

L'estimation de l'utilisation des ressources mémoires a fait l'objet de nombreuses recherches par le passé. En effet, pour les concepteurs la connaissance de l'utilisation des ressources dans les premières phases de développement constitue un atout important. Pour les applications comme celles des télécommunications qui sont dominées par le traitement de données, l'entreposage et le mouvement de celles-ci est une partie critique du processus de conception. De plus, il s'avère que la mémoire est le composant occupant le plus d'espace sur la surface d'un circuit intégré, contribuant ainsi au coût et à la consommation de puissance. Avec les contraintes de temps et de coût liés à l'introduction de nouveaux produits, il s'avère important de choisir la meilleure solution pour l'application. Dans le but d'estimer les besoins en mémoire d'une nouvelle application à partir d'une description haut niveau, plusieurs méthodes ont été développées. De plus, pour les concepteurs le choix d'une méthodologie entraine le choix d'un modèle de représentation haut niveau associé avec celle-ci. Le concepteur doit donc déterminer laquelle des méthodologies utilise la représentation la mieux adaptée à ses besoins. Dans ce chapitre, nous commencerons par présenter les métriques de performances des circuits DSP utilisées pour comparer différents circuits. Ensuite, certains modèles de représentation haut niveau existant seront comparés. Finalement, les méthodologies d'estimation existantes seront présentées.

4.1 Les métriques de performances des circuits DSP

L'ère des télécommunications numériques a permis le développement de processeurs de signal numérique, aussi connus sous l'acronyme DSP, toujours plus puissants et plus diversifiés. Avec l'arrivée de nouvelles applications et de nouveaux standards plus complexes, le choix d'une plateforme devient critique dès les premières phases du développement. Or, il s'avère que l'estimation des performances d'un DSP est un processus

ardu. Ainsi, des méthodes comme les métriques simples et les tests de performances algorithmiques ont été introduites par les concepteurs afin de quantifier les performances des DSP.

4.1.1 Les métriques simples

Tel que présentés par (Lapsley et Blalock, 1996) il existe des métriques de performance simples pour les DSP comme le débit de traitement mesuré en million d'instructions par seconde (MIPS), le million d'opérations par seconde MOPS et le nombre de multiplications-additions par seconde (MACS). Le MIPS est le plus utilisé mais il n'est pas recommandé pour comparer deux processeurs entre eux puisque la quantité de travail faite par instruction varie en fonction de l'architecture des DSP. Le MIPS est par contre utile pour comparer deux applications ou deux implémentations, lorsque le DSP utilisé est le même. Le MOPS est moins utilisé car la définition de ce qu'est une opération peut varier entre deux processeurs. Le MACS est une métrique plus spécifique aux DPS car les multiplications-additions sont des opérations très répandues dans les applications de filtrage, de corrélation et de multiplication de vecteurs. Cette métrique est surtout utilisée par les manufacturiers pour vanter les performances de leurs circuits. La grande faiblesse de ces métriques est qu'elles ne permettent pas de mesurer les performances secondaires comme l'utilisation de la mémoire et la consommation de puissance. Or, dans les applications de communications numériques portables, la puissance et la mémoire sont souvent limitées.

4.1.2 Les tests de performance algorithmiques

Afin de palier aux lacunes des précédentes méthodes, (Lapsley et Blalock, 1996) ont défini une nouvelle métrique ou méthodologie appelée « Algorithm kernel benchmarking ». Les « Algorithm kernels » ou noyaux algorithmiques sont des blocs mathématiques constituant les fonctions de bases des applications de traitements numériques de signaux. Les transformées de Fourier rapides (TFR), les additions de vecteurs et les filtres numériques sont

cités comme exemples de noyaux algorithmiques. Le Tableau 2.1 présente des fonctions utilisées par les auteurs comme tests de performance. Selon les auteurs, les noyaux algorithmiques possèdent plusieurs avantages comme test de performance (« benchmark ») :

- la pertinence;
- la facilité de spécification;
- l'optimisation;
- la facilité d'implémentation.

Les noyaux sont pertinents, car ils peuvent être sélectionnés en fonction de l'application visée et ils sont les composants les plus exigeants en puissance de calcul. Les noyaux algorithmiques ont une taille modeste donc, leurs spécifications sont limitées ce qui permet de choisir le bon type d'implémentation. De plus, étant donnée leur taille réduite, il est facile d'obtenir du code optimal en langage assembleur et ce dans un temps raisonnable. Ensuite, il faut mesurer le temps d'exécution d'un noyau sur le processeur. Les auteurs suggèrent d'utiliser un logiciel de simulation d'instructions qui permet de calculer le nombre de cycles d'exécution d'un algorithme. Pour ce qui est des mesures de puissance consommée, il faut utiliser une carte de développement matérielle avec le processeur visé.

4.1.3 Profilage de l'application

Afin de prévoir les performances d'une application sur un processeur visé (Lapsley et Blalock, 1996) utilisent le profilage de l'application. Il est important de savoir comment les noyaux algorithmiques sont utilisés dans l'application pour obtenir des estimations plus complètes. Le profilage d'une application consiste à déterminer le nombre d'exécutions des sous-sections d'une application. Cela permet aux concepteurs d'estimer l'importance relative de chacun des tests de performances algorithmiques d'une application (Lapsley et Blalock, 1996). Finalement, on peut estimer les performances d'un processeur pour une application en multipliant le temps d'exécution d'un test de performances par leur nombre d'exécutions.

4.2 Modèles de représentation des applications

Lors de la conception d'une application, les concepteurs utilisent souvent plus d'un modèle de représentation. Ces modèles sont utilisés au cours des différentes phases de la conception et le passage entre les différents niveaux d'abstraction de la description s'effectue de manière semi-automatisée ou manuelle. Dans les domaines du traitement des signaux et des télécommunications numériques, l'implémentation d'une application dans un circuit DSP exige l'utilisation d'un langage de description bas niveau tel que le langage C et ou l'assembleur. Or, la complexité des applications étudiées demande un effort considérable aux concepteurs lors de l'implémentation afin d'obtenir les meilleures performances. Par exemple, selon (Sangiovanni-Vincentelli, 2007) les téléphones cellulaires d'aujourd'hui contiendraient plus d'un million de lignes de code. Il est à noter que les modèles de description haut niveau utilisés servent seulement à valider les spécifications des algorithmes avec des simulations dans des logiciels tels que Matlab.

4.2.1 Boucles imbriquées multidimensionnelles

Les algorithmes de traitement de signaux effectuent des opérations répétitives sur des données. C'est pourquoi il est possible de décrire ces algorithmes avec des boucles imbriquées multidimensionnelles aussi appelées spécifications affines (Florin, Hongwei et Ilie, 2007). Une spécification affine est une description haut niveau qui caractérise un algorithme sous forme de pseudo-code contenant des boucles imbriquées, des opérateurs conditionnels et des tableaux de données multidimensionnels. La Figure 4.1 présente un exemple de spécification affine tel que décrit par (Florin, Hongwei et Ilie, 2007). Ce type de spécification est utilisé par les méthodes d'estimation scalaire et non scalaire décrites dans la section 4.3 pour être transformé en graphe de contrôle et de flux de données (CDFG). Cette description n'a pas été retenue pour le projet puisque le logiciel Matlab/Simulink a été choisi comme représentation principale des applications étudiées.

4.2.2 Le graphe de contrôle et de flux de données

Le CDFG tel que présenté par (Amellal et Kaminska, 1993) est un graphe où « chaque nœuds peux être un nœud d'opération ou un nœud de contrôle comme un branchement, une boucle and une fin de boucle ». Les liens entre les nœuds représentent les transferts de données entre les opérateurs. Un CDFG permet donc de représenter les différentes dépendances de données entre les opérations d'une application. De plus, la représentation sous forme de graphe permet de aussi de déterminer le dégré de parallélisme d'une application. Des outils comme le compilateur SUIF peuvent générer un CDFG à partir du code C d'une application. Le CDFG n'est pas utilisé dans la méthodologie d'estimation des ressources mémoires décrite dans ce document, mais il est utilisé dans d'autres parties du projet OPERA.

```
T[0] = 0 ;                  // A[10][529] : input
for ( j=16 ; j<=512 ; j++ )
{ S[0][j-16][0] = 0 ;
  for ( k=0 ; k<=8 ; k++)
    for ( i=j-16 ; i<=j+16 ; i++ )
      S[0][j-16][33*k+i-j+17] = A[4][j] - A[k][i]@16
                              + S[0][j-16][33*k+i-j+16] ;
  T[j-15] = S[0][j-16][297] + T[j-16] ;
}
for( j=16 ; j<=512 ; j++ )
{ S[1][j-16][0] = 0 ;
  for( k=1 ; k<=9 ; k++)
    for( i=j-16 ; i<=j+16 ; i++ )
      S[1][j-16][33*k+i-j-16] = A[5][j] - A[k][i]@32
                              + S[1][j-16][33*k+i-j-17] ;
  T[j+482] = S[1][j-16][297] + T[j+481] ;
}
out = T[994];               // out : output
```

Figure 4.1 Exemple de spécification affine.

Tiré de Florin, Hongwei et Ilie (2007, p. 448)

4.2.3 Matlab/Simulink

Les modèles Matlab et Simulink comme représentation haut niveau d'une application sont souvent utilisés afin de valider les spécifications d'une application. En effet, avec une librairie de fonctions fréquemment rencontrées dans les applications de traitement de signaux, Matlab s'avère un outil efficace de simulation. Par contre, aucune méthodologie d'estimation n'utilise des modèles Matlab comme représentation haut-niveau. Alors, un modèle Matlab utilisé lors de la validation doit être converti vers une représentation intermédiaire afin de pouvoir utiliser une méthodologie existante.

4.2.4 Génération d'un format de représentation intermédiaire sous forme de CDFG

La modélisation d'application de traitement numérique du signal est souvent ardue, du fait de la complexité inhérente de celles-ci. Dans l'environnement Matlab les développeurs profitent de la puissance du langage qui facilite la création d'algorithmes effectuant des opérations sur des données présentées sous-forme de matrices et de vecteurs. De plus, l'environnement Simulink permet de valider ces algorithmes rapidement en intégrant des scripts en langage Matlab à un modèle Simulink. En effet, Simulink est un puissant outil de validation car sa librairies de blocs contient plusieurs modèles de sources de données, de modèles de canaux et d'analyse des résultats. Il est aussi plus aisé de visualiser une application dans son ensemble grâce à l'interface graphique. La méthodologie globale de modélisation du projet OPERA se base sur les modèles Simulink. Or, ceux-ci sont souvent constitués d'au moins un script Matlab. Il s'avère donc important de pouvoir intégrer les scripts dans la méthodologie globale.

Les méthodologies d'estimation sont basées sur l'utilisation de la représentation de l'application en graphe de contrôle et de flux de données (CDFG). La Figure 4.2 présente les étapes du processus de génération du CDFG à partir d'un bloc « Embedded Matlab

Function » (EMF). Les EMF sont présentés dans la section 5.1.3.2 de ce document. Les blocs jaunes représentent ce qui est fonctionnel et les blocs bleus ce qui est en cours de réalisation dans le projet OPERA. La première étape du processus consiste à générer du code C ANSI générique à partir d'un bloc EMF. L'outil Real-Time Workshop (RTW) de Simulink est une extension qui offre la possibilité de générer du code C automatiquement à partir d'un modèle ou d'un sous-système. Ainsi, on peut configurer RTW pour générer du code pour un bloc EMF contenant une fonction qui sera implémentée sur la plateforme. Par contre, le code généré par RTW n'est pas optimisé pour la plateforme Vocallo et les performances sont dépendantes de la puissance du compilateur. Ainsi, le compilateur C du Vocallo actuellement disponible ne supporte pas les instructions spécialisées ce qui limites les performances. Mais cela ne limite pas son utilisation dans des méthodologies d'estimations puisque celles-ci travaillent à partir du graphe et non à partir du code compilé. Un logiciel de génération automatique du graphe à partir du code C, basé sur le compilateur SUIF, est cours de développement. SUIF est un logiciel d'analyse haut niveau de programmes écrient en C et Fortran (Gerald Aigner, 1999). Ensuite, ce graphe pourra être utilisé par des méthodologies d'estimation évoluées présentées dans les sections suivantes.

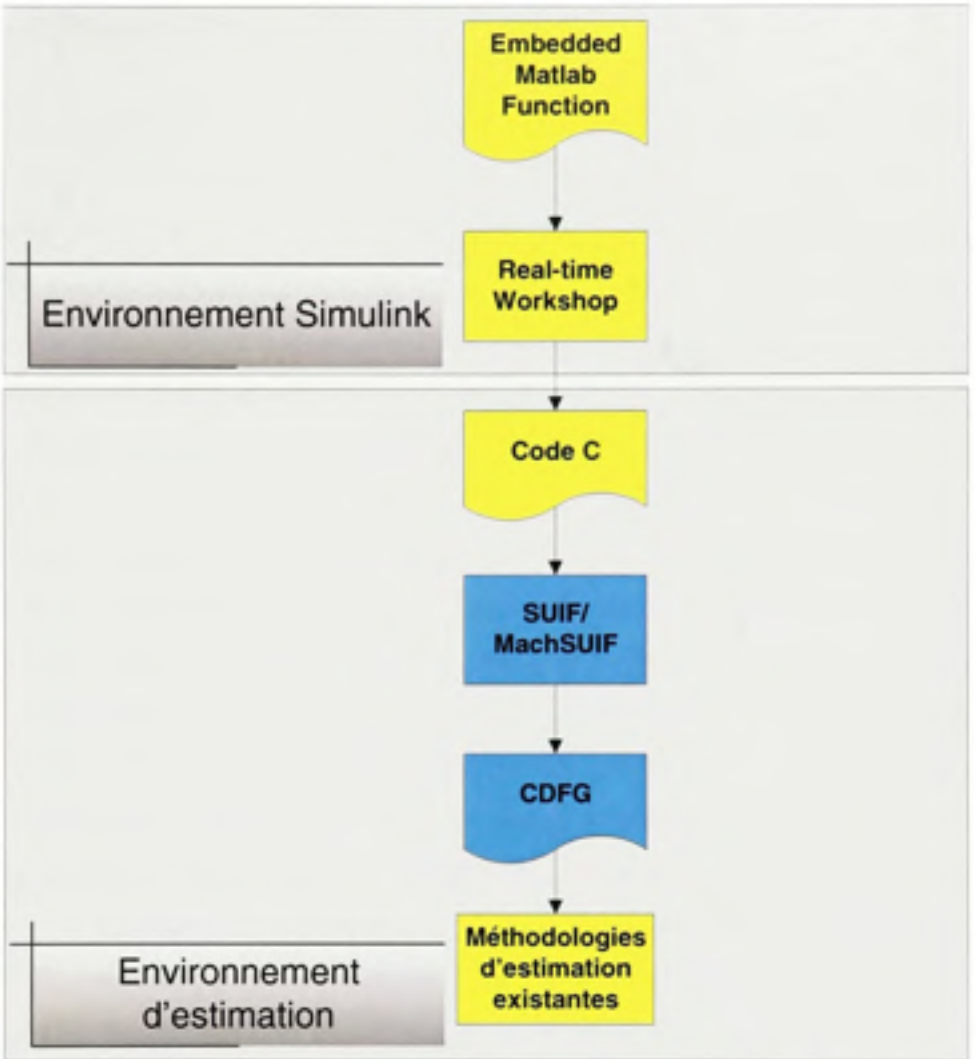

Figure 4.2 Processus de génération du CDFG.

4.3 Méthodes d'estimations utilisant la représentation CDFG

Les méthodologies présentées dans cette section estiment l'utilisation des ressources mémoire d'une application à partir d'une représentation intermédiaire sous forme de graphe de contrôle et de flux de données (CDFG).

4.3.1 Méthodes scalaires

La majeure partie des travaux effectués sur l'estimation de l'utilisation des ressources mémoire a été faite sur les scalaires (Kjeldsberg, Catthoor et Aas, 2003). Un scalaire est l'équivalent d'un signal, d'une variable ou de chaque élément d'un tableau de l'application. Les estimations scalaires sont utilisées par les compilateurs actuels pour l'allocation des registres d'un programme (Florin, Hongwei et Ilie, 2007). Des algorithmes comme le « clique partitioning »(Chia-Jeng et Siewiorek, 1986) ou le « left-edge »(Kurdahi et Parker, 1987) sont des exemples de méthodes d'estimation scalaire. L'algorithme du «clique partitioning » consiste à créer un graphe où un nœud représente une variable ou un élément d'un tableau et un lien entre deux nœuds indique que ces variables sont combinables. Les variables sont dites combinables lorsque leurs temps de vie est disjoint ou que l'une est utilisé comme source de l'autre. Ensuite, lorsque le graphe est construit, l'algorithme regroupe les nœuds en cliques. Une clique est un groupe de nœuds où tous les nœuds sont interconnectés entre eux. L'algorithme de partitionnement essai de minimiser le nombre de cliques (Chia-Jeng et Siewiorek, 1986). Par contre, la complexité de l'algorithme est une fonction polynomiale du nombre de nœuds donc, les techniques scalaires sont limitées lors de la présence de tableaux multidimensionnels. En ce qui concerne l'algorithme « left-edge » sa complexité est $O(n^2)$ où n est le nombre d'éléments de données (Kurdahi et Parker, 1987). Le principe de fonctionnement du « left-edge » est semblable au « clique partitioning », il consiste à définir des axes de temps de vies à chaque variable et d'assigner de gauche à droite sur une même ligne plusieurs axes. Les axes ne doivent pas déborder les uns sur les autres et lorsqu'il n'y a plus de possibilités d'assignation sur une même ligne, on doit en créer une nouvelle ligne et recommencer les assignations. Selon (Kjeldsberg, Catthoor et Aas, 2003) la complexité de ces deux algorithmes augmente au carré en fonction du nombre d'éléments de données. Ces techniques impliques aussi que la connaissance du temps de production et consommations de chacun des éléments avant de procéder à une estimation (Balasa, Catthoor et Hugo De, 1995). Ainsi, ces méthodes ne sont pas optimales pour les

applications de traitement du signal puisque les opérations s'effectuent sur des tableaux multidimensionnels.

4.3.2 Méthodes non scalaires

Dans le but de combler les lacunes des techniques scalaires avec les tableaux multidimensionnels, des recherches ont été effectuées pour séparer les tableaux en unités appropriées avant ou pendant l'estimation (Kjeldsberg, Catthoor et Aas, 2003). En effet, pour estimer l'espace mémoire requis par des tableaux multidimensionnels avec une méthode scalaire chaque élément du tableau doit être extrait pour former un scalaire, augmentant l'effort de calcul nécessaire. Ces méthodes sont basées sur l'utilisation du CDFG, mais au contraire des méthodes scalaires les nœuds ne représentent plus une seule variable. En effet, avant de procéder à l'estimation des ressources, ces méthodes doivent réorganiser le CDFG en groupant les scalaires, variables et éléments de tableaux, dans des ensembles de données. Ainsi, les nœuds du graphe ne représentent plus un scalaire mais des ensemble de données et les liens correspondent aux relations de dépendances entre les groupes (Balasa, Catthoor et Hugo De, 1995). Les nœuds sont ensuite pondérés selon la taille de leur ensemble de données et le poids des liens équivaut au nombre exact de dépendances de données entre deux nœuds. Finalement, les méthodes parcourent le graphe en essayant de trouver un ordonnancement produisant un minimum d'ensemble de données actifs en même temps, ce qui permet de réduire la quantité de mémoire nécéssaire. Les techniques non-scalaires se divisent en deux catégories : celles qui requièrent un ordre d'exécution fixe et celles qui assument un ordre non-procédural où l'ordre d'exécution n'est pas fixé (Florin, Hongwei et Ilie, 2007). Dans les méthodes ordonnancées, il y a celle proposée par Verbauwhede *et al.* où un axe du temps de production et de consommation est construit pour chaque tableau. La différence entre les deux temps indique la quantité d'éléments produits entre les deux, le maximum de cette quantité indique l'espace de stockage maximal requis (Verbauwhede, Scheers et Rabaey, 1994). Grun *et al.* (Grun, Balasa et Dutt, 1998) proposent une méthode qui analyse la dépendance des données avant d'effectuer l'estimation de la quantité de

mémoire utilisée. L'algorithme analyse le code pour déterminer le nombre d'éléments produits et consommés à l'intérieur d'une boucle. Ainsi, la quantité de mémoire requise pour l'exécution d'une boucle est la somme du nombre d'éléments au début de la boucle et le nombre d'éléments produits dans la boucle moins le nombre consommé dans la boucle. Pour ce qui est des méthodes d'estimation, lorsque l'ordre d'exécution est partiellement fixé Kjeldsberg *et al.* proposent d'utiliser la même méthode d'analyse des dépendances des données que (Grun, Balasa et Dutt, 1998) mais d'utiliser l'information disponible sur l'ordre d'exécution pour générer une estimation plus précise. Les méthodes non-scalaires sont efficaces pour produire une estimation de la quantité de mémoire en regroupant les scalaires d'une application, diminuant ainsi la complexité de l'opération. Par contre, elles nécessitent un format de CDFG plus évolué, ce qui impose un ensemble d'outils supplémentaires.

4.4 Utilisation des méthodologies dans le projet OPERA

L'utilisation des tests de performances algorithmiques (TPA) dans une méthodologie d'estimation de performaces pose problème, car ils sont limités à un ensemble restreint de fonctions formant une librairie. Chaque fonction de la librairie doit avoir été caractérisée sur la plateforme au préalable. Ainsi, lors du développement d'une nouvelle application, les TPA ne peuvent être utilisés que dans le cas où l'application est constituée de blocs appartenant tous à la librairie. Lorsque de nouveaux algorithmes, non caractérisés, sont intégrés dans une application, on ne peut fournir une estimation des performances avec cette méthode. Or, pour une application de communication sans-fil, l'augmentation des débits de données supportés pousse les concepteurs à utiliser des techniques de traitement de signal plus sophistiquées. Par exemple, (Rysavy, 2006) explique que les récepteurs 3G à haut débit devraient avoir recours à des techniques MIMO dans les prochaines évolutions du standard UMTS. Donc, notre méthodologie d'estimation des ressources ne peut se baser seulement sur des TPA puisque celles-ci ne pourraient pas suivrent le développement des nouvelles technologies. Notre méthode doit être apte à estimer les performances d'une application à partir d'une représentation haut niveau sans avoir à implémenter l'application en entier sur la

plateforme. Enfin, les méthodes basées sur le CDFG, qu'elles soient scalaire ou non, nécessitent des outils de génération de graphe à partir d'une représentation haut niveau. Ces méthodes sont efficaces mais leur utilisation dans le projet OPERA est restreinte par l'absence d'outils de génération du CDFG à partir d'une représentation Matlab-Simulink. Un outil de génération est en cours de développement et son utilisation avec les méthodes proposé précédemment n'est pas encore envisagée. Les méthodes scalaires devraient être abordées en premier lieu puisqu'elles ne nécessitent pas de réorganisation du CDFG et leur algorithme d'estimation est plus simple. Par contre, afin d'obtenir de meilleurs performances dans des algorithmes plus complexes l'implémentation des méthodes non-scalaires devra être effectuée.

4.5 Conclusion

Dans ce chapitre, des métriques de performances ainsi que les principales méthodologies d'estimation de la mémoire pour des applications de traitement de signaux existantes ont été présentées. Il est souligné que ces méthodes utilisent une représentation haut niveau qui n'est pas compatible avec les objectifs du projet. Donc, aucune de ces méthodes n'a été retenue pour leur utilisation dans la présente recherche. Par contre, dans les phases ultérieures du projet OPERA, lorsqu'il sera nécessaire de déterminer le potentiel d'optimisation d'un algorithme, ces méthodes pourraient être utiles. Il faudrait cependant adapter la représentation Matlab/Simulink pour utiliser ces méthodes. Bien qu'un outil de génération soit en cours de développement, il est important de fournir une première estimation de l'utilisation de la mémoire basée directement sur la représentation Simulink afin de valider l'outil. Le processus de génération de graphe proposé à la Figure 4.2 requiert l'utilisation de plusieurs logiciels et format de représentation intermédiaires qui peuvent introduire des erreurs avec la représentation initiale. Ainsi, une méthode d'estimation analysant directement la représentation Matlab de l'application est proposée dans le prochain chapitre. Elle servira de référence pour la suite du projet.

CHAPITRE 5

MÉTHODOLOGIE PROPOSÉE

Il a été mentionné dans les chapitres précédents que l'estimation de l'utilisation des ressources mémoires d'une application est critique dans les premières phases de la conception. En effet, l'espace mémoire intégrée sur les processeurs est limitée et le coût des transferts vers la mémoire externe est élevé en temps et puissance. En ciblant les point critiques d'une application à l'aide d'estimations, les concepteurs peuvent concentrer leurs efforts d'optimisation afin d'obtenir de meilleurs performances. Par contre, les méthodes d'estimation existantes sont souvent complexes et ne s'intègrent pas dans les premières phases du processus de conception. Ainsi, pour des applications de communication sans fil, ce processus commence par la modélisation de l'application à haut niveau pour valider les spécifications. La méthodologie proposée s'intègre à l'outil de simulation Simulink du logiciel Matlab. Ce chapitre présente en premier lieu la méthodologie d'estimation basée sur Simulink, ensuite le logiciel d'analyse et finalement les résultats d'estimation.

5.1 Estimation des performances à partir d'un modèle Simulink

L'outil Simulink du logiciel Matlab permet de modéliser graphiquement une application de télécommunication dans son ensemble. Il fournit plusieurs blocs utilitaires comme des modèles de canaux et des outils d'analyses de performances du lien de communication. Il existe aussi des librairies contenant des blocs constituant les éléments fondamentaux des applications de télécommunication et de traitement de signaux. Ces librairies incluent des blocs comme des :

- modulateurs/démodulateurs QAM;
- encodeurs/décodeurs;
- entrelaceurs;

- filtres numériques;
- modèles de canaux.

Les blocs sont configurables, ce qui permet leur utilisation dans plusieurs applications. Les concepteurs peuvent ainsi modéliser et simuler rapidement une nouvelle application et ses spécifications. Par contre, lorsque vient l'étape de l'implémentation matérielle, Matlab ne procure pas d'indications de performances sur la plateforme ciblée. Dans les sections suivantes, des solutions pour obtenir des résultats de performances à partir d'un modèle Simulink sont examinées.

5.1.1 Analyse du fichier Simulink

La première solution envisagée fut l'analyse du contenu d'un fichier Simulink. Le fichier Simulink contient les informations relatives au modèle de l'application. Les informations sont accessibles en ouvrant le fichier en mode texte. Ainsi, le premier logiciel développé peut analyser le contenu des fichiers Simulink afin d'en extraire les données pertinentes sur l'application pour en estimer les performances. Cette solution a été approfondie mais il en ressortit que le fichier ne contenait pas toutes les informations nécessaires pour obtenir une estimation. Le fichier inclut les liens entre les blocs constituant le modèle ainsi que la valeur des paramètres de configuration. Il ne contient pas les algorithmes ou la structure interne de blocs complexes compris dans les librairies. Donc, il n'est pas possible d'obtenir une estimation de l'utilisation des ressources pour des modèles utilisant des blocs complexes. Par contre, cette solution est envisageable dans le cadre de l'analyse de la structure haut niveau de l'application et de la génération automatique de graphe de flux de données. Mais cela ne fait pas partie des objectifs du présent mémoire, alors cette solution n'a pas été investiguée davantage.

5.1.2 Real Time Workshop Embedded Coder

La deuxième solution examinée est le Real Time Workshop (RTW) Embedded Coder. C'est un ajout à l'outil Simulink qui permet de générer du code C optimisé pour des systèmes embarqués à partir d'un modèle Simulink. Cette solution nécessite un ensemble de configurations pour la plateforme ciblée pour guider le générateur de code. Cet ensemble de configurations contient des noyaux de code C pour chacun des blocs constituant le modèle Simulink. Or, il s'avère que ce ne sont pas tous les blocs disponibles qui sont supportés. De plus il s'agit d'un ensemble restreint pour certains processeurs. Par exemple, il existe déjà des modules pour certains DSP de la compagnie Texas Instruments Inc.(TI), mais ces modules sont optimisés pour les plateformes de TI. Il est possible de créer un ensemble de configurations sur mesure pour d'autres plateformes, sauf qu'il faut alors écrire le code des blocs utilisés. Le développement d'un tel ensemble est long et exige une connaissance approfondie de la plateforme. Cette solution n'est donc pas idéale alors que l'on cherche à obtenir des résultats rapidement et avec des connaissances minimales du matériel. De plus, pour obtenir des résultats de performances, le code doit être profilé sur un simulateur d'instructions ou sur une version matérielle de la plateforme. L'objectif de la présente méthodologie étant de fournir une estimation de la performance d'une application alors que la plateforme est en cours de développement, cette solution ne peut être retenue.

5.1.3 Analyse des scripts Matlab Embedded

La dernière solution considérée est l'analyse des scripts Matlab Embedded. Celle-ci a été retenue, car il s'est avéré premièrement que l'analyse des fichiers Simulink ne permet pas de voir la structure interne des blocs complexes. Ensuite, l'utilisation de générateur de code n'était pas viable pour une plateforme en cours de développement. Ainsi, les deux premières solutions envisagées n'ont pu fournir les données nécessaires à une méthode d'estimation de l'utilisation des ressources mémoires. Finalement, après l'étude de modèles Simulink d'applications existantes, avec le logiciel d'analyse du fichier Simulink dont il est question

dans la section 5.1.1, il en est ressorti que ceux-ci étaient constitués de plusieurs blocs invoquant des scripts écrit en langage Matlab ou à des fonctions définies en code C ou C++. Par exemple, pour un modèle représentant un système OFDMA, la classification des types de blocs qui constituent le modèle est représentée dans le **Tableau 5.1**. Les blocs de type « Reference » appellent des fonctions définies en code C ou C++ et les blocs « MatlabFnc » appellent des scripts en langage Matlab. Donc, dans ce modèle il y a 127 blocs du type « Reference » et 17 qui appellent une fonction en langage Matlab alors que seulement 98 sont des blocs primaires. Les blocs primaires sont des opérations fondamentales telles que des multiplicateurs, et autre fonctions mathématiques simples. On remarque que 39% des blocs invoquent une fonction externe au modèle Simulink. Ainsi, pour obtenir une estimation des performances fiable il est nécessaire d'obtenir les informations sur ces blocs.

Tableau 5.1 Occurrences des différents types de blocs dans un modèle Simulink d'une application OFDMA

Occurrences des différents types de blocs dans un modèle		
Type de Blocs	Nombres	Proportion en %
Reference	127	35
Inport	65	18
Outport	60	16
MatlabFnc	17	4
Blocs primaires	98	27
Total	367	100

5.1.3.1 Utilisation des scripts Matlab dans un modèle Simulink

Lorsque le concepteur veut valider un algorithme qu'il désire implémenter ou qu'une fonction faisant partie d'une application n'a pas d'équivalent en bloc Simulink, l'emploi de scripts Matlab est une bonne solution. En effet, le langage Matlab comprend de nombreuses caractéristiques et opérateurs mathématiques qui simplifient l'écriture d'algorithmes de traitement de signaux. Le langage est particulièrement bien adapté à la manipulation de

matrices, vecteurs et tableaux multi-dimensionnels. Ainsi, lors du développement du décodeur de Viterbi pour le projet OPERA, le concepteur a d'abord validé son algorithme en utilisant un bloc « MatlabFnc » dans un modèle Simulink. L'algorithme du décodeur écrit en script est le même que celui implémenté sur la plateforme. Par contre, effectuer une estimation des performances d'une application à partir d'un script Matlab pose problème en particulier dans le cas des ressources mémoires. Le langage Matlab fait abstraction des types de données et les tailles des tableaux sont souvent implicitement définis. Ainsi, il est possible d'ajouter des éléments à la fin d'un tableau sans avoir défini au préalable sa taille. De plus, lorsqu'il s'agit d'opérations sur des nombres complexes, Matlab effectue les bonnes opérations de manière transparente au programmeur. Ce style de codage n'est pas supporté par les principaux langages de programmation des DSP et systèmes embarqués. Donc, l'utilisation de script Matlab n'est pas supportée par la méthodologie d'estimation proposée.

5.1.3.2 Embedded Matlab Function

Les Embedded Matlab Function (EMF) sont des blocs disponibles dans Simulink et qui utilisent un sous-ensemble du langage Matlab. Cela rend les modèles Simulink compatibles avec RTW pour la génération de codes sur des systèmes embarqués, contrairement aux MatlabFnc qui ne sont pas supportés par ce même outil. Le langage utilisé dans les EMF a la particularité d'être plus près d'un code implémenté sur un DSP. Dans les faits, les EMF limitent le programmeur à un style de codage qui exclut les déclarations de tableaux ou de variables implicites. Ainsi, la taille et le type d'une variable doivent être définis en avance et ne peuvent être modifiés par la suite à l'instar du langage C. Ce qui semble être une limitation est un avantage pour une méthode d'estimation des ressources mémoires puisque les informations supplémentaires servent aux calculs. Pour un concepteur, l'exercice de rendre compatible un script Matlab avec le format EMF est semblable à celui de l'implémentation.

5.1.3.3 Méthodologie d'estimation des EMF proposée

La méthodologie d'estimation proposée fait l'analyse des scripts EMF qui composent un modèle d'applications. L'analyse d'un script, où la taille des tableaux et leurs types sont définis clairement dans le code, permet de calculer l'espace mémoire total requis par la fonction. La déclaration des entrées et sorties de la fonction au début d'un script est utilisée pour déterminer les besoins en bande passante de la fonction. Cette estimation fournit une limite supérieure de la quantité de mémoire utilisée. En faisant le total de tous les tampons et variables déclarés dans le script, la méthode ne tient pas compte des possibilités d'optimisation telle que la réutilisation de tampons. La méthode analyse chacun des blocs constituant une application séparément. Le concepteur peut ensuite utiliser les informations fournies par la méthodologie afin de déterminer les parties d'une application qui consomment le plus de ressources mémoires et ainsi concentrer ses efforts d'optimisation.

5.1.3.4 Hypothèses de base

La méthodologie proposée se base sur l'hypothèse qu'une application complexe de traitement numérique du signal peut être décomposée en blocs ou primitives qui sont couramment utilisés dans ce domaine. On suppose que les hypothèses suivantes sont respectées :

- les primitives sont les composantes de l'application qui sont les plus exigeantes en termes de ressources mémoires;
- chaque primitive est exécutée séquentiellement sur un même noyau, ainsi une primitive en cours d'exécution a accès aux ressources du noyau exclusivement;
- au niveau de la puce entière, les primitives sont exécutées parallèlement sur les 15 noyaux;

- lorsque le traitement effectué par une primitive est terminé et qu'une nouvelle primitive commence à traiter des données, les registres qui étaient utilisés sont libérés;
- la validité de l'estimation dépend de la capacité du concepteur à choisir les types et les tailles optimaux des variables en fonction de la précision désirée;
- les types et les tailles des données sont connus du concepteur lors de l'écriture du script EMF;
- l'algorithme et les variables d'un script sont très semblables à son implémentation.

5.2 Logiciel d'analyse

La méthodologie a été automatisée dans un logiciel d'analyse des scripts EMF. Ce logiciel est codé en langage PERL. Ce langage permet de parcourir un fichier texte et d'en extraire des informations à partir de certains mots-clés. Les informations extraites par le logiciel sont les déclarations de tableaux et de variables. Elles servent ensuite à calculer et générer un rapport sur l'utilisation des ressources mémoires automatiquement. Pour estimer les ressources d'une application décrite en MATLAB, il faut suivre les étapes présentées dans la Figure 5.1.

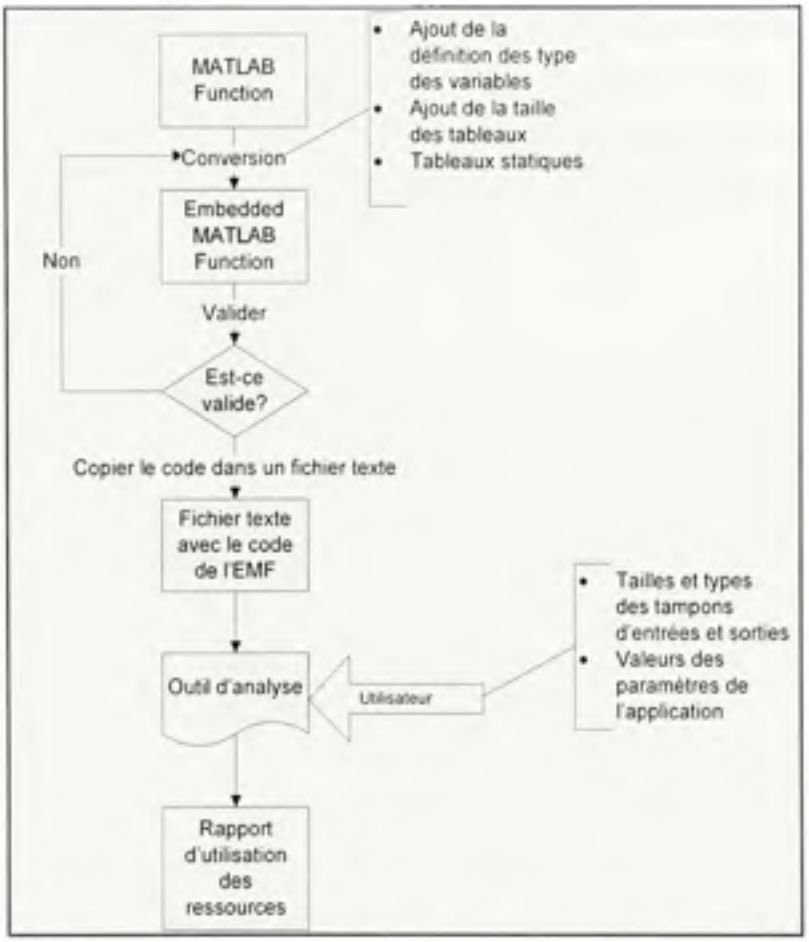

Figure 5.1 Processus d'utilisation du logiciel d'analyse.

Pour obtenir une estimation des performances d'une fonction EMF, l'utilisateur doit fournir au logiciel un fichier texte contenant le code de la fonction avec les annotations et les déclarations de types mentionnés précédemment. Afin de produire des résultats valables l'utilisateur doit spécifier les formats des tampons d'entrées et de sorties, car ceux-ci ne sont pas spécifiés par l'interface de la fonction EMF. Le logiciel supporte les types de données entières, les tableaux multi dimensions statiques et les nombres complexes. Les nombres exprimés en notation point flottant ne sont pas acceptés car la plateforme Vocallo ne les prend pas en charge. L'allocation dynamique de mémoire n'est pas supportée, car le format EMF ne l'admet pas et les outils de développement de la plateforme non plus. Après

l'analyse d'un script, le logiciel produit un rapport de l'utilisation des ressources par les différents tampons et variables de l'application dans un fichier lisible par un tableur électronique. Présentement, le logiciel existe dans une version autonome, mais les prochaines versions devraient s'intégrer à un outil d'analyse de l'application globale.

5.2.1 Vérification de l'outil

Pour vérifier l'outil d'analyse, l'implémentation d'un filtre FIR réel par bloc a été réalisée. La forme du FIR est celle présentée dans la Figure 2.2. Ainsi, pour vérifier la fonctionnalité de l'outil, le résultat de l'équation (2.2) a servi de référence. Par exemple, pour un filtre avec les spécifications du Tableau 5.2 l'utilisation des ressources devrait correspondre aux résultats qui suivent.

Tableau 5.2 Spécifications du filtre FIR réel réalisé

Spécifications	Valeurs
N_s	16
N_q	16
N	8

Ainsi la taille du tampon d'entrée devrait être de :

$$\Pi_e = N_s \times N_q$$
$$\Pi_e = 16 \times 16 = 256\ bits \Rightarrow 32\ bytes \tag{5.1}$$

La taille du tampon de sortie :

$$\begin{aligned} &\Pi_s = 2 \times N_s \times N_q \\ &\Pi_s = 2 \times \Pi_e \\ &\Pi_s = 2 \times 256 = 512\ bits \Rightarrow 64\ bytes \end{aligned} \tag{5.2}$$

La taille des autres tampons du FIR:

$$\begin{aligned} &\Pi_d = N \times N_q \\ &\Pi_d = 8 \times 16 = 128\ bits \Rightarrow 16\ bytes \end{aligned} \tag{5.3}$$

$$\begin{aligned} &\Pi_h = N \times N_q \\ &\Pi_h = \Pi_d = 128\ bits \Rightarrow 16\ bytes \end{aligned} \tag{5.4}$$

Ce qui nous donne un espace total théorique de :

$$\begin{aligned} &\Pi_t = \Pi_e + \Pi_s + \Pi_d + \Pi_h \\ &\Pi_t = 32 + 64 + 16 + 16 = 128\ bytes \end{aligned} \tag{5.5}$$

Alors si on compare avec l'estimation théorique de l'équation (5.5) avec l'estimation de l'outil qui nous génère le rapport du Tableau 5.3, on peut remarquer que l'espace total est le même. L'outil indique aussi l'espace nécessaire aux variables temporaires utilisées par l'algorithme. Ces variables peuvent être stockées dans des registres ou bien en mémoire locale. Dans le premier cas, cette information peut être utilisée puisque le nombre de registres dans un processeur est très limité, 64 registres dans le cas du Vocallo. Le logiciel indique alors la limite supérieure de registres utilisés par l'application, puisque le compilateur peut réutiliser un même registre lorsqu'une variable est transférée en mémoire. Le temps de transfert indiqué dans le tableau pour les tampons d'entrée et de sortie est

calculé à partir de la vitesse de transfert des données du bus mémoire. Ainsi, la vérification théorique indique que l'outil est fonctionnel et les estimations sont valables.

Tableau 5.3 Rapport d'estimation d'utilisation de la mémoire pour un FIR réel à 8 coefficients et 8 échantillons d'entrée

Variables		
Name	Type	Size(bytes)
Ntaps	int16	2
j	int16	2
i	int16	2
Total		6
All Buffers		
Name	Type	Size(bytes)
x_in	int16	32
y_out	int32	64
coef	int16	16
x_reg	int16	16
Total		128
Outputs Buffers		
Name	Type	Size(bytes)
y_out	int32	64
Total		64
Transfert time (s)	6.014976e-008	
Inputs Buffers		
Name	Type	Size(bytes)
x_in	int16	32
Total		32
Transfert time (s)	3.007488e-008	

5.3 Résultats d'estimation

Cette section présente les résultats obtenus lors de la validation de la méthodologie d'estimation. Dans un premier temps, la méthodologie de collecte des résultats pour différentes applications est présentée. Ensuite, les résultats de ces tests d'estimation sont comparés avec des analyses théoriques et leur implémentation sur la plateforme. Finalement, une discussion sur les résultats suivra.

5.3.1 Méthodologie de compilation des résultats

Afin de valider correctement la méthode d'estimation, celle-ci a été utilisée avec les applications décrites dans le chapitre 2. Les applications suivantes ont été sélectionnées pour valider la méthode d'estimation et elles proviennent du domaine des communications sans-fil :

- Filtre FIR réel par bloc;
- Filtre FIR complexe;
- Filtre égaliseur adaptatif LMS complexe;
- Décodeur de Viterbi;
- Maximum vectoriel;
- Addition de vecteurs.

Elles ont d'abord été réalisées en langage MATLAB afin de valider l'algorithme puis transformée en Embedded Matlab Function (EMF) qui tient lieu de modèle de référence. À partir de ce dernier, on peut soit obtenir une estimation avec l'outil d'analyse ou implanter l'algorithme sur la plateforme matérielle, voir la Figure 5.2. Des vecteurs de données générés avec le modèle de référence sont aussi utilisés lors de la vérification de l'implémentation matérielle. Il est essentiel d'avoir une implémentation qui est identique ou très semblable au modèle de référence afin d'obtenir des résultats qui concordent avec

l'estimation. Les tailles et le nombre de tampons ainsi que les types des données sont les paramètres qui influencent le plus l'erreur entre l'estimation et l'implémentation. L'outil d'analyse ne tient pas compte de l'algorithme des applications, donc les différences entre l'EMF et l'implémentation n'affectent pas les résultats. Les applications sélectionnées, sauf le décodeur de Viterbi, ont des paramètres qui font varier la taille de certains tampons. En variant ces paramètres, il a été possible de déterminer si l'erreur entre l'estimation et l'implémentation était statique ou variable. On s'attend à obtenir une certaine erreur statique entre l'estimation et l'implémentation car l'environnement de développement intégré (IDE) d'Octasic ajoute la quantité de mémoire utilisée par des fichiers d'entêtes à son rapport d'utilisation de la mémoire.

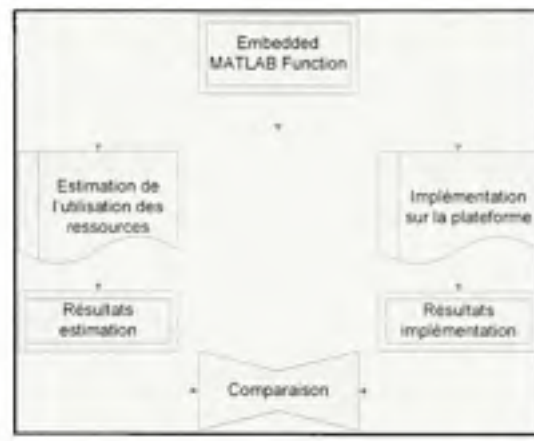

Figure 5.2 L'EMF comme modèle de référence.

Les résultats obtenus sont ensuite comparés sous forme graphique afin de déceler des erreurs dynamiques. Lorsqu'il s'avère que l'amplitude de l'erreur change en fonction des paramètres, une comparaison entre l'EMF et de l'implémentation est effectuée afin de déterminer si l'erreur est due à la méthode ou à une différence avec le modèle de référence.

5.3.1.1 Version de l'outil de développement d'Octasic

Les résultats de la partie implémentation proviennent du rapport des ressources généré par la première version de l'IDE d'Octasic pour toutes les applications sauf dans le cas du FIR réel, où la nouvelle version a aussi été utilisée.

5.3.2 Présentation des résultats

Cette section présente les résultats d'estimation et d'implémentation obtenus avec les différentes applications sélectionnées. Ces résultats ont été collectés selon la méthodologie décrite dans la section précédente.

5.3.2.1 Filtre FIR réel par bloc

Les résultats pour le FIR réel proviennent de trois sources : l'outil d'analyse, la première version et la nouvelle version de l'IDE. La raison pour laquelle le FIR a été implémenté sur les deux versions de l'IDE est de démontrer que les résultats d'estimation sont valables indépendamment de la version. Les deux paramètres variables du filtre sont le nombre de coefficients et la taille de la trame de données à traiter. Les résultats ont été collectés avec un paramètre fixé et l'autre variable. Le nombre de bits de quantification pour les échantillons est de 16 bits. On obtient ainsi deux séries de données indépendantes qui sont présentées dans le Tableau 5.4.

Tableau 5.4 Résultats compilés pour le filtre FIR réel par bloc

Filtre FIR réel						
Nombre de coefficients	Nombre d'échantillons	Mémoire Estimation	Mémoire implémentation ancien IDE	Erreur	Mémoire implémentation nouvel IDE	Taille du programme
		(bytes)	(bytes)	(bytes)	(bytes)	(bytes)
4	16	112	128	16	112	516
4	32	208	224	16	208	516
4	64	400	416	16	400	520
4	128	784	800	16	784	520
4	256	1552	1568	16	1552	520
8	16	128	144	16	128	516
16	16	160	176	16	160	516
32	16	224	240	16	224	516
64	16	352	368	16	352	524
128	16	608	624	16	608	524

On remarque que l'erreur entre l'estimation et l'implémentation sur l'ancien IDE est fixe peu importe la variation des paramètres de configuration du filtre. Cette erreur est causée par l'ajout de constantes et de déclarations supplémentaires de l'IDE qui ne font pas partie de la fonction. Avec la nouvelle version, il n'y a pas d'erreur car il est possible de déterminer la mémoire utilisée par la fonction seulement.

Ainsi, représentées graphiquement les deux séries de résultats nous donnent les Figure 5.3 et Figure 5.4. L'erreur entre l'estimation demeure statique peu importe la variation des paramètres. Les résultats démontrent aussi que la quantité de mémoire utilisée par un filtre FIR dépend de la taille de ses tampons. De plus, l'estimation s'est avérée exacte dans tous les cas avec la nouvelle version de l'IDE. Le cas du filtre FIR réel démontre aussi la validité de la méthode avec les deux versions de l'IDE d'Octasic.

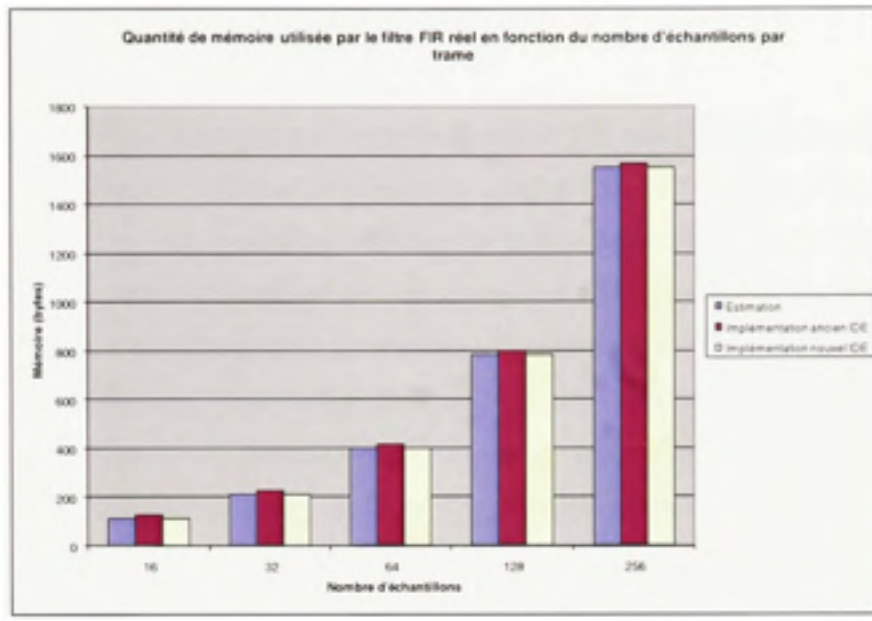

Figure 5.3 Résultats pour le filtre FIR réel en fonction du nombre d'échantillons par trame.

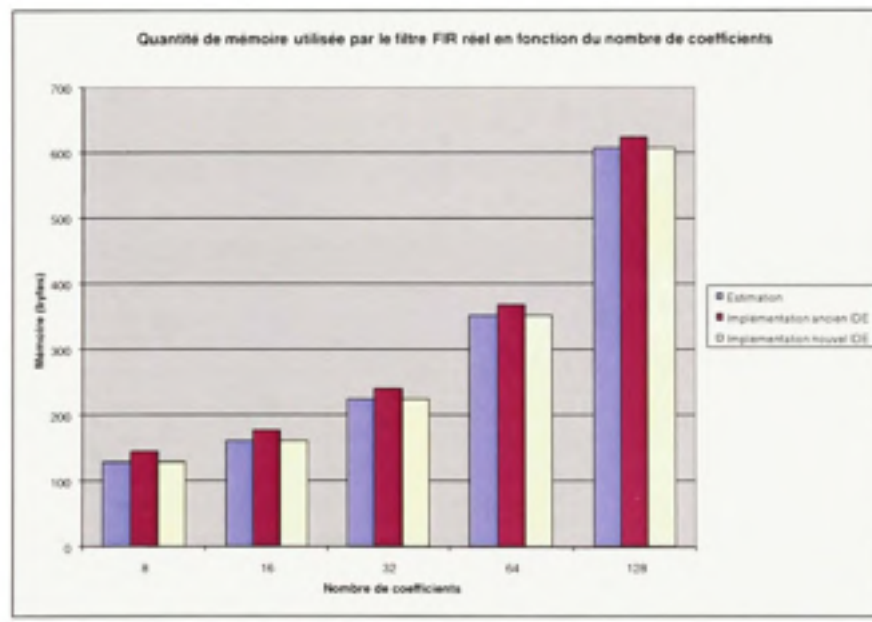

Figure 5.4 Résultats pour le filtre FIR réel en fonction du nombre de coefficients.

5.3.2.2 Filtre FIR complexe

La méthode utilisée pour produire les résultats pour le FIR complexe est la même que celle du FIR réel. La taille des trames et le nombre de coefficients sont encore les deux paramètres variables puisque le FIR complexe effectue la même opération que le filtre réel. La différence notable entre les résultats du FIR complexe (Tableau 5.5) et le filtre réel est la quantité de mémoire nécessaire pour les mêmes paramètres. En effet, les données exprimées en notation complexe ont deux composantes ce qui nécessite deux espaces mémoires par donnée. Dans la Figure 5.5, lorsque l'on fait varier la taille de la trame reçue, l'erreur entre l'estimation et l'implémentation est statique. Or, on remarque dans la Figure 5.6 que l'erreur augmente avec le nombre de coefficients. Cette erreur dynamique est due à une différence entre le script EMF et l'implémentation. Un tampon supplémentaire a été utilisé dans l'implémentation pour effectuer le complexe conjugué des coefficients du filtre. La taille du tampon dépend seulement du nombre de coefficients c'est donc pourquoi celle-ci n'apparait pas dans la Figure 5.5. Mise à part cette différence, l'estimation de la quantité de mémoires se situe très près de l'implémentation.

Tableau 5.5 Résultats compilés pour le filtre FIR complexe

Filtre FIR complexe					
Nombre de coefficients	Nombre d'échantillons	Mémoire estimation	Mémoire implémentation	Erreur	Taille du programme
		(bytes)	(bytes)	(bytes)	(bytes)
4	16	224	284	60	1680
4	32	416	476	60	1684
4	64	800	860	60	1684
4	128	1568	1628	60	1684
4	256	3104	3164	60	1684
8	16	256	332	76	1684
16	16	320	428	108	1684
32	16	448	620	172	1692
64	16	704	1004	300	1700
128	16	1216	1772	556	1700

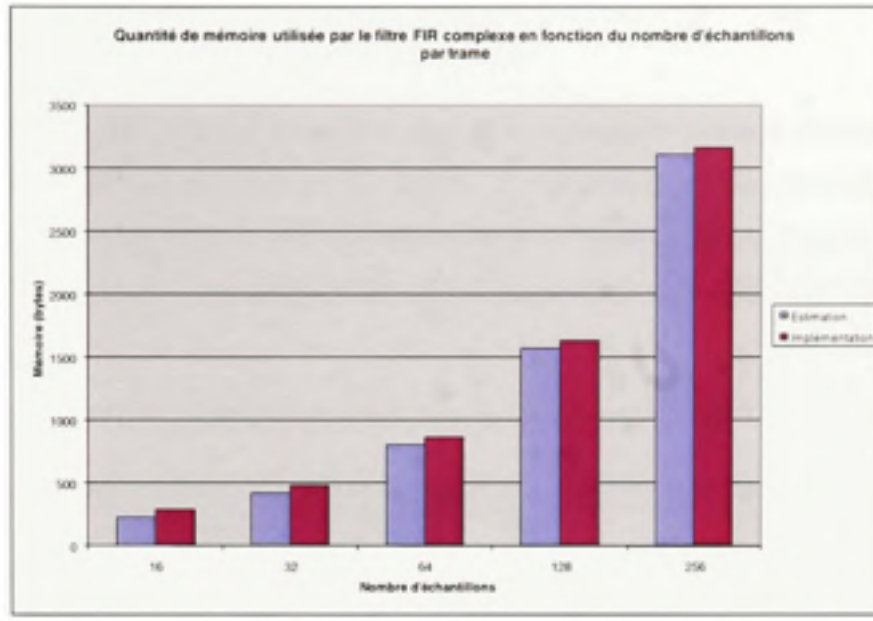

Figure 5.5 Résultats pour le filtre FIR complexe en fonction du nombre d'échantillons par trame.

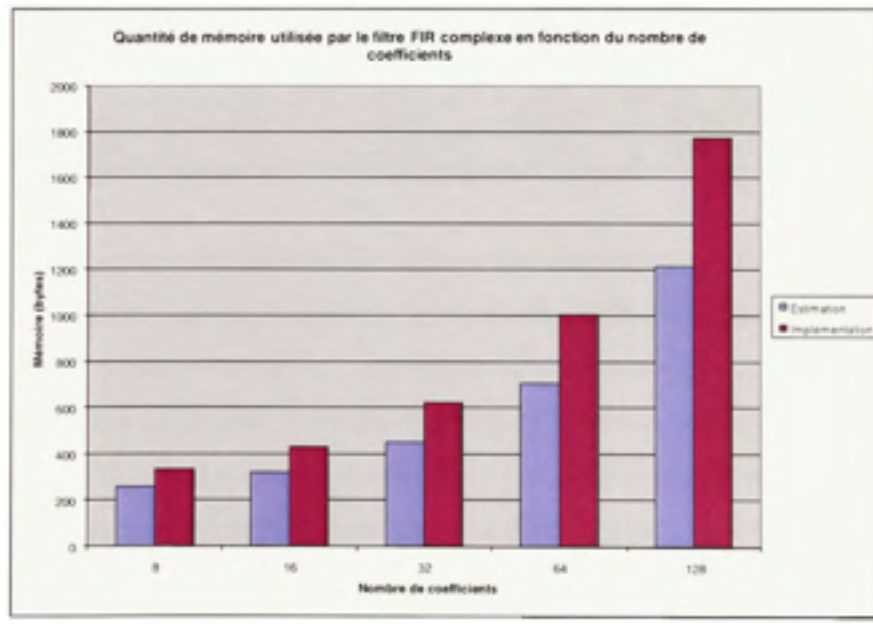

Figure 5.6 Résultats pour le filtre FIR complexe en fonction du nombre de coefficients.

5.3.2.3 Filtre égaliseur adaptif LMS complexe

Le cas de l'égaliseur LMS est semblable aux deux précédents puisque celui-ci effectue la même opération principale mais en y ajoutant une fonction de correction des poids des coefficients. Comme dans le cas du FIR complexe, lorsque le nombre de coefficients augmente, l'erreur entre l'estimation et l'implémentation augmente (Tableau 5.6).

Tableau 5.6 Résultats compilés pour le filtre égaliseur LMS complexe

Filtre LMS EQ complexe					
Nombre de coefficients	Nombre d'échantillons	Mémoire estimation	Mémoire implémentation	Erreur	Taille du programme
		(bytes)	(bytes)	(bytes)	(bytes)
4	16	446	416	-30	3884
4	32	830	672	-158	3884
4	64	1598	1184	-414	3884
4	128	3184	2208	-976	3884
4	256	6206	4256	-1950	3884
8	16	478	464	-14	3884
16	16	542	560	18	3888
32	16	670	752	82	3896
64	16	926	1136	210	3904
128	16	1438	1904	466	3904

Cela est normal puisque la fonction de filtrage du LMS est une copie conforme du FIR complexe. Par contre, lorsque la taille de la trame augmente, une erreur croissante apparait (Figure 5.7). L'erreur provient de l'algorithme d'adaptation des coefficients qui, dans le script EMF, contient un tampon supplémentaire qui est nécessaire pour faire fonctionner la fonction dans MATLAB. Le langage Embedded MATLAB ne supporte pas les changements de la taille des données directement dans une opération lorsque l'on utilise les object « fixed point ». Alors qu'en langage C, cela s'effectue en forçant le type directement dans l'opération. Donc, la taille de ce tampon dépendant du nombre d'échantillons dans la trame l'erreur augmente en fonction de celui-ci.

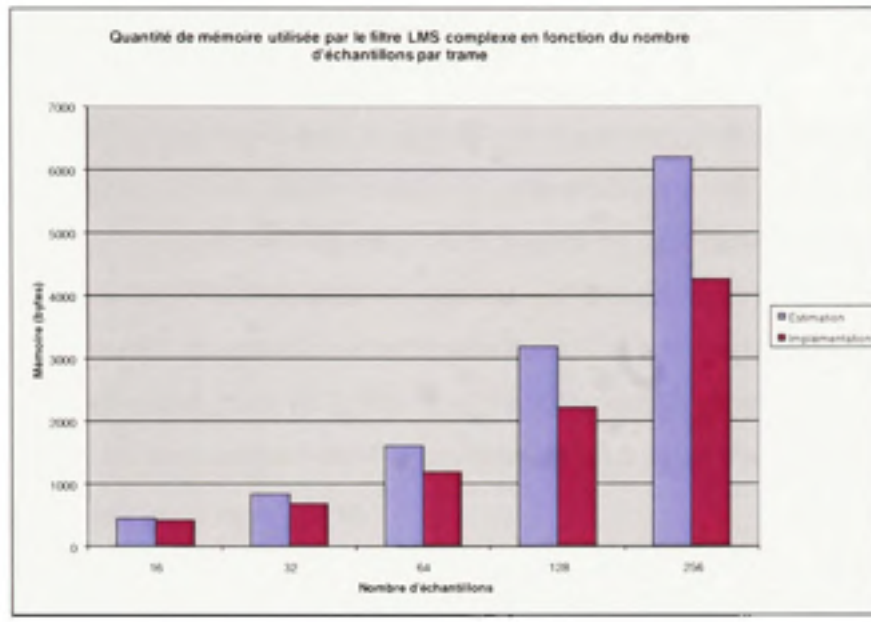

Figure 5.7 Résultats pour le filtre égaliseur LMS complexe en fonction du nombre d'échantillons par trame.

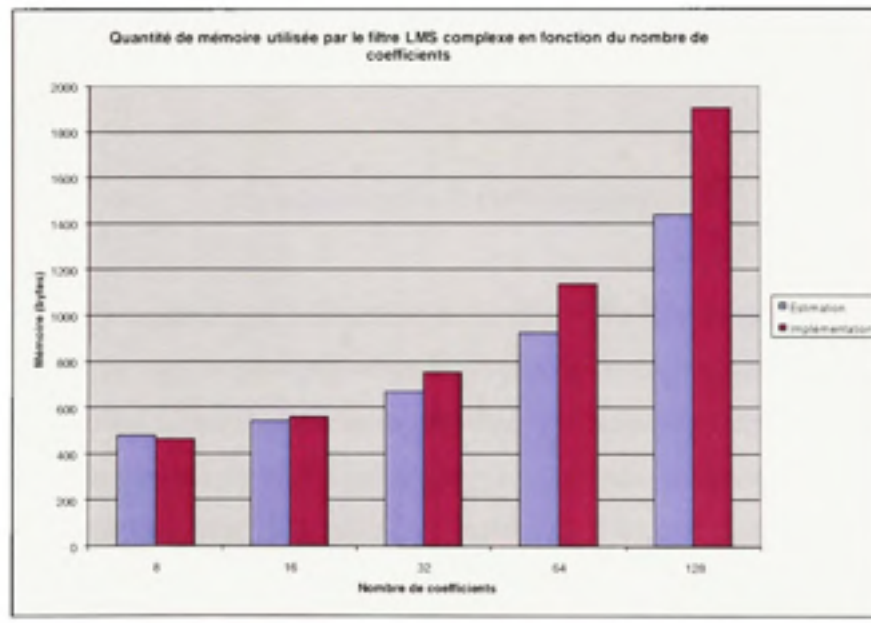

Figure 5.8 Résultats pour le filtre égaliseur LMS complexe en fonction du nombre de coefficients.

5.3.2.4 Décodeur de Viterbi

L'analyse du décodeur de Viterbi dans le cadre de ce projet s'est avérée intéressante puisque ce dernier a été implémenté en tenant compte des optimisations possibles avec la plateforme Vocallo. Comme le démontre le Tableau 5.7 et la Figure 5.9, la différence entre l'estimation et l'implémentation de 115924 octets est énorme, en comparaison avec les résultats du Tableau 5.8 où aucune optimisation n'a été effectuée. Cette surestimation est causée par l'utilisation du type d'une taille de 1 bit dans l'implémentation pour certains tampons du décodeur. Ainsi, tel que présenté dans la section 2.1.5.4.1 la matrice de retraçage a une dimension Π_R donnée par l'équation(5.6).

$$\Pi_R = n_{états} \times n_{sym} \tag{5.6}$$

Où $n_{états}$ est le nombre d'états du décodeur et n_{sym} est le nombre de symboles à décoder.

Dans notre cas le décodeur possède 256 états et 512 symboles à décoder, ce qui nous donne une dimension :

$$\Pi_R = 256 \times 512 = 131072\ (\textit{éléments}) \tag{5.7}$$

Or, cette matrice ne contient que la direction de branchement dans le treillis qui est exprimée par un 1 ou un 0. Le type le plus petit disponible dans le langage EMF a une taille de 8 bits. Ensuite, le tampon qui contient la sortie du décodeur peut aussi être exprimé sur un seul bit et sa taille dépend du nombre de symboles par trame. Ces deux optimisations assez simples à implémenter permettent ainsi d'économiser beaucoup d'espace mémoire. Ainsi, il est possible de calculer la différence théorique entre les deux résultats (équation(5.8)).

$$\Delta_{théorique} = \Pi_{est} - \Pi_{imp} \tag{5.8}$$

La mémoire nécessaire pour la matrice de retraçage dans l'estimation est de 131072 octets et la taille du tampon de sortie est de 512 octets (équation(5.9)). Lors de l'implémentation, la mémoire utilisée par ces deux tampons est huit fois moindre puisque les données sont exprimées sur un bit seulement (équation(5.10)).

$$\Pi_{est} = 131072 + 512 = 131584\ (bytes) \tag{5.9}$$

$$\Pi_{imp} = \frac{\Pi_{est}}{8} = \frac{131584}{8} = 16448\ (bytes) \tag{5.10}$$

La différence théorique est alors de :

$$\boxed{\Delta_{théorique} = 131584 - 16448 = 115136\ (bytes)} \tag{5.11}$$

L'erreur résiduelle est le surplus ajouté par l'outil de développement.

Tableau 5.7 Résultats compilés pour le décodeur de Viterbi

Décodeur de Viterbi					
Taux de codage R	Nombre d'échantillons	Mémoire estimation (bytes)	Mémoire implémentation optimisée (bytes)	Erreur (bytes)	Taille du programme (bytes)
1/3	504	134336	18412	-115924	2480

Tableau 5.8 Résultats compilés pour décodeur de Viterbi version non optimisée

Décodeur de Viterbi					
Taux de codage R	Nombre d'échantillons	Mémoire estimation (bytes)	Mémoire implémentation 8 bits (bytes)	Erreur (bytes)	Taille du programme (bytes)
1/3	504	134336	132972	-1364	2480

Figure 5.9 Résultats pour décodeur de Viterbi en fonction du nombre d'échantillons par trame.

Le potentiel d'optimisation du décodeur de Viterbi est bien représenté dans la Figure 5.9 ainsi, on remarque que la quantité de mémoire nécessaire dans la version optimisée est 7 fois moindre. Par contre, ces optimisations sont possibles en raison de l'algorithme de l'application qui nécessite seulement un bit pour la matrice de retraçage et la sortie décodée. Cette optimisation n'exploite pas les possibilités de réutilisation d'emplacement mémoire en tenant compte de la structure de l'application. Ainsi, une technique de fenêtrage pourrait diminuer la quantité mémoire nécessaire pour les métriques accumulées au dépend tu temps de traitement.

5.3.2.5 Fonction de maximum vectoriel et addition de vecteurs

Les fonctions de maximum vectoriel et addition de vecteurs sont assez simples et les résultats obtenus dans les deux cas n'ont fait ressortir aucune particularité. Le seul paramètre variable de ces fonctions est la taille de trame à traiter. L'erreur entre l'estimation et

l'implémentation est statique et est égale pour les deux fonctions, voir le Tableau 5.9 et le Tableau 5.10. L'erreur est causée par le surplus ajouté par l'ancienne version de l'IDE. On remarque que la quantité de mémoires augmente selon le nombre d'échantillons par trame. Ce qui est logique puisque la taille des tampons ne dépend que de ceci.

Tableau 5.9 Résultats compilés pour la fonction de maximum vectoriel

Vector Max				
Nombre d'échantillons	Mémoire estimation	Mémoire implémentation	Erreur	Taille du programme
	(bytes)	(bytes)	(bytes)	(bytes)
16	36	44	8	264
32	68	76	8	264
64	132	140	8	264
128	260	268	8	264
256	516	524	8	264

Tableau 5.10 Résultats compilés pour la fonction d'addition de vecteurs

Vector Add				
Nombre d'échantillons	Mémoire estimation	Mémoire implémentation	Erreur	Taille du programme
	(bytes)	(bytes)	(bytes)	(bytes)
16	96	104	8	236
32	192	200	8	232
64	384	392	8	236
128	768	776	8	236
256	1536	1544	8	236

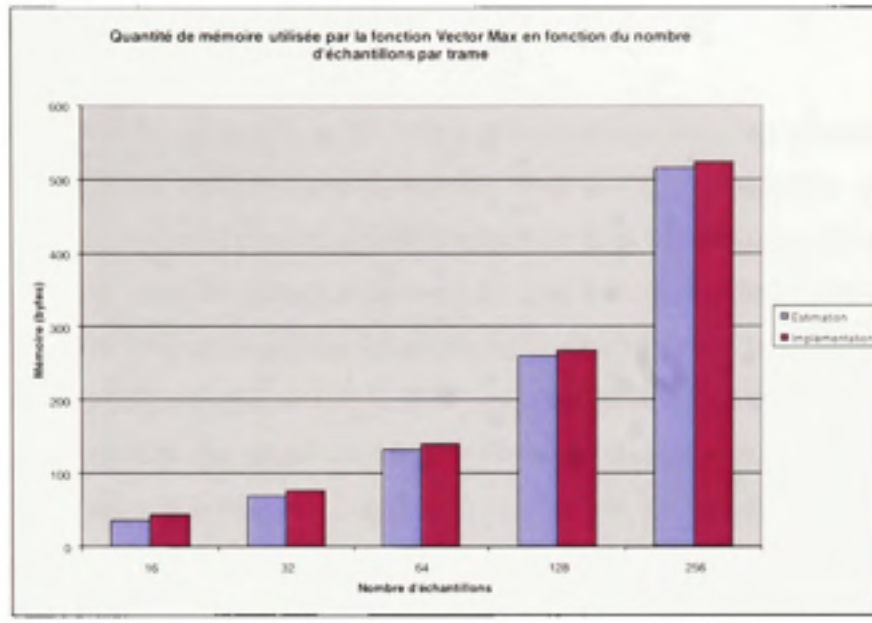

Figure 5.10 Résultats pour la fonction de maximum vectoriel en fonction du nombre d'échantillons par trame.

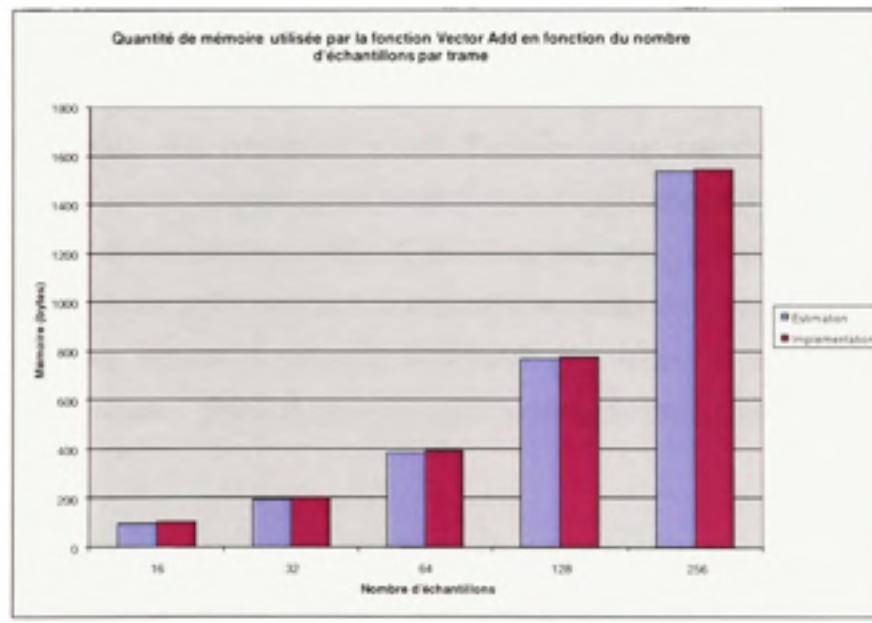

Figure 5.11 Résultats pour la fonction d'addition vectorielle en fonction du nombre d'échantillons par trame.

5.3.3 Discussion sur les résultats

Un examen détaillé de la variation de l'erreur d'estimation dans les résultats expose la sensibilité de l'outil aux différences entre le script EMF et l'implémentation. Ainsi, dans le cas du FIR complexe l'ajout d'un tampon intermédiaire dans l'implémentation fait varier de manière importante l'erreur lorsque le nombre de coefficients du filtre varie. Par contre l'erreur est statique lorsque le nombre de coefficients est fixe et que la taille de la trame de varie. Le même problème ressort avec le filtre égaliseur LMS. Cela démontre l'importance d'avoir un script EMF et une implémentation la plus semblable possible afin de ne pas sous-estimer les ressources nécessaires. Dans un autre sens, la différence entre l'estimation et l'implémentation peux nous indiquer le dégrée d'optimisation atteint en considérant le script EMF comme un modèle de référence. Ainsi, en examinant les résultats du décodeur de Viterbi on peut remarquer que l'implémentation matérielle est très optimale en termes d'utilisation de mémoire.

5.3.3.1 Exemple de partitionnement d'une application sur Vocallo

Lorsque l'estimation des ressources a été effectuée pour chacune des primitives, le concepteur peut passer à l'étape du partitionnement de l'application sur les différents noyaux. En effet, puisque le Vocallo est constitué de 15 noyaux avec chacun 96 Ko de mémoire locale, l'application pourrait devoir être partitionnée sur plusieurs noyaux. Ainsi, la taille de certaines primitives, comme le décodeur de Viterbi, peut utiliser la totalité de la mémoire locale d'un seul noyau. Prenons, par exemple, une application composée des primitives suivantes :

- FIR complexe à 4 coefficients et sur 256 échantillons;
- FIR réel par bloc à 128 coefficients et 16 échantillons;
- Décodeur de Viterbi à un taux 1/3, version optimisée;
- Filtre LMS complexe à 4 coefficients et 256 échantillons.

Toutes ces primitives ont été définies précédemment et la quantité de mémoire nécessaire à chacune a été estimée avec l'outil d'analyse. Le partitionnement proposé sur le Vocallo est présenté dans la Figure 5.12. Les deux primitives qui nécessitent le moins de ressources ont été assignées au noyau 1, alors que les deux autres qui sont plus exigeantes sont assignées à un noyau chacune. Il est à noter qu'au total la quantité de mémoire requise par l'ensemble des primitives est inférieure aux ressources disponibles sur un noyau. Mais, le concepteur doit aussi tenir compte des ressources de calculs requises par les primitives. Ainsi, en supposant que le concepteur a estimé que les ressources de calculs d'un noyau sont suffisantes au FIR complexe et au FIR réel. Celui-ci a pu déterminer que les ressources mémoires d'un seul noyau sont aussi suffisantes pour ces deux primitives en utilisant l'outil d'estimation. Par contre, les primitives du décodeur de Viterbi et du Filtre LMS nécessitent plus de ressources et elles devront être assignées à un noyau chacune.

Figure 5.12 Exemple de partitionnement sur Vocallo

5.3.4 Conclusion

Dans ce chapitre, une méthodologie d'estimation des ressources mémoires à haut niveau a été présentée. La méthodologie a été intégrée dans un outil d'analyse des scripts EMF qui génère un rapport d'estimation. Les résultats de l'estimation ont ensuite été comparés avec leurs équivalents sur la plateforme Vocallo. Ceux-ci ont démontré que le l'estimation est plausible dans la mesure où le script MATLAB et l'implémentation sont similaires. Dans les cas où des différences dans l'utilisation des tampons sont présentes les résultats peuvent diverger. De plus, la méthode d'estimation ne tient pas compte des possibilités d'optimisations de la plateforme comme l'utilisation de données avec des tailles qui ne sont pas standard

CONCLUSION

Il a été démontré que par leur nature, les applications de traitement de signal et plus spécifiquement les applications de télécommunications sans fil sont dominées par les mouvements et entreposages de données. Ainsi, l'impact de l'utilisation de la mémoire d'une application, qui sera implémentée sur une plateforme matérielle avec des ressources définies, peut entrainer un non-respect des spécifications. Il en est ressorti qu'une connaissance des besoins en ressources mémoires de l'application est nécessaire dès les premières phases de développement. Cela est d'autant plus important dans le cas des applications de télécommunications numériques sans fil où les plateformes visées sont des processeurs DSP avec des ressources limitées. De plus, les spécifications des standards de télécommunication sans fil tel que l'UMTS sont contraignantes en termes de temps de traitement des données. Or, il s'avère qu'une nouvelle application de traitement de signal est d'abord validée avec des outils de modélisation haut niveau, tel que le logiciel Matlab et son environnement graphique Simulink. De plus, les applications étudiées dans le cadre de cette recherche, sont implémentées sur la plateforme multi-noyaux Vocallo de la compagnie Octasic. Ainsi, la problématique est d'estimer l'utilisation des ressources mémoires sur la plateforme d'Octasic à partir de représentations Matlab/Simulink d'applications de traitement de signal. Ce mémoire propose donc une méthodologie afin d'estimer les ressources mémoires nécessaires à une application implémentée sur le Vocallo, à partir d'une représentation Matlab/Simulink de celle-ci. Afin de parvenir à cette méthodologie, les applications visées par le projet OPÉRA ont d'abord été étudiées, ensuite les particularités de l'architecture Vocallo et les méthodes d'estimations existantes pour finalement développer la méthodologie proposée.

L'étude des applications a démontré que les applications de traitement du signal requièrent de l'espace mémoire pour leurs tampons d'entrées et de sorties. Il est aussi ressorti que l'accès à la mémoire doit avoir une bande passante suffisante afin de respecter les contraintes de temps d'une application. De plus, certains algorithmes, comme le décodeur de Viterbi, utilisent une grande quantité de mémoire lors du traitement de données. Ainsi, ces

applications peuvent nécessiter une quantité de mémoire supérieure aux ressources disponibles sur la plateforme matérielle.

La méthodologie d'estimation proposée dans ce document cible une plateforme matérielle donnée. Alors, celle-ci a donc été étudiée en détail afin de déterminer ses principales contraintes. Cela s'avère crucial, puisque contrairement aux contraintes logicielles, celles de la plateforme sont fixes. En effet, les contraintes matérielles dépendent des caractéristiques de la plateforme, et dans le cas des DSP comme le Vocallo, elles sont limitées en termes de ressources spécifiques disponibles. Ainsi, l'étude du Vocallo a permis de faire ressortir que la quantité de mémoire embarquée, le nombre de registres et la bande passante vers la mémoire extérieure sont les contraintes matérielles qui détermineront si une application peut être implémentée sur le circuit. Donc, la méthodologie proposée fournit une estimation de l'utilisation de la mémoire locale et de la bande passante vers la mémoire externe d'une application. Mais, l'estimation du nombre de registres nécessaires n'a pas été réalisée. En effet, un même registre peut être réutilisé plusieurs fois dans une même application et cela sans vraiment nuire aux performances.

Afin de définir une méthodologie d'estimation efficace, les méthodologies déjà proposées dans la littérature ont été examinées. Il en ressort que la plupart de ces méthodologies sont basées sur des représentations de l'application sous forme de boucles imbriquées et de tableaux multidimensionnels. Ce style de représentation haut niveau ne convient pas à ce projet de recherche puisque l'utilisation de Matlab et Simulink comme outil de modélisation restreint le type de représentations aux scripts Matlab et modèles Simulink. Ensuite, les méthodologies sont de types statiques ou dynamiques. Les méthodologies statiques calculent la quantité de mémoire nécessaire sans tenir compte des possibilités de réutilisation à l'intérieur d'une application. À l'opposé, les méthodes dynamiques explorent les possibilités de réutilisation de la mémoire et peuvent ainsi guider un concepteur lors de la phase d'optimisation de l'application. Par contre, les méthodes dynamiques sont plus complexes à intégrer puisqu'elles requièrent la représentation de boucles imbriquées et de tableaux

multidimensionnels sous la forme d'un CDFG. Ainsi, dans le cadre du projet, la représentation Matlab devrait être convertie dans un format intermédiaire afin d'être compatible avec les méthodes dynamiques. Alors, la méthodologie développée dans le cadre de ce projet sera du type statique. Cela permet dans un premier temps de valider la pertinence de l'utilisation de la représentation Matlab dans une méthodologie d'estimation des ressources mémoires. De plus, cette méthode permettra aussi de valider les méthodes plus évoluées développées dans la continuité du projet OPERA, en particulier la génération d'un graphe CDFG.

La réalisation de la méthodologie a été implémentée dans un logiciel d'analyse des scripts Embedded Matlab Function. L'analyse du fichier Simulink dans son ensemble a d'abord été envisagée, mais cette solution n'a pas été retenue, car elle ne permet pas l'analyse des fonctions les plus couteuses en ressources d'une application. Il a été démontré que le modèle Simulink est composé de blocs primaires et aussi de blocs faisant références à une fonction externe. Ces fonctions externes sont dans la plupart des cas des scripts écrits en code Matlab. Afin d'obtenir une estimation pertinente de l'utilisation des ressources mémoires, c'est-à-dire qui tient comptes des fonctions les plus couteuses d'une application, l'outil a été développé pour être compatible avec le langage Matlab. Par contre, Matlab est un langage trop haut niveau pour être utilisé directement afin d'obtenir une estimation fiable, c'est pourquoi l'outil analyse les scripts Embedded Matlab Function conçus pour être compatibles avec les plateformes Real-Time Workshop. Real-Time Workshop est un outil incorporé dans Simulink qui permet l'implémentation de fonctions Simulink sur des processeurs embarqués et DSP. Ainsi, les scripts EMF contiennent des informations supplémentaires sur la taille des tableaux et les types de données employées. Il a été démontré dans ce document que l'estimation des ressources mémoires d'une application, implémentée sur le Vocallo, est suffisamment précise lorsqu'elle est implémentée en se basant sur le script EMF. L'automatisation de la méthode d'estimation dans un logiciel d'analyse des scripts procure des résultats rapidement pour différentes formes d'implémentation d'une même application. Le concepteur peut alors comparer les différentes saveurs d'implémentations et déterminer

laquelle est la plus adéquate. De plus, le logiciel d'estimation pourra être intégré dans une méthodologie de développement basée sur des modèles Simulink.

Donc, il a été démontré qu'il est possible d'obtenir une estimation fiable, de l'utilisation des ressources mémoires d'une application de traitement du signal à partir d'un modèle haut niveau. Les estimations ont été réalisées sur plusieurs fonctions élémentaires de traitement du signal, et ce, à partir d'un script EMF. Les estimations ont été réalisées grâce à un logiciel d'analyse automatique et validées avec leurs implémentations sur la plateforme Vocallo. La validation a démontré que les estimations sont précises sauf dans les cas où l'algorithme de l'implémentation diffère de celui du script EMF.

RECOMMANDATIONS

Les résultats présentés dans le présent document, ainsi que les outils développés pendant la recherche doivent être intégrés dans la méthodologie globale du projet OPERA. Cette section présente donc certaines recommandations afin de permettre l'intégration et l'évolution des outils dans la continuité du projet. Ainsi, depuis l'obtention des résultats, l'environnement de développement du Vocallo a été amélioré et des travaux sur la création de modèles CDFG à partir du modèle Simulink ont été effectués.

Premièrement, afin d'intégrer l'outil d'analyse des scripts EMF dans une méthode d'analyse des fichiers Simulink, l'interface de l'outil a été modifié. Le passage des paramètres de la taille des tampons d'entrées et sorties s'effectue par un fichier de configuration qui contient les informations nécessaires. Le format du fichier permet de définir plusieurs tampons et leurs types de données. Ainsi, lors d'une analyse du modèle Simulink, les informations d'entrées et sorties d'un bloc EMF sont extraites du fichier Simulink et inscrites dans le fichier de configuration. Le script d'analyse est ensuite appelé en passant en paramètre le code contenu à l'intérieur du bloc EMF et le fichier de configuration. Finalement, l'outil global se sert des informations contenues dans le rapport généré par le script lors de la création du modèle CDFG.

Par contre, le script d'analyse peut seulement estimer l'utilisation des ressources mémoires et les besoins en ressources de calcul ne le sont pas. Le principal obstacle à l'estimation des ressources de calcul à partir d'un script EMF est le niveau d'abstraction du langage Matlab. En effet, dans le langage Matlab, il est possible d'effectuer des opérations sur des matrices multi-dimensions sans avoir recourt à des fonctions ou opérateurs spéciaux. Contrairement au langage C où des opérations telle que la multiplication vectorielle doivent être implémentées à l'aide de boucles. Ainsi, l'estimation des ressources de calcul est impossible en analysant directement le code Matlab même dans le cas des EMF. Toutefois, il s'avère possible de générer du code C à partir des scripts EMF en utilisant l'outil Real-Time

Workshop de Simulink. Les performances du code généré pour le filtre LMS ont été comparées avec celui du code C non optimisé, écrit à partir du script EMF, en effectuant un profilage sur la plateforme Vocallo. Le code écrit ne contenait pas d'optimisation au niveau du code C, ni de code assembleur qui auraient pu tirer parti des instructions de traitement des nombres complexes ou d'opérations SIMD. Les résultats du profilage sont présentés dans le Tableau 6.1. Il en ressort que les performances du code généré par RTW sont légèrement supérieures. Ce qui permet d'affirmer qu'il est possible d'utiliser ce code afin d'estimer les ressources de calculs nécessaires. Ainsi, en utilisant un outil de génération de CDFG tel que SUIF, il serait possible d'intégrer les ressources nécessaires au traitement des données d'un bloc EMF au graphe du modèle Simulink.

Tableau 6.1 Performances du filtre LMS sur la plateforme Vocallo

Version du filtre LMS	Temps d'exécution (s)	Nombre d'instructions exécutés
Code C	41,56E-06	13787
Code C RTW	37,68E-06	7291

Une autre possibilité d'évolution ou plus précisément d'adaptation de l'outil d'analyse deviendra possible avec l'arrivé à maturité d'un compilateur C optimisant pour le Vocallo. Il ne sera plus nécessaire d'écrire du code en langage assembleur afin d'obtenir des performances optimale. Cela rendra possible l'utilisation du code C généré par le RTW dans des applications exécutées sur le Vocallo. Par contre, RTW créer sa propre structure de données et utilise plusieurs types de données modifiées. Les déclarations des structures sont contenues dans un fichier d'entête et la définition des tampons dans un autre fichier. De même que les définitions des types de données utilisés par RTW sont dans un troisième fichier. Ainsi, pour des applications de traitement du signal complexes il s'avère ardu d'effectuer le suivi des structures de données du programme et par le fait même des ressources mémoires nécessaires à son exécution. Le script d'analyse pourrait être modifié afin d'utiliser les informations extraites du script EMF pour analyser les fichiers d'entête et fournir un rapport de l'utilisation de la mémoire par le programme C. L'outil ferait

abstraction des types de données propre à RTW et fournirais un rapport en octets pour chacune des structures de l'application. Le concepteur pourrait aussi vérifier si RTW génère des structures redondantes et optimiser l'application.

Finalement, l'intégration de l'outil dans l'environnement du projet OPERA est probablement la continuité la plus probable de la présente recherche. Cette intégration est d'ailleurs en cours et utilise la version modifiée du script d'analyse de même que l'outil d'analyse du fichier Simulink décrit dans la section 5.1.1.

LISTE DE RÉFÉRENCES BIBLIOGRAPHIQUES

Amellal, S., et B. Kaminska. 1993. « Scheduling of a control data flow graph ». In *Circuits and Systems, 1993., ISCAS '93, 1993 IEEE International Symposium on*. p. 1666-1669 vol.3.

Ashok, Bindra. 2007. « Tackling complex signal-processing tasks for 3G LTE ». *R.F. Design*. (may).

Balasa, F., F. Catthoor et Man Hugo De. 1995. « Background memory area estimation for multidimensional signal processing systems ». *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 3, n° 2, p. 157-172.

Berkeley Design Technology, inc. (BDTI) 2007. *BDTI DSP Kernel Benchmarks*. <http://www.bdti.com/products/services_bdti_benchmarks.html>. Consulté le 06/08/2007.

Chia-Jeng, Tseng, et D. P. Siewiorek. 1986. « Automated Synthesis of Data Paths in Digital Systems ». *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 5, n° 3, p. 379-395.

Farhang-Boroujeny, B. 1998. *Adaptive Filters Theory and Application*. Chichester, UK: John Wiley & Sons.

Ferrari, A., et A. Sangiovanni-Vincentelli. 1999. « System design: traditional concepts and new paradigms ». *Computer Design, 1999. (ICCD '99) International Conference on*, p. 2-12.

Florin, Balasa, Zhu Hongwei et I. Luican Ilie. 2007. « Computation of Storage Requirements for Multi-Dimensional Signal Processing Applications ». *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 15, n° 4, p. 447-460.

Gerald Aigner, Amer Diwan, David L. Heine, Monica S. Lam, David L. Moore, Brian R. Murphy, Constantine Sapuntzakis. 1999. « An Overview of then SUIF2 Compiler Infrastructure ». Stanford University: <http://suif.stanford.edu/suif/suif2/doc-2.2.0-4/>.

Grun, P., F. Balasa et N. Dutt. 1998. « Memory size estimation for multimedia applications ». In., p. 145-149.

Hendrix, Henry. 2002. « Viterbi Decoding Techniques for the TMS320C54x ». Dallas, Texas: Texas Instruments.

Hennessy, John L., et David A. Patterson. 2003. *Architecture des ordinateurs, Une approche quantitative*, 3e. Vuibert Informatique.

Keutzer, K., A. R. Newton, J. M. Rabaey et A. Sangiovanni-Vincentelli. 2000. « System-level design: orthogonalization of concerns and platform-based design ». *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 19, n° 12, p. 1523-1543.

Kjeldsberg, P. G., F. Catthoor et E. J. Aas. 2003. « Data dependency size estimation for use in memory optimization ». *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 22, n° 7, p. 908-921.

Kurdahi, F. J., et A. C. Parker. 1987. « REAL: A Program for REgister ALlocation ». In., p. 210-215.

Lapsley, Phil, et Garrick Blalock. 1996. « How to estimate DSP processor performance ». *IEEE Spectrum*, vol. 33, n° 7, p. 74-78.

Neuvo, Y. 2004. « Cellular phones as embedded systems ». Solid-State Circuits Conference, 2004. Digest of Technical Papers. ISSCC. 2004 IEEE International.

Octasic Inc. 2008. *Vocallo : The Expandable Media Gateway Solution*. <http://www.octasic.com/en/products/vocallo/overview.php>.

Panda, P. R., F. Catthoor, N. D. Dutt, K. Danckaert, E. Brockmeyer, C. Kulkarni, A. Vandercappelle et P. G. Kjeldsberg. 2001. *Data and memory optimization techniques for embedded systems*, 6. ACM Press, 149-206 p.

Rysavy, Peter. 2006. « Mobile Broadband: EDGE, HSPA & LTE ». 3G Americas.

S.Cormier. 2006. *Implementing a Viterbi decoder for UMTS on a Vocallo multi-DSP device*. Octasic inc.

Säckinger, Eduard. 1997. *Evolution of Microprocessor & DSP Architecture*. Bell Laboratories , Lucent Technologies.

Sangiovanni-Vincentelli, A. 2007. « Quo Vadis, SLD? Reasoning About the Trends and Challenges of System Level Design ». *Proceedings of the IEEE*, vol. 95, n° 3, p. 467-506.

Sklar, Bernard. 2001. *DIGITAL COMMUNICATIONS Fundamentals and Applications (2nd edition)*. Upper Saddle River, N.J.: Prentice-Hall.

Stalling, William. 2003. *Computer organization and architecture : designing for performance*, 6th ed. Upper Saddle River, N.J.: Prentice Hall.

The MathWorks, Inc. 2007. « Getting Started with Simulink ». <http://www.mathworks.com/access/helpdesk/help/pdf_doc/simulink/sl_gs.pdf>.

Thiele, L., E. Wandeler et S. Chakraborty. 2005. « Performance analysis of multiprocessor DSPs: a stream-oriented component model ». *Signal Processing Magazine, IEEE*, vol. 22, n° 3, p. 38-46.

Verbauwhede, I. M., C. J. Scheers et J. M. Rabaey. 1994. « Memory Estimation for High Level Synthesis ». In., p. 143-148.

Viterbi, A. 1967. « Error bounds for convolutional codes and an asymptotically optimum decoding algorithm ». *Information Theory, IEEE Transactions on*, vol. 13, n° 2, p. 260-269.