UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN GÉNIE ÉLECTRIQUE

PAR
SIMON BOUCHER

IMPLÉMENTATION SUR DSP D'ALGORITHMES DE DÉTECTION MULTI-UTILISATEUR POUR LES SYSTÈMES DS-CDMA

SEPTEMBRE 2007

Résumé

Dans une cellule, un certain nombre d'usagers peuvent se connecter au réseau en occupant une même fréquence grâce à la technologie d'accès multiple par division de code (*CDMA*). Étant sur la même fréquence, les signaux usagers créent de l'interférence entre eux, dite d'accès multiple (*MAI*), ce qui empêche le bon recouvrement du signal désiré au récepteur de la station de base, et du fait même, réduit la capacité d'utilisateurs du réseau. Des algorithmes nommés détecteurs multi-utilisateur (*MUDs*) permettent un meilleur recouvrement des signaux en éliminant la MAI et donc permettent théoriquement une meilleure capacité au réseau. Comme les MUDs les plus performants sont extrêmement complexes, les unités de traitement ne sont pas assez puissantes pour traiter les signaux d'un grand nombre d'usagers dans les temps requis, ce qui encore une fois, réduit la capacité du réseau. C'est ainsi que des MUDs allouant un compromis performance versus complexité ont été proposés, dont l'algorithme CF-MUD, lequel ne présente pas encore d'évaluation de complexité en implantation réelle. Ce projet évalue donc le RAKE qui servira de référence de plancher, le MPIC comme référence de plafond et CF-MUD étant l'algorithme d'intérêt.

Ce mémoire évalue la complexité de différents algorithmes *MUDs* afin de déterminer lesquels sont les plus susceptibles d'être implantés sur processeur numérique de signal (*DSP*) tout en rencontrant les contraintes temps réel des spécifications régissant les systèmes cellulaires de troisième génération (*3G*).

L'unité de traitement cible pour l'algorithme est le DSP TMS320C6416 (*C6416*) de Texas Instruments (*TI*). Son architecture est à virgule fixe et son rendement optimal de calcul s'effectue sur une résolution de 16 bits. Les algorithmes sont simulés sur une plate-forme Matlab® fonctionnant à virgule flottante et avec une résolution de 16 bits. Nous avons donc modifié la plate-forme pour que la simulation s'effectue en virgule fixe et avec une résolution équivalente à celle du DSP. Ceci a requis une évaluation stricte de l'amplitude du signal d'entrée et des coefficients générés par l'estimateur de canal afin de pouvoir les quantifier convenablement. Aussi, les algorithmes ont été modifiés de sorte à permettre la mise à l'échelle des variables et des signaux résultants en cours de traitement.

La contrainte de temps réel du traitement des signaux est très élevée. Ceci impose que les algorithmes MUD soient codés de manière à utiliser en tout temps les ressources disponibles du DSP. Par conséquent, les MUDs ont été programmé en C et optimisés pour l'architecture du C6416 à l'aide de l'outil d'optimisation offert par le logiciel l'utilisation et de programmation des DSPs de TI, Code Composer Studio (*CCS*). Ce dernier construit des pipelines utilisant le parallélisme du C6416 et remplit les cases de retard par des instructions utiles, permettant de minimiser la complexité des programmes.

L'algorithme CF-MUD a été codé directement en langage assembleur optimisé afin de comparer la complexité résultante de ce programme avec celle du même programme généré par l'outil d'optimisation de CCS. L'optimisation manuelle assure d'introduire tous les principes d'optimisation de code tels que le pipeline, le parallélisme, le remplacement des cases de retard par des instructions utiles, l'optimisation de la largeur des mots de chargement, le déroulement des boucles et la fusion de boucles imbriquées.

Finalement, ce projet permet de situer la complexité de l'algorithme CF-MUD par rapport à celles du RAKE et du MPIC et en même temps de valider le taux de complexité du code généré par l'outil d'optimisation de CCS. Des recherches sont en cours afin de permettre au CF-MUD de converger plus rapidement lors de l'adaptation de ses coefficients, ce qui devrait réduire considérablement sa complexité.

Liste des abréviations

2G	Deuxième Génération
3G	Troisième Génération
ASIC	Application Specific Integrated Circuit
AWGN	Additive White Gaussian Noise
C6416	DSP TMS320C6416
C64x	Famille de DSP TMS320C64x, incluant le C6416
CCS	Code Composer Studio
CF-MUD	Cascade Filter MultiUser Detector
DS-CDMA	Direct Sequence CDMA
DSP	Digital Signal Processor
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
IC	Interference Cancellation
IMT	International Mobile Telecommunications
ISI	InterSymbol Interference
LMS	Least-Mean-Square
MAI	Multiple Access Interference

MC-CDMA	Multi-Carrier CDMA
MF	Matched Filter
MIPS	Millions d'Instructions Par Seconde
MPIC	Multistage Parallel Interference Cancellation
MUD	MultiUser Detector
OVSF	*Orthogonal Variable Spreading Factor*
PIC	Parallel Interference Cancellation
SIC	Successive Interference Cancellation
SRAM	Static Random Access Memory
SUD	Single User Detector
TI	Texas Instruments
UCT	Unité Centrale de Traitement
UIT	Union Internationale des Télécommunications
UT	Unité de Traitement
VelociTI	VEry LOng Code Instruction of TI
WCDMA	Wideband Code Division Multiple Access

Liste des symboles

Eb/No	rapport de l'énergie binaire sur l'énergie du bruit en décibel
I	énergie de l'interférence
L	nombre de multi-trajets
mu	pas d'adaptation
mudegree	degré binaire du pas d'adaptation ($mu = 2^{mudegree}$)
N	nombre de données à traiter
N_adapt	nombre de données d'entraînement
N_chips	nombre de chips à traiter
Nc	OVSF utilisé ou facteur d'étalement
P	énergie du signal désiré
Q	degré binaire de quantification (facteur de quantification = 2^Q)
U	nombre d'utilisateurs
N_étage	nombre d'étage d'annulation d'interférence du MPIC
Nw	nombre de coefficients des filtres du bloc Signature de CF-MUD
Nv	nombre de coefficients du filtre du bloc Détection de CF-MUD
N_adapt_filtre	nombre de données d'adaptation par filtre
N_adapt_chips	nombre de chips émises à l'adaptation

mu_bloc1	pas d'adaptation du bloc Signature
mu_bloc2	pas d'adaptation du bloc Détection
N_SF	nombre de filtres à utiliser par rapport à OVSF
C/A	rapport de complexité du code C sur le code assembleur
f	fréquence

Table des matières

Chapitre 1
Introduction

Les télécommunications sans fil ont pris de l'ampleur durant la Seconde Guerre Mondiale grâce aux développements dans le domaine des semi-conducteurs [KOR01]. Durant les années 1980, la première génération apparaît avec les systèmes cellulaires mobiles, entièrement analogiques, qui se sont répandus suivant différentes applications et plusieurs standards. La deuxième génération apparue plus tard avec les systèmes numériques. Cette génération de communication cellulaire permet une capacité beaucoup plus importante que la première. Elle est dédiée au transfert de la voix, mais aussi aux données d'applications ne requérant guère un fort débit. Elle est gérée par seulement quelques standards principaux.

Aujourd'hui, la seconde génération de cellulaire ne suffit plus. En effet, l'accroissement de la diversité des services offerts demande la gestion d'applications nécessitant un haut débit binaire telles que la vidéoconférence ou l'Internet. De plus, la croissance du marché de la téléphonie cellulaire fait en sorte que la bande passante allouée n'est plus suffisante pour supporter de nouveaux utilisateurs [SAN01].

De là, la nécessité d'adopter de nouveaux standards pouvant supporter de forts débits binaires et un grand nombre d'utilisateurs. Ce mandat appartient à la troisième génération

(*3G*) de télécommunication cellulaire, dans laquelle l'Union Internationale des Télécommunications (*UIT*) vise à adopter un standard universel pouvant gérer de hauts débits et davantage d'utilisateurs. La norme IMT-2000 (*International Mobile Telecommunications 2000*) est la norme officielle de l'UIT pour la standardisation des systèmes [SAN01].

Plusieurs réseaux de télécommunication de la 3G utilisent des systèmes cellulaires basés sur la technologie d'accès multiple WCDMA (*Wideband Code Division Multiple Access*) [SAN01]. Cette technologie présente deux intérêts majeurs. Le premier est qu'elle permet d'atteindre d'importants débits binaires grâce à sa large bande. Le second est que sa capacité ne dépend pas de la dimension de la bande de fréquence allouée (ressource rare), mais plutôt du niveau d'interférence dans le réseau [MA02], [GUO00]. La capacité ce définit par le nombre d'usagers connectés simultanément à une même station de base (antenne).

Cette interférence, dite d'accès multiple (*MAI*), est directement causée par les signaux entre usagers, donc elle dépend du nombre d'usagers à l'intérieur du réseau. Plusieurs techniques de réduction d'interférence existent, dont la configuration et la sectorisation d'antennes, les réseaux d'antennes, et les antennes intelligentes ou adaptatives. Il existe aussi des techniques d'annulation d'interférence (*IC*), dont appartient la détection multi-utilisateur [SAN01], [IEC98], [THO99], [GUO00], [DUE95].

Les trois dernières techniques font toujours objet de recherches actives du fait qu'elles s'avèrent théoriquement efficaces et non exclusives, mais aussi parce qu'elles présentent

une très grande complexité d'implantation pratique. En effet, le nombre d'opération arithmétique est tellement élevé que sa mise en œuvre sur DSP, FPGA ou ASIC demeure un problème non trivial en terme de coût et consommation de puissance afin de respecter les contraintes temporelles et d'exécution des standards de la 3G. L'intérêt de la présente étude porte sur la détection multi-utilisateur (*MUD*). Au niveau d'une station de base, laquelle est munie d'équipements électroniques très performants servant de récepteur aux signaux émis par les mobiles des usagers, le MUD permet l'extraction des signaux de chaque usager du signal global reçu au départ, et par le fait même, l'élimination d'une grande quantité d'interférences [SAN01] [DUE95].

1.1 Problématique

Comme mentionné à l'introduction, la capacité d'un système basé sur le CDMA est limitée par la quantité d'interférences présente dans le réseau. Ces interférences résultent du bruit général associé à l'équipement et au canal de transmission, de l'interférence inter-symbole (*ISI*) et principalement de la MAI. Cette dernière provient de l'utilisation d'une même fréquence de transmission par plusieurs utilisateurs (principe même du CDMA) pour se connecter à une même station de base. Par conséquent, un excédant d'usagers dans le réseau (même fréquence, même station de base) amène le rapport P/I, P représente l'énergie du signal désiré et I l'énergie de l'interférence, en dessous de un, ce qui implique que le signal désiré (désétalé) ne peut plus être recouvert des autres signaux (Figure 1.1).

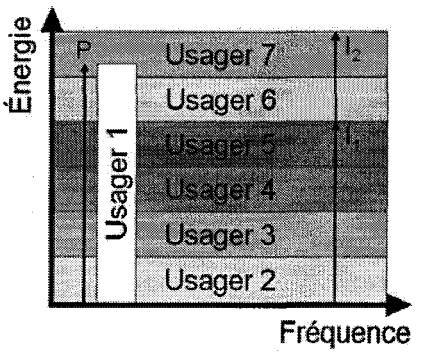


Figure 1.1 Effet de l'interférence d'accès multiple (MAI) sur la capacité du système

Différentes techniques d'annulation d'interférence existent, dont celle étudiée dans ce projet, la détection multi-utilisateur. Certains MUDs sont extrêmement performants pour éliminer les interférences et permettent donc d'accroître considérablement la capacité d'un système [MA02], [GUO00], [HAL95]. Hélas, ces mêmes détecteurs présentent une complexité d'implantation très élevée. Les équipements électroniques qui les implantent peuvent être constitués d'ASICs (*Application Specific Integrated Circuit*), de FPGA (*Field Programmable Gate Array*) ou de DSP (*Digital Signal Processor*). Un ASIC suffira à la tâche, mais sa dimension et sa consommation risque d'être trop importantes, et de plus, il présente l'inconvénient d'être dédié à une application précise, donc il est très coûteux. Les FPGA et DSP sont reprogrammables, mais peuvent voir leur capacité de calcul saturée. Effectivement, le récepteur de la station de base doit traiter les trames, contenant 38400 données, de tous les utilisateurs qui y sont connectés en dedans de 10 ms, et ce pour différentes qualités de service basées sur les OVSFs (*Orthogonal Variables Spreading Factor*) standards [SAN01]. Ce qui implique que, quel que soit le nombre d'usagers connectés et l'OVSF utilisé, le récepteur n'a que 10 ms pour traiter une trame de 38400 données de chaque usager et ensuite passer à une seconde trame. Par conséquent, la

complexité mathématique des algorithmes de détection multi-utilisateur limite aussi la capacité du système.

Plusieurs solutions existent pour régler ce problème, mais chaque solution comporte ses avantages et ses inconvénients, ainsi que ses difficultés et ses coûts. Bien sûr, on peut effectuer du traitement en parallèle avec plusieurs unités de traitement (coûteux et difficile à réaliser), on peut dédoubler le matériel pour augmenter directement la capacité du système (très coûteux), et aussi, on peut concevoir de nouveaux MUDs ou « simplement » trouver un compromis entre la performance et la complexité des algorithmes existants. Les deux dernières solutions suscitent particulièrement l'intérêt des unités de recherche puisqu'elles sont plus plausibles et plus faciles à mettre en oeuvre. De là l'intérêt d'évaluer la complexité d'implantation sur processeur numérique des signaux (*DSP*) de différents MUDs.

1.2 Objectifs

L'objectif principal de ce projet est d'évaluer la complexité de différents MUDs afin de déterminer lesquels sont les plus susceptibles d'être implantés sur DSP tout en rencontrant les contraintes temps réel des spécifications régissant les systèmes cellulaires. L'évaluation de la complexité se fait relativement au nombre de cycles d'instruction nécessaires et à la quantité mémoire requise pour effectuer le traitement. Les algorithmes à évaluer se retrouve dans la littérature relative à la conception de récepteurs multi-utilisateur pour les systèmes cellulaires spécifiant le WCDMA comme technologie d'accès multiple. Les sous-objectifs permettant de rencontrer l'objectif principal sont :

1. Étude des algorithmes de détection multi-utilisateur tels que : RAKE, MPIC et CF-MUD.
2. Étude du DSP TMS320C6416 de Texas Instruments (*TI*), plus spécifiquement de son architecture et des opérations qu'il effectue.
3. Utilisation et modification d'une plate-forme Matlab® simulant le comportement d'une station de base d'un système WCDMA utilisant différents MUDs comme récepteur et simulant aussi les émetteurs mobiles ainsi que leur canal de transmission afin de générer automatiquement les paramètres et les données nécessaires au traitement effectué par le DSP.
4. Programmation des algorithmes en C pour fin de comparaison de leur complexité et programmation optimale de l'algorithme CF-MUD en langage assembleur afin d'offrir le minimum de complexité possible et aussi d'offrir un traitement comparable à celui que la théorie permet.

1.3 Méthodologie

Les algorithmes à évaluer ont déjà été codés en langage Matlab® sur une plate-forme de simulation de MUDs [DAH01], [DAH02]. Par conséquent, cette plate-forme sert directement de référence de traitement pour la plate-forme DSP. La plate-forme Matlab® permet la génération aléatoire des données, la simulation de l'émetteur (portable), la génération du canal à trajets multiples, celle du bruit Gaussien additif blanc (*AWGN*) (*Additive White Gaussian Noise*) ainsi que celle du signal reçu au récepteur. Le récepteur DSP reçoit le signal des émetteurs simulés par Matlab® pour effectuer son traitement.

Le traitement qu'effectue Matlab® à la réception des données se fait à l'aide de programmes algorithmiques reflétant intégralement les équations théoriques de l'algorithme en cause. Par conséquent les résultats reflètent le comportement idéal de l'algorithme et même celui du canal, lequel ne présente pas de non-linéarité. De plus, cette plate-forme effectue un travail sur des nombres en point flottant de 64 bits de résolution, ce qui lui permet d'atteindre une très haute précision de traitement.

Les programmes Matlab® servent de référence à la programmation des programmes implantés dans le DSP. Ceux-ci sont codés une première fois en C afin d'évaluer leur comportement avec un langage plus bas que celui de Matlab®. Visual Studio sert de plate-forme de validation des programmes C pour les nombres à point flottant de 64 bits de résolution. Ensuite, les programmes sont modifiés pour fonctionner à point fixe avec une résolution de 32 bits et réadapter encore une fois pour fonctionner avec une résolution de 16 bits. Cette dernière permet d'atteindre un traitement optimal en termes de vitesse de traitement avec le DSP C6416, dont l'architecture interne est conçue pour travailler de manière optimale en 16 bits.

Les programmes à point fixe et à résolution 16 bits sont bien sûr beaucoup moins précis que les programmes Matlab® originaux. Pour atteindre des performances qualitatives similaires à celles de la théorie, une gestion stricte de la quantification s'impose. Donc la dynamique des signaux entrant au récepteur MUD doit être connue et gérée convenablement. Celle-ci est donc ramenée entre 1 non inclus et -1 inclus, puis quantifiée avec un facteur maximal qui ne permet pas les débordements. Pour que la gestion de la quantification ne soit pas trop

coûteuse dans le DSP, le facteur de quantification est de degré binaire, ce qui permet d'effectuer des opérations de décalage binaire au lieu de la division.

Les programmes à point fixe de résolution 16 bits gérant la quantification sur Visual Studio deviennent aussi des programmes de référence lorsque leur fonctionnement reflète celui de Matlab®. Par conséquent, le programme CF-MUD est codé en assembleur pour atteindre le code le moins complexe possible. On utilise les instructions optimales fournies par l'architecture du C6416, dont certaines permettent le dédoublement d'unités fonctionnelles. La vérification de l'intégrité de leur fonctionnement ainsi que leur optimisation finale s'effectuent à l'aide de Code Composer Studio (*CCS*), le logiciel d'exploitation des DSP de TI. Leur complexité de traitement est évaluée arithmétiquement avec Matlab® et validée à l'aide de CCS, lequel inclut une routine de comptage du nombre de cycles exécutés par le DSP. La complexité de la mémoire est déterminée à l'aide de Matlab® puisqu'il est facile de connaître la dimension des vecteurs de données.

1.4 Organisation du mémoire

Pour commencer, le chapitre 2 traite des bases de la détection multi-utilisateur. Plus spécifiquement en discutant de la technologie d'accès multiple utilisée à la section 2.1, du principe de la détection multi-utilisateur à la section 2.2, du récepteur RAKE, du détecteur MPIC et du détecteur CF-MUD à la section 2.3 et d'une évaluation comparative de leurs performances de traitement respectives à la section 2.4.

Le chapitre 3 traite de l'implantation des algorithmes en virgule fixe. On discute de la plate-forme de simulation sur Matlab® à la section 3.1, du principe de la quantification des

signaux à la section 3.2, et de la quantification des signaux générés par la plate-forme Matlab® ainsi que de la mise à l'échelle des variables à l'intérieur même des algorithmes, à la section 3.3.

Le chapitre 4 traite de l'implantation des algorithmes sur DSP. La section 4.1 discute de la structure du TMS320C6416, la section 4.2 élabore sur les principes d'optimisation du code du DSP, la section 4.3 traite de la programmation et de l'optimisation des algorithmes et la section 4.4 parle de la validation et de l'évaluation des programmes construits.

Le chapitre 5 expose les résultats de l'évaluation de la complexité de l'implantation des algorithmes. La section 5.1 discute des paramètres et des critères d'évaluation de la complexité, la discussion sur les résultats d'évaluation de la complexité des algorithmes RAKE, MPIC et CF-MUD sont donnés aux sections 5.2, 5.3 et 5.4 respectivement, tandis que les graphiques et les tableaux des résultats sont affichés aux sections 5.5 et 5.6 respectivement.

Chapitre 2
Algorithmes de détection multi-utilisateur

Déterminer la complexité de différents algorithmes en se basant sur leur implantation impose la connaissance de la théorie inhérente aux méthodes. Par conséquent, les principes relatifs à la détection multi-utilisateur doivent être abordés. Ce chapitre se veut donc une introduction à la détection multi-utilisateur. Comme celle-ci est directement rattachée à sa technologie d'accès multiple, le CDMA doit être présenté en premier lieu, à la section 2.1. Les principes et les différentes catégories de MUDs seront traités à la section 2.2.

Ces catégories de MUDs possèdent différents niveaux de traitement lesquelles dépendent de la logique de traitement du détecteur. On retrouve des détecteurs linéaires, non linéaires, adaptatifs ou non-adaptatifs. La section 2.3 présente trois MUDs différents. Comme ces détecteurs utilisent des principes de traitement différents, ils n'éliminent pas les interférences au même degré. Une évaluation comparative des performances sera donc présentée à la section 2.4.

2.1 DS-CDMA

La technologie d'accès multiple DS-CDMA permet à plusieurs utilisateurs d'occuper la même fréquence, et leurs signaux sont séparés les uns des autres par le moyen d'un code

spécial (Figure 2.1) [KOR01]. Il existe quelques variantes de CDMA en fonction du type de codage utilisé. Notons le FH-CDMA (*Frequency Hopping CDMA*) et la variante étudiée ici, le DS-SS-CDMA (*Direct Sequence Spread Spectrum CDMA*) appelé seulement DS-CDMA.

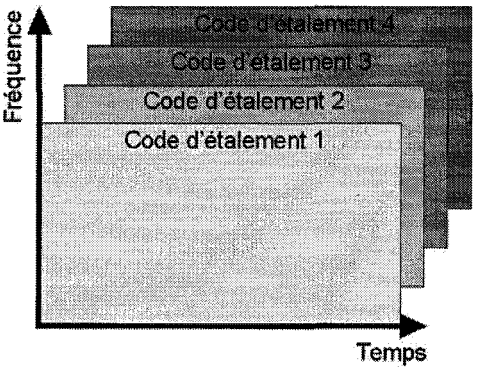


Figure 2.1 Principe du CDMA

Chaque utilisateur est assigné à un code distinct qu'on applique comme une seconde modulation. Ce code est utilisé pour transformer les signaux de l'utilisateur en une version codée à étalement de spectre (*spread-spectrum-coded*) du flot des données. Le récepteur utilise donc le même code d'étalement pour retransformer le signal étalé dans la version originale du flot des données. Les codes sont choisis avec une faible corrélation croisée entre eux. Ce qui signifie que, corrélé à la réception le signal étalé avec son code correspondant, désétale seulement le signal étalé par ce même code. Tous les autres signaux demeurent étaler sur une large bande passante. Par conséquent, seul le récepteur connaissant le code d'étalement correspondant peut extraire le signal original du signal à spectre étalé reçu. Le WCDMA est une forme de CDMA à large bande, sa bande s'étend sur 5 MHz [KOR01].

2.1.1 Étalement de spectre

La transmission à étalement de spectre est une technique qui transforme le signal original en une autre forme occupant une bande passante plus large que le signal original nécessiterait normalement [KOR01]. La transformation est appelée étalement. La séquence de données originale est multipliée ou traitée par OU exclusif avec un code d'étalement qui typiquement contient une bande passante beaucoup plus large que celle du signal original. Cette procédure est représentée à la Figure 2.2. Le désétalement s'effectue de la même façon que l'étalement, excepté que cette fois-ci, c'est le signal étalé que l'on traite, toujours avec le même code, pour retrouver le signal original. Les bits du code étalé sont appelés bribes (*chips*) pour les différentier des bits de la séquence de données, lesquels sont appelés symboles.

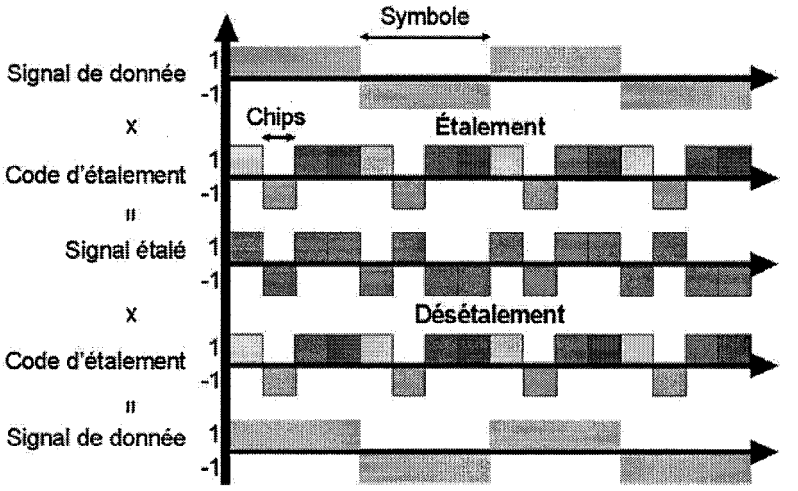


Figure 2.2 Procédure d'étalement et de désétalement [2]

Chaque utilisateur détient son propre code d'étalement. Le transmetteur et le récepteur utilisent le même code pour les deux transformations [KOR01]. Le rapport de fréquence entre les deux signaux se nomme gain de traitement (*processing gain*) ou plus

communément facteur d'étalement. Ce rapport signifie seulement le nombre de *chips* utilisées pour étaler une donnée d'un symbole. Plus ce rapport est faible, plus le chargement de données transporté par le signal est payant en terme d'information utile.

2.2 Détection multi-utilisateur

Dans un système CDMA idéal, le canal ne comporte pas de non-linéarité permettant au récepteur de seulement désétaler le signal reçu. Effectivement, les émetteurs étalent les signaux utilisateur à l'aide des codes orthogonaux parfaitement séparables [KOR01]. Dans la réalité, ceci est aussi vrai, mais le canal est imparfait et détruit l'orthogonalité des signaux étalés, ce qui entraîne des erreurs à la réception, puisque les signaux ne sont plus parfaitement séparables et que chaque usager de plus dans le réseau, ajoute un degré supplémentaire d'interférence MAI dans le signal d'entrée au récepteur. Les performances se détériorent en fonction du chargement du réseau en usagers. On peut donc utiliser le MUD pour estimer l'interférence générée par les autres utilisateurs afin de l'éliminer du signal d'entrée et de séparer le signal de l'usager en cours, ce qui permet d'obtenir des performances de traitement supérieures. Le concept fondamental de la détection multi-utilisateur est de considérer tous les utilisateurs comme des signaux utiles à chacun [HAL95]. Par conséquent, les signaux interférents sont utilisés pour le bénéfice de chaque utilisateur. Le détecteur optimal détecterait et recevrait simultanément tous les signaux et retrancherait les autres signaux du signal désiré. Cependant, les algorithmes MUDs optimaux sont trop complexes à implanter en pratique, par conséquent, on utilise des algorithmes sous-optimaux [KOR01]. Ceux-ci se divisent en deux catégories principales, la

détection conjointe ou linéaire, puis la détection par des techniques d'annulation d'interférence ou non linéaires [SAN01].

2.2.1 Détection conjointe ou linéaire

C'est une méthode générale qui utilise les décisions précédentes des autres utilisateurs pour éliminer l'interférence du signal de l'utilisateur désiré [SAN01]. On commence par ordonnancer les utilisateurs en ordre décroissant selon la puissance de leur signal, et ensuite on traite. Le traitement s'effectue en appliquant une transformée linéaire aux sorties des filtres appariés (*MF*) lesquels sont en charge d'éliminer le MAI [KOR01]. Ce type de détection est très performant, mais encore une fois, très complexe à intégrer, ce qui nécessite des compromis entre la performance et la complexité.

2.2.2 Techniques d'annulation d'interférence ou détection non linéaire

Ce type de technique consiste en l'estimation des interférences d'accès multiple et intersymbole, puis de les retrancher de l'interférence totale estimée [SAN01]. Les techniques d'annulation d'interférence étudiées dans le cadre du projet sont de types non linéaires. Il existe trois techniques principales d'annulation d'interférence à détecteur non linéaire : annulation successive de l'interférence (SIC), annulation de l'interférence en parallèle (PIC) et des techniques hybrides.

Le principe du SIC est d'ordonner les utilisateurs en fonction de leur puissance, puis d'estimer l'interférence de l'utilisateur ayant la plus forte puissance et de la retrancher du signal des autres utilisateurs [SAN01]. Cette technique n'est pas optimale puisque seulement le dernier utilisateur profite de la réduction totale de l'interférence causée par les

autres utilisateurs. Dans certains cas, il peut même y avoir divergence des résultats et sa complexité accroît proportionnellement au nombre d'utilisateurs.

Le principe du PIC est semblable à celui du SIC, mais ici, l'interférence de tous les utilisateurs est retranchée pour chaque utilisateur [SAN01]. C'est-à-dire qu'on effectue un premier traitement de filtrage, ensuite on utilise le résultat du traitement de tous les utilisateurs pour estimer le bruit réel (excluant le MAI), on ajoute ce bruit au signal recouvert de l'utilisateur en cours et effectue à nouveau le traitement, mais cette fois-ci, le MAI est éliminé. Le MAI est totalement éliminé si le canal est parfaitement connu et si la sortie du premier étage est parfaite. Ce traitement peut-être effectué en parallèle, et peut-être répété en plusieurs étapes pour augmenter la précision. Le PIC nécessite plus de complexité mathématique que le SIC.

Les techniques hybrides combinent les idées du PIC et du SIC. Traditionnellement, on effectue un groupement des signaux d'information des différents utilisateurs. Puis pour chaque groupe, une première détection en parallèle est effectuée et suivie d'une annulation d'interférence successive. Cette technique est un compromis entre performance et complexité.

2.2.3 Autres classifications

Les MUDs peuvent être classifiés selon différents qualificatifs en fonction de leur architecture. Ces qualificatifs ne sont pas exclusifs entre eux, au contraire, ils sont complémentaires. Donc, un même algorithme détient plusieurs qualificatifs. Les détecteurs centralisés effectuent le traitement du signal désiré en tenant compte des autres signaux des

usagers. Les détecteurs décentralisés ne tiennent pas compte des autres signaux pour traiter celui désiré, ce sont par définition des détecteurs à simple utilisateur (*SUD*). On utilise les SUDs aux récepteurs des mobiles, lesquels restituent seulement l'information d'un usager. Ensuite, on retrouve les détecteurs directs, lesquels, ne nécessitent pas des coefficients estimés du canal de transmission, ces algorithmes sont adaptatifs, ils déterminent eux-mêmes leurs coefficients. Les détecteurs indirects, au contraire, nécessitent la connaissance des coefficients, laquelle provient de l'estimateur de canal. La méthode indirecte nécessite l'inversion de la matrice des coefficients, ce qui est assez coûteux en temps de calcul et qui introduit des erreurs relatives à la quantification [DAH04].

2.3 Algorithmes

Comme le démontre les quelques classifications de MUD, il existe différentes méthodes pour séparer les signaux les uns des autres en minimisant l'effet négatif de l'interférence MAI. Bien entendu, ces algorithmes n'obtiennent pas des performances équivalentes et ne présentent pas les mêmes degrés de complexité. Afin de comparaisons futures, trois algorithmes sont décrits ici; le RAKE, le PIC à étages multiples (*MPIC*), et le CF-MUD. Le choix de ces algorithmes s'appuie sur l'importance qu'on leur donne dans la littérature. La description des ces algorithmes ne seront pas décrit dans ce mémoire puisqu'ils sont disponibles dans la littérature. À ce sujet, nous invitons le lecteur à consulter [VER98] et [DAH04]. Le RAKE est un SUD peu complexe utilisé dans les systèmes de deuxième génération (*2G*) en Amérique du Nord [DAH02]. Le MPIC est une méthode très performante utilisée comme référence de comparaison dans la littérature. Enfin,

l'algorithme CF-MUD est une nouvelle méthode allouant performance et moindre complexité [AXI04, HOQ06, DAH04].

2.3.1 Détecteur de base RAKE

Le RAKE est un détecteur linéaire décentralisé indirect. Son fonctionnement s'explique par une brève introduction sur la diversité de trajets. Le signal original propagé dans l'air (le canal) se propage par réflexion sur des obstacles avant que le récepteur du mobile ou de la station de base le capte sous différentes copies, décalées et retardées du signal transmis, provenant du trajet direct et des trajets de réflexion. Le détecteur RAKE permet la réception et la combinaison des signaux à trajets multiples, sans quoi, un signal individuel reçu au RAKE pourrait être trop faible pour produire le résultat désiré. La combinaison de ces signaux augmente le SNR du signal résultant. Le RAKE consiste en plusieurs doigts qui représentent des corrélateurs (Figure 2.3), lesquels désétalent les signaux à trajets multiples avec une copie locale convenablement retardée et décalée de la version originale du code d'étalement (combinée au code de brouillage) de la station de base. Ensuite les signaux résultants des différents trajets sont pondérés et additionnés entre eux pour produire un signal de puissance supérieure représentant le signal original [SAN01]. Le RAKE n'est pas un MUD puisqu'il n'élimine pas le MAI, c'est un SUD. Il est originalement conçu pour traiter qu'un seul utilisateur sur une même fréquence.

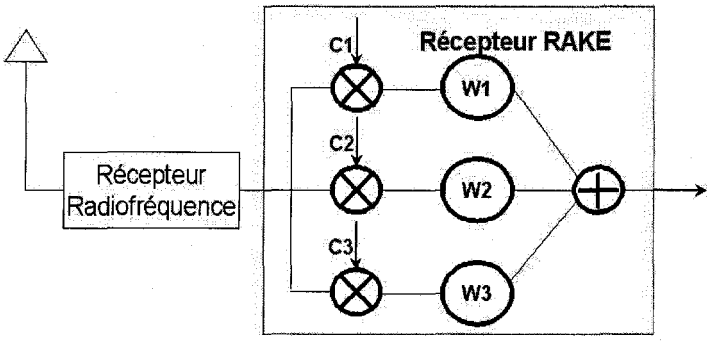


Figure 2.3 Récepteur RAKE à trois corrélateurs

2.3.2 Détecteur MPIC au niveau chip

Le détecteur MPIC est classé comme non linéaire, centralisé et indirect. Il est conçu d'un détecteur RAKE et de quelques étages de suppression de MAI (Figure 2.4). Premièrement, le signal entrant, une fois mémorisé, subit un premier traitement dans un RAKE conventionnel et les signaux restitués sont propagés vers le premier étage IC (Figure 2.5). L'étage IC, dans un premier temps, force les signaux restitués à 1 ou –1 et les considèrent comme étant les signaux usagers originaux. Ensuite, il imite le rôle des émetteurs en étalant les signaux avec leurs codes d'étalement respectifs, simule le canal en créant des versions désynchronisées des signaux (diversité de trajets), en effectuant une convolution du signal de chaque usager avec l'estimé des coefficients du canal respectif de chaque usager. Ces signaux, gardés en mémoire, représentent chaque signal reçu comme s'il était seul dans la cellule, donc sans MAI. Les nouveaux signaux isolés sont combinés ensemble pour produire l'estimé du signal entrant. Ce dernier, n'incluant pas le bruit, est soustrait du signal entrant mémorisé ce qui résulte en l'estimé du bruit, excluant la MAI. Comme ce bruit contient de l'information supplémentaire à tous les signaux utilisateur, il est additionné au signal isolé de chaque usager, lesquels sont traités individuellement par un autre RAKE, qui à son tour, propage les signaux vers un autre étage. C'est à l'étape

d'addition du bruit avec le signal isolé de chaque usager qu'on constate l'élimination de l'interférence, puisque seul le signal de l'usager en cours est considéré, contrairement au traitement du premier RAKE, où tous les signaux étaient combinés ensemble [HAL95].

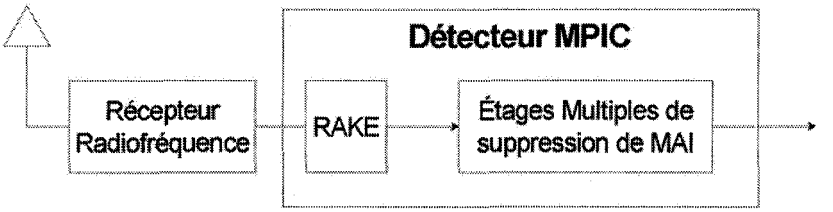


Figure 2.4 Détecteur MPIC

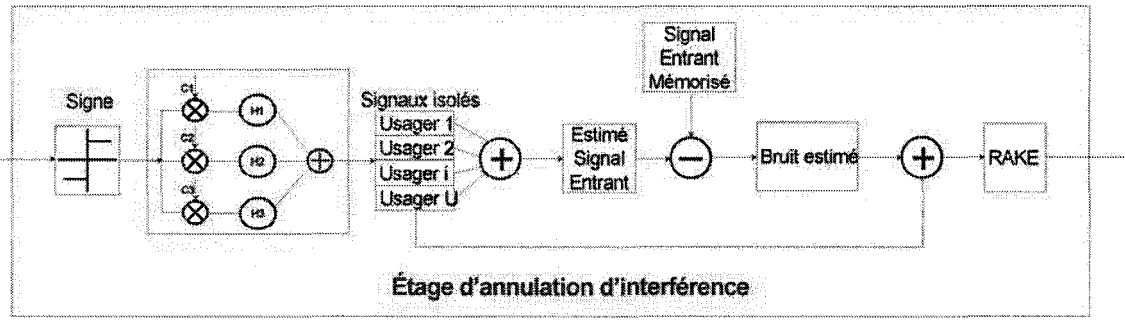
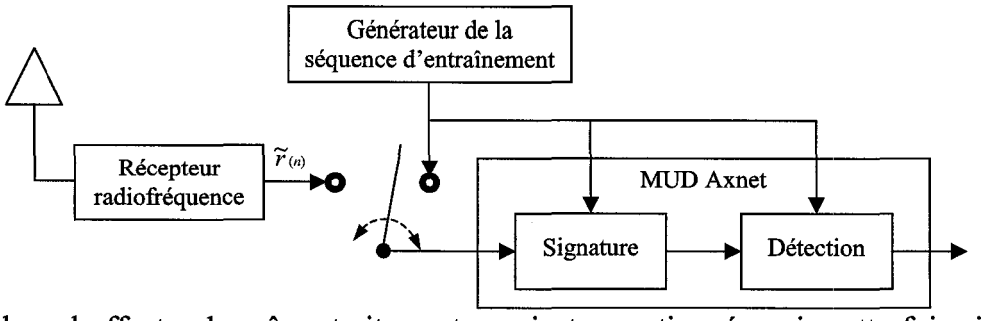


Figure 2.5 Étages IC

2.3.3 Détecteur CF-MUD

Le détecteur CF-MUD, aussi appelé Cascade Filter MUD (*CF-MUD*), se classifie parmi les MUDs linéaires, directs et centralisés. En effet, comparé aux deux précédents, CF-MUD adapte automatiquement les coefficients de ses filtres au lieu d'utiliser ceux fournis par l'estimateur de canal. Ce détecteur repose sur deux blocs, Signature et Détection, dans chacun d'eux, on retrouve une fonction d'adaptation et une fonction de détection (Figure 2.6). L'algorithme Écart des moindres carrés (*LMS*) exécute les fonctions d'adaptation et le filtre à réponse impulsionnelle finie (RIF ou *FIR*) exécute les fonctions de détection (Figure 2.7). Le principe de fonctionnement de CF-MUD se divise donc en deux opérations majeures, le bloc Signature restitue les signaux utilisateur et fonctionne en nombre

complexe, tandis que le bloc Détection élimine l'interférence résiduelle et fonctionne en nombre réel [AXI04]. Le bloc Signature adapte la signature de tous les utilisateurs sans connaissance préalable de leur code d'étalement. Ceci est fait à l'aide d'un signal entrant d'entraînement dont l'information de chaque utilisateur, qui y est contenue, est déjà connue par le détecteur. Ce signal étalé est donc propagé dans le LMS, qui à son tour, propage les coefficients adaptés vers la fonction de détection de ce même bloc, puis propage aussi les résultats du traitement du signal d'entraînement vers le bloc détection. Le FIR du bloc détection utilise les coefficients préalablement adaptés pour traiter le signal entrant et propage les signaux des utilisateurs vers le bloc Détection. Les premières données d'entraînement de chaque utilisateur virtuel sont éliminées dans le bloc détection, puisqu'elles ont été filtrées avec des coefficients insuffisamment adaptés. Donc, le LMS adapte ses coefficients en fonction des données résiduelles au moment précédent, présent et futur de chaque utilisateur pour chaque signal désiré. C'est-à-dire qu'ici, on n'utilise pas seulement les données du signal désiré, mais bien celles de tous les signaux, et qu'en plus, on utilise la donnée précédente, la donnée présente et la donnée suivante de chaque utilisateur pour adapter le filtre désiré au moment présent. Les coefficients sont propagés au



FIR réel lequel effectue le même traitement que juste mentionné, mais cette fois-ci, sans

adaptation et en utilisant 100% des données utilisateur restituées au bloc précédent par le FIR complexe.

Figure 2.6 Détecteur CF-MUD (anciennement nommé Axnet)

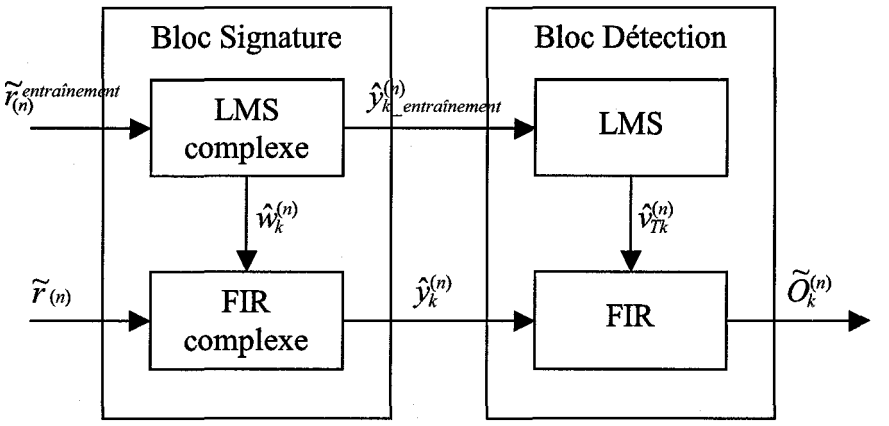


Figure 2.7 Représentation interne des blocs Signature et Détection

2.4 Comparaison des MUDs

Les algorithmes de détection multi-utilisateur présentent des degrés de complexité et de performance différents les uns des autres. Afin de justifier le choix d'un détecteur particulier, on doit évaluer nos besoins en terme de performance et nos limites en terme de complexité. Par conséquent, une étude comparative des MUD a été réalisée dans des travaux au LSSI [DAH02], [DAH04] et nous permet de déterminer les détecteurs suffisamment performants et implantables en temps réel pour répondre aux besoins définis.

2.4.1 Performance

Comme mentionné à la section 2.3, le RAKE est un SUD et n'est donc pas conçu pour éliminer l'interférence d'accès multiple. Comme son traitement consiste seulement à séparer les signaux usager les uns des autres, ces performances se dégradent très rapidement plus le nombre d'usagers est élevé [DAH02]. Aussi, son traitement dépend de la qualité de l'estimer du canal émis par l'estimateur de canal, lequel ne peut reproduire exactement le canal. La capacité du RAKE ne correspond pas aux normes de la 3G.

Le MPIC est un MUD très performant. Il réduit graduellement l'interférence d'accès multiple à partir de ses multiples étages de suppression d'interférence. Plus on y ajoute d'étage, plus il est performant, jusqu'à saturation. Cependant, le MPIC requiert aussi l'estimé du canal, ce qui le rend légèrement moins précis. La capacité du MPIC est l'une des plus élevées [DAH02].

L'algorithme CF-MUD est aussi très performant. Son premier bloc agit comme un SUD, tandis que le second joue le rôle du MUD en utilisant l'information de tous les utilisateurs pour améliorer la qualité du signal restitué de l'utilisateur en cours. L'avantage de cet algorithme sur les deux précédents est qu'il ne nécessite pas d'estimé du canal puisqu'il adapte lui-même ces coefficients. Ceci a pour effet de minimiser les erreurs d'estimation et aussi d'éviter d'inverser la matrice des coefficients du canal pour obtenir ceux du filtre [DAH02].

2.4.2 Complexité

Le tableau 2.1 présente la complexité de différents algorithmes MUD. On remarque que la complexité de calcul du RAKE et du MPIC, généralisé par PIC dans le tableau, est plutôt affectée par celle de l'estimateur de canal, laquelle n'est pas prise en compte dans notre étude. Donc on se concentre seulement sur la complexité des détecteurs. Les paramètres utilisés pour déterminer la complexité sont K, le nombre d'usagers dans une même cellule ; F, le gain de traitement ; et L, le nombre de doigts ou de corrélateur [DAH02]. L'interprétation de ce tableau n'est pas directe et nécessite certains éclaircissements. La complexité du MPIC n'est pas équivalente à celle d'un PIC. Le PIC est seulement un étage d'élimination d'interférence, tandis que notre MPIC en contient trois et qu'en plus il contient l'équivalent de deux RAKE par étage pour estimer l'interférence. Donc, le MPIC aurait une complexité approximative d'ordre O(3FK(2L+1)). Le récepteur adaptatif mentionné correspond seulement à la fonction de détection du bloc Signature du détecteur CF-MUD. Par conséquent, sa complexité serait approximativement de 4 fois celle mentionnée au tableau 2.1. Les résultats du tableau 2.1 tiennent compte seulement de la structure théorique de l'algorithme et non pas des particularités d'une implantation possible. La complexité théorique de CF-MUD se situerait donc entre celle du RAKE et du MPIC, mais plus près de celle du RAKE.

2.5 Conclusion

La majorité des groupes de normalisations des systèmes de troisième génération favorisent l'utilisation du W-CDMA comme technologie d'accès multiple puisque celle-ci permet intrinsèquement d'augmenter la capacité d'un système sans augmenter le nombre de

fréquences nécessaires. Cette technologie distincte chaque signal utilisateur en leur appliquant un code d'étalement de spectre propre à chacun.

Tableau 2.1 Complexité de calcul de détecteurs multi-utilisateurs [DAH02].

Détecteur	**Complexité détecteur**	**Complexité estimateur de canal**
Récepteur conventionnel	O(FK)	$O((FK)^2)$
Rake	O(FKL)	$O((FK)^2)$
MMSE	$O((FK)^3)$	$O((FK)^3)$
PIC	O(FK)	$O((FK)^2)$
SIC	O(FK)	$O((FK)^2)$
Récepteur adaptatif	O(FK)	-

Comme le désétalement de signaux n'est pas un traitement suffisant pour restituer l'information des utilisateurs, on a recourt au MUD pour compléter le traitement de désétalement. La fonction du MUD est, par un moyen quelconque, d'éliminer le MAI pour faciliter la séparation des signaux.

On a vu qu'il existait différents types de MUD de complexité et performance différentes. La connaissance d'un éventail de détecteurs permet de choisir celui qui est le mieux adapté à notre application. Le RAKE est un algorithme simple, mais peu performant, le MPIC est très performant, mais aussi très complexe à implanter en temps réel tandis que CF-MUD, est aussi performant et en plus beaucoup moins complexe que le MPIC. Dans le cas des télécommunications cellulaires, l'algorithme CF-MUD présente les caractéristiques requises pour répondre aux normes de la 3G.

Chapitre 3
Implantation des algorithmes en virgule fixe

Les algorithmes étudiés au chapitre 2 ne représentent qu'une petite partie d'un système cellulaire. Évaluer leur comportement requiert donc une plate-forme de simulation complète incluant des émetteurs et un canal virtuel. La plate-forme de simulation de MUDs a déjà été conçue sur Matlab® dans le cadre d'un autre projet [DAH04]. La section 3.1 présente les caractéristiques du simulateur. L'objectif de ce chapitre consiste à bien comprendre les algorithmes à implémenter sur DSP ainsi qu'à réaliser leur implémentation en virgule fixe.

Ce simulateur exécute les MUDs de manière à leur permettre d'atteindre des performances de traitement idéales, puisque les méthodes utilisées, autant à la simulation de l'émetteur que du récepteur, ne représentent pas parfaitement la réalité et ne considèrent pas toutes les contraintes qu'impose l'implantation physique. Effectivement, Matlab® est un logiciel de simulation admettant le calcul avec des nombres à virgule flottante de 64 bits de résolution, permettant donc d'atteindre une très haute précision de calcul. Dans la réalité, les systèmes doivent accepter une capacité maximale d'utilisateurs, par conséquent, ils doivent effectuer leur traitement le plus rapidement possible. Les unités centrales de traitement à haute résolution sont très complexes à fabriquer, donc très coûteuses, puis celles qui fonctionnent

à virgule flottante ne sont pas aussi performantes en terme de vitesse. Par conséquent, un système réel nécessite un processeur muni d'une architecture conçue pour admettre un haut débit d'exécution d'instructions. Les DSPs 16-bit à virgule fixe possèdent de telles caractéristiques et sont alors destinés au traitement de signal. Le DSP amène deux contraintes supplémentaires, la virgule fixe et une résolution de calcul réduite à 16 bits. Par conséquent, les algorithmes nécessitent d'être réadapté pour satisfaire les contraintes exposées en minimisant la dégradation de leur qualité de traitement. Ce qui nous amène à la section 3.2, laquelle présente la quantification et la mise à l'échelle. Celles-ci représentent respectivement les techniques d'adaptation des signaux d'entrée à la résolution de traitement et d'adaptation des algorithmes au traitement en virgule fixe.

La quantification implique aussi la gestion des signaux d'entrée et des autres paramètres. Bien entendu, les données d'entrée doivent respecter la résolution à virgule fixe, c'est-à-dire être supérieur ou égal à un, puisque les nombres inférieurs à un sont interprétés comme étant zéro, et être inférieur à la capacité maximale admise par un registre de résolution de 16 bits. Les données sont donc interprétées comme des nombres entiers. La section 3.3 traite du processus de mise à l'échelle des données d'entrée et leurs variables correspondantes à l'intérieur du traitement.

3.1 Plate-forme de simulation sur Matlab®

Le simulateur de MUD introduit seulement les composants numériques, lesquels sont suffisants pour évaluer les MUDs. La plate-forme se compose de trois modules principaux, l'émetteur, le canal et le récepteur.

Le rôle de l'émetteur est de transmettre des données vers un canal partagé avec le récepteur. Pour se faire, il doit générer de manière aléatoire des données selon le nombre d'usagers introduit dans le système. Les données de chaque usager doivent être étalées en fonction du facteur d'étalement requis selon le service demandé. Dans ce cas-ci, tous les usagers utilisent le même service. Pour attribuer un code à chacun, l'émetteur se réfère à un fichier contenant une liste de code d'étalement. Une fois étalé, les signaux résultants sont émis vers le canal.

Le canal est similaire pour tous les usagers, mais pas identique. Son rôle est de transformer les signaux en les faisant convoluer avec ses coefficients, d'introduire des versions retardées dans le temps des signaux émis, principe de l'interférence inter-symbole, de les faire interférer entre eux, principe de l'interférence d'accès multiple, en les additionnant pour ne former qu'un signal résultant contenant l'information de tous les usagers, puis de lui additionner le bruit blanc (*AWGN*). Le module du canal génère donc des coefficients de convolution en fonction de la puissance accordée aux coefficients, lesquels représentent des trajets, il génère un vecteur de retards ainsi que le vecteur de bruit blanc en fonction du niveau de bruit désiré dans le canal. Enfin, il effectue le traitement sur les signaux et produit le signal résultant au récepteur.

Le rôle du récepteur est de restituer au mieux l'information de chaque usager en exécutant un algorithme de détection multi-utilisateur sur le signal reçu. Pour ce faire, certains algorithmes dont le MPIC et le RAKE nécessitent la connaissance de leurs coefficients, et des retards associés. Ces paramètres proviennent de l'estimateur du canal, ainsi que d'une

estimation des retards pouvant avoir été introduits, laquelle provient aussi de l'estimateur de canal.

3.2 Quantification - théorie

À la base, la quantification est la transformation d'un signal analogique en signal numérique par l'intermédiaire d'un convertisseur analogique à numérique. [BEL98] la définit ainsi : « La quantification est l'approximation de chaque valeur du signal s(t) par un multiple entier d'une quantité élémentaire q, appelée échelon de quantification. ». Cette opération consiste donc à faire passer le signal analogique dans un composant dont la caractéristique est en marche d'escalier, comme à la Figure 3.1.

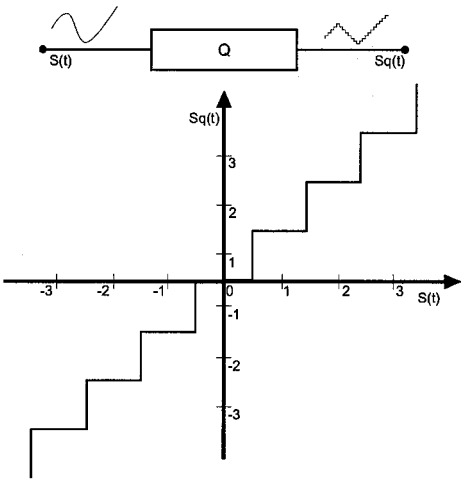


Figure 3.1 Opération de quantification [BEL98] p.48

L'amplitude d'un signal numérique ne peut prendre qu'un nombre fini de valeurs. Il faut donc choisir un ensemble fini de nombres réels qu'on appelle les valeurs représentables. Si le signal n'est pas déjà numérique, il faut le quantifier par l'intermédiaire d'un convertisseur analogique à numérique. Le choix des valeurs représentables et des

opérations arithmétiques est très important puisqu'en forçant le signal à ces valeurs, on introduit une erreur, mais pire encore, les opérations arithmétiques ne transforment pas nécessairement les valeurs représentables en valeurs représentables [KUN91].

3.2.1 Représentation en virgule fixe

Dans ce projet, les valeurs représentables sont représentées en virgule fixe. Pour ce faire, on crée une suite de valeurs (toujours représentables) équidistantes sur l'axe réel. Cette suite peut être située n'importe où sur l'axe réel, mais souvent, elle sera située de manière à être symétrique par rapport à l'origine. Bien sûr, plus le nombre de valeur est élevé, plus la résolution binaire devrait l'être. La dynamique de nos signaux numériques est l'intervalle [–1, 1]. Cet intervalle peut être défini par deux représentations, c'est-à-dire par signe et par module, et par complément à 2 [KUN91].

a) Représentation par signe et par module

Cette représentation consiste à réserver un bit pour le signe et d'écrire le module comme fraction binaire avec les bits restants (Figure 3.2). Elle présente comme inconvénient de ne pas représenter 1 et –1, mais en retour, elle représente deux fois 0 avec –0, qui correspond à 10…0 et +0, qui correspond à 00…0. Par conséquent, la dynamique n'est pas [–1, 1], mais plutôt $[-1 + 2^{-(b-1)}, +1 - 2^{-(b-1)}]$, où b est la le nombre de bit. On peut aussi observer la dynamique sous la forme d'un ensemble de valeurs représentables général :

$\{X_{min}, X_{min} + q, X_{min} + 2q, \ldots, X_{max} - q, X_{max}\}$. $X_{min} = -1 + q$ et $X_{max} = 1 - q$.

Le facteur ou pas de quantification q se calcule ainsi : $q = X_{max} - X_{min} / (2^b - 2)$ [KUN91].

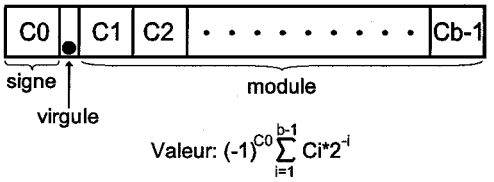


Figure 3.2 Représentation par signe et par module [BEL98] p.280

b) Représentation par complément à 2

Cette méthode est très répandue parce qu'elle présente des avantages pour les opérations arithmétiques. Elle consiste simplement à remplacer le nombre négatif –r par 2 – r et à développer ce dernier comme fraction binaire (Figure 3.2). Ici, 10...0 correspond à –1, lequel est maintenant représentable, et par le fait même, 0 n'est représenté qu'une seule fois. La dynamique devient donc $[-1,\ 1-2^{-(b-1)}]$ et peut aussi s'observer ainsi :

$\{ X_{min}, X_{min}+q, X_{min}+2q, \ldots, X_{max}-q, X_{max} \}$. $X_{min} = -1$ et $X_{max} = 1-q$.

Le pas quantification se calcule ainsi : $q = X_{max} - X_{min} / (2^b - 1)$ [KUN91].

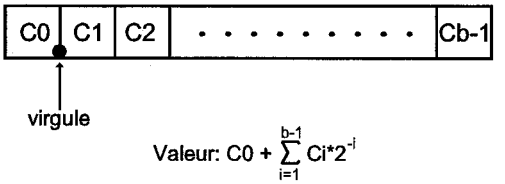


Figure 3.3 Représentation par complément à 2 [BEL98] p.281

3.2.2 Mise à l'échelle

La dynamique des signaux à virgule fixe nous oblige à définir la correspondance entre les valeurs représentables et les mots binaires par l'intermédiaire des paramètres X_{max} et X_{min}. Ceci implique donc un facteur d'échelle, lequel nécessite souvent d'être modifié à l'intérieur même d'une application, dont un filtre numérique.

Une fois la représentation des nombres choisie, on doit s'assurer que les valeurs du signal soient représentables. Pour ce, il faut les modifier jusqu'à ce qu'elle soit représentable, ce qui s'appelle la quantification du signal, lorsque le signal est analogique au départ, ou la mise à l'échelle de ses valeurs, lorsque le signal est déjà numérisé. Dans un filtre numérique, il faut quantifier ou mettre à l'échelle les valeurs d'un signal à l'entrée du filtre ainsi qu'à l'intérieur de celui-ci. En effet, à l'intérieur du filtre, les opérations d'addition et de multiplication de deux valeurs représentables ne produisent pas nécessairement une autre valeur représentable, selon la résolution de calcul désirée. Par conséquent, le résultat de telles opérations effectuées sur 16 bits peut produire un résultat s'étendant au-delà de 16 bits, jusqu'à 32 bits. Il peut être possible, à l'intérieur du filtre, de travailler avec plusieurs résolutions, mais souvent, pour rester optimal en terme de vitesse de traitement, il faut requantifier les résultats pour qu'il demeure à leur résolution originale, laquelle devrait avoir été choisie pour ses performances à l'intérieure de l'architecture matérielle utilisée. Cette mise à l'échelle ou requantification à l'intérieur du filtre revient à une augmentation du pas de quantification et/ou d'une réduction de la dynamique des valeurs [KUN91].

Cette gestion de la quantification est très importante car elle assure les valeurs de rester dans le domaine représentable. En effet, le résultat d'une opération arithmétique se situant à l'extérieur de l'intervalle des valeurs représentables, qu'on appelle dépassement, peut engendrer des erreurs de traitement qui altèrent définitivement la restitution des signaux. En effet, un résultant à l'extérieur de la plage représentable sera considéré par le processeur comme valide, mais cette valeur contenue dans le registre ne correspondra pas à ce qu'elle devait correspondre et engendrera une erreur de quantification que le filtre traînera avec lui

jusqu'à ce que cette valeur n'est plus d'impact dans les opérations, ce qui peut devenir parfois catastrophique.

3.3 Quantification du signal et mise à l'échelle des variables

Après avoir étudié les bases de la quantification, nous avons les connaissances suffisantes pour modifier la plate-forme Matlab® et les MUDs pour les faire fonctionner en virgule fixe. Comme le DSP TMS320C6416 fonctionne à virgule fixe, que ses instructions sont optimisées à une résolution de 16 bits et que sa représentation des nombres est par complément à 2, notre représentation des nombres sera la même ainsi que la résolution de travail.

La résolution sur 16 bits de nombres signées représentés par complément à 2 correspond à une résolution du module de 15 bits, ce qui offre une dynamique de $[-2^{15}, 2^{15}-1]$ ou [–32768, 32767], où X_{max} correspond à 1-q ou 32767 et X_{min} correspond à -1 ou 32768. Cette dynamique ne tient pas compte de l'architecture du DSP par rapport à l'ensemble d'instructions qu'il offre. Ce qui signifie qu'une telle dynamique ne laisse pas une grande marge de manœuvre par rapport au dépassement, mais permet une plus grande précision. Dans notre cas, la dynamique de notre signal doit être réduite pour tenir compte du jeu d'instructions du DSP qui nous permet d'effectuer l'opération multiplie-accumule *MAC* sans permettre une mise à l'échelle entre la multiplication et l'addition. Cette instruction range le résultat sur 32 bits. Aussi, pour augmenter la vitesse de traitement, le jeu d'instruction permet de charger jusqu'à 8 valeurs 16-bit simultanément, rendant avantageux de réduire le nombre de mises à l'échelle à l'intérieur du filtre, mais obligeant de réduire aussi la dynamique. Comme le traitement des MUDs nécessitent beaucoup d'opérations du

genre $\sum(A \bullet B)$, et qu'il s'effectue avec des nombres complexes, il est possible de charger 4 réels (R_x) et 4 imaginaires (I_x), d'effectuer les huit multiplies-accumules avec les coefficients (CR_x et CI_x) des filtres; $PR_{12} = R_1 \bullet CR_1 + R_2 \bullet CR_2$ et $PI_{12} = I_1 \bullet CI_1 + I_2 \bullet CI_2$, puis d'additionner les 4 résultats ensemble; $PR = PR_{12} + PR_{34}$ et $PI = PI_{12} + PI_{34}$, puis d'additionner les deux résultats résiduels; $P = PR + PI$ pour enfin pouvoir mettre ce résultat à l'échelle. Par conséquent, le résultat provient d'une multiplication suivie de trois additions consécutives, sans ajustement de quantification. Il faut donc s'assurer que la dynamique du signal ne soit pas trop grande pour éviter tout dépassement. Comme les résultats de la multiplication sont rangés sur 32 bits, il est plus efficace d'effectuer les additions sur 32 bits. Donc avant mise à l'échelle, la résolution maximale permise est de 31 bits, c'est-à-dire 32 moins le bit de signe. Les additions étant exécutée en dernier, on enlève 1 bit de résolution pour chaque addition exécutée ce qui protège la dynamique résultante contre les dépassements provenant du cas extrême d'une addition de deux valeurs occupant le nombre de bits maximal. Par exemple, pour obtenir une somme signé sur 4 bits, on additionne $-2^3-2^3 = -2^4$. Il reste donc 28 bits, puis on divise ce nombre par deux pour tenir compte de l'effet de la multiplication de deux valeurs, par exemple, $(-2^3)\bullet(-2^3)= 2^6$, lequel occupe 7 bits. La dynamique maximale que nous pouvons utiliser pour quantifier notre signal est $[-2^{14}, 2^{14}-1]$ ou [-16384, 16383].

Le calcul effectué précédemment s'applique à un service haut débit d'OVSF 4, lequel traite seulement quatre échantillons du signal étalé pour restituer un échantillon du signal informationnel. Dans le cas du traitement de la voix, l'OVSF est 64, il faut donc traiter 64 échantillons du signal étalé pour en restituer un. Par conséquent, il faut répéter 16 fois plus

de MACs qu'il en fallait pour l'OVSF 4, et il faut les additionner ensemble selon $S = P_1 + P_2 + \ldots + P_{16}$. Ce qui ajoute 16 additions de plus et nécessite une réduction de la dynamique. Cette façon de faire est trop drastique puisque tous les échantillons ne sont pas d'amplitude maximale. Par conséquent, on ne peut pas suivre cette méthode, puisque la dynamique sera trop faible pour permettre un traitement adéquat.

Par conséquent, il est préférable de déterminer le meilleur facteur de mise à l'échelle par essaie-erreur sur la plate-forme virgule fixe en comparant les résultats de traitement avec ceux de la plate-forme virgule flottante. Lorsque les résultats sur un grand nombre d'évaluation sont sensiblement les mêmes, c'est que le facteur est adéquat, dans notre cas, le facteur de mise à l'échelle optimal, était de 2^{13}. Bien entendu, cela dépend du nombre d'usagers et du facteur d'étalement.

Sur la plate-forme Matlab®, le signal d'entrée est déjà numérique. Afin qu'il corresponde à la dynamique calculé, le signal est analysé pour déterminer une moyenne de son amplitude maximale absolue, laquelle est utilisée comme diviseur pour ajuster les échantillons du signal dans la dynamique [-1, 1[, puis celle-ci est quantifiée en multipliant les nouveaux échantillons par un facteur de mise à l'échelle. Le signal est maintenant prêt à être filtré.

Lors du filtrage du signal, après un certain nombre d'opérations, les registres de calcul risque de déborder, donc il faut réduire la dynamique du signal en temps d'exécution en effectuant une division par le facteur de mise à l'échelle ou par un décalage par la droite équivalent à la division précédente. Cette opération permet d'éviter tout dépassement qui pourrait altérer le comportement du filtre.

3.4 Conclusion

La simulation des algorithmes est effectuée sur une plate-forme Matlab® [DAH04]. Cette plate-forme produit des résultats idéaux puisqu'elle est basée sur une architecture à virgule flottante munie d'une résolution de 64 bits. Ce programme simule l'émetteur en générant les données usager et en les étalant pour les faire passer au canal. Il simule aussi le canal en déformant les signaux émis, en introduisant des versions des signaux retardées dans le temps, en les bruitant et en les agglomérant ensemble pour générer l'interférence d'accès multiple. Pour finir, il simule le récepteur en exécutant un MUD.

Dans la réalité, on n'utilise une architecture à virgule fixe et à résolution de 16 bits. Pour ne pas corrompre les données originales, il faut les quantifier. La quantification est une transformation d'un signal analogique en un signal numérique de valeurs représentables. Les valeurs représentables, en virgule fixe, peuvent être représentées par signe et par module, ou par complément à 2. La dernière représentation est choisie par rapport à la première puisqu'elle définit la valeur 0 seulement qu'une fois et qu'elle amène des avantages sur les opérations arithmétiques. Les données quantifiées ou déjà numériques doivent souvent être mise à l'échelle pour demeurer dans la dynamique des valeurs représentables choisies au départ.

La quantification et la mise à l'échelle des signaux et variables du système requièrent une étude approfondie pour s'assurer de ne pas altérer les résultats de traitement de filtrage. Par conséquent, avec une résolution de calcul initiale de 16 bits, d'un rangement des résultats temporaires sur 32 bits, d'une représentation des nombres signée en complément à 2 puis d'une dynamique s'étalant de -1 inclus jusqu'à 1 exclus, on évalue que le facteur de mise à

l'échelle serait optimal à 2^{13}, ce qui éviterait des dépassements lors des opérations de calcul puis permettrait une résolution de calcul suffisante.

La plate-forme de simulation a été testée en point fixe avec un facteur de mise à l'échelle de 2^{13} et avec d'autres facteurs se situant autour du précédent. La comparaison des résultats de traitement en virgule fixe avec ce facteur de mise à l'échelle a démontré que la résolution était suffisante et que les résultats était sensiblement les mêmes que sur la plate-forme à virgule flottante.

Chapitre 4
Implémentation des méthodes MUD sur DSP

L'implémentation des algorithmes en virgule fixe, effectuée au chapitre 3, évoque une étape de l'adaptation des méthodes aux contraintes qu'impose l'équipement destiné à les exécuter. Cette étape exige la connaissance préalable des caractéristiques de l'unité de traitement (*UT*) qui se rapportent à sa résolution de calcul. De la même manière, le choix de l'UT requiert la connaissance des contraintes inhérentes au système à implanter.

La détection multi-utilisateur, selon les normes de la 3G [SAN01], appartient aux systèmes temps réel et implique un traitement extrêmement rapide afin d'accommoder le plus grand nombre d'utilisateurs avec un minimum d'infrastructures. Par conséquent, on doit implanter les récepteurs dans des équipements adaptés aux contraintes précédentes, tel un DSP. Il existe différents fabricants et familles de DSP, chacune arborant une architecture particulière. Il convient donc de choisir le processeur possédant les ressources nécessaires afin de répondre à des besoins spécifiques. La section 4.1 présente la structure du TMS320C6416.

Afin d'exploiter au maximum les ressourcent du DSP, le code des algorithmes nécessite deux autres étapes d'adaptation. La première assure la transformation du code initial de Matlab® dans un langage compréhensible par le compilateur du DSP. La seconde transforme le code en langage machine optimisé en fonction de la structure du DSP. La section 4.2 élucide donc les étapes de programmation et d'optimisation.

Le DSP sert uniquement de MUD. Par conséquent, Matlab® reste le simulateur de l'émetteur, du canal et du récepteur radiofréquence. Afin de permettre au DSP de communiquer avec Matlab®, ce dernier génère des fichiers de données compatibles au compilateur et que le DSP peut utiliser. La section 4.3 traite de la génération des fichiers de données.

Ces fichiers de données permettent de valider le traitement effectué par le DPS avec le code optimisé des algorithmes. Ces performances de traitement sont alors comparées à celle de Matlab® selon les mêmes échantillons de données. Une fois validé, on évalue la complexité du code. La validation et l'évaluation sont discutées à la section 4.4.

4.1 Structure du DSP TMS320C6416

Les DSPs TMS320C64x (*C64x*) obtiennent les plus hautes performances des DSPs à points fixes de la plate forme TMS320C6000. Le C64x est basé sur la seconde génération de l'architecture haute performance VelociTI (*VEry LOng Code Instruction of TI*), VelociTI.2, présentée à la Figure 4.1 [TI03].

Le C64x atteint une performance jusqu'à 8000 millions d'instructions par seconde (*MIPS*) à une fréquence d'horloge de 1000 MHz. Son noyau contient 64 registres à usages généraux

de 32 bits et huit unités fonctionnelles hautement indépendantes dont deux multiplieurs pour un résultat de 32 bits et six unités arithmétiques et logiques. Ces unités incluent les extensions VelociTI.2. Ces extensions renferment de nouvelles instructions pour accélérer les performances et déployer le parallélisme de l'architecture VelociTI. Le C64x peut produire quatre multiplieurs-accumulateurs (*MAC*) de 16 bits par cycles pour un total de 4000 millions de MAC par seconde, ou huit MACs de 8 bits pour un total de 8000 MMACs [TI03].

Le C64x contient d'autres périphériques non utilisés pour ce projet, mais qui méritent tout de même d'être mentionnés. Il possède deux coprocesseurs embarqués hautes performances pour le décodage; *Viterbi Decoder Coprocessor* (*VCP*) et *Turbo Decoder Coprocessor* (*TCP*), un contrôleur évolué d'accès directe à la mémoire (*EDMA*), un contrôleur d'accès au média Ethernet (*EMAC*), un module de gestion des entrées/sorties des données (*MDIO*), une interface de mémoire externe (*EMIF*), des entrées/sorties à usage général (*GPIO*), une interface de port hôte (*HPI*), un port série audio multivoie (*McASP*), un port série bufferisé multivoie (*McBSP*), un connecteur de composant périphérique (*PCI*), un contrôleur de boucle à verrouillage de phase (*PLL*), un contrôleur d'accès direct à la mémoire (*DMA*), un registre d'horloge et quelques autres périphériques.

4.1.1 Architecture

L'architecture interne du C64x dénombre deux ensembles hautement indépendants d'unité fonctionnelle (*1* et *2*) et deux fichiers de registre (*A* et *B*). Chaque fichier contient 32 registres à usages généraux, et chaque ensemble contient quatre unités fonctionnelles indépendantes; *L*, *S*, *M* et *D*. Ces dernières communiquent directement avec leur fichier de

registre associé, mais ne peuvent pas communiquer directement avec celui du côté opposé. La communication directe s'effectue par l'intermédiaire des chemins de donnée correspondants à un côté, tandis que l'indirecte se fait par l'intermédiaire des chemins croisés *1X* et *2X*. La communication entre les unités fonctionnelles *D* et la mémoire se fait par l'intermédiaire des chemins d'adresse *DA1* et *DA2*, tandis que celle entre la mémoire et les registres se fait par l'intermédiaire des chemins de données « charge de la mémoire » *LD1*, *LD2* et des chemins de données « range en mémoire » *ST1* et *ST2*.

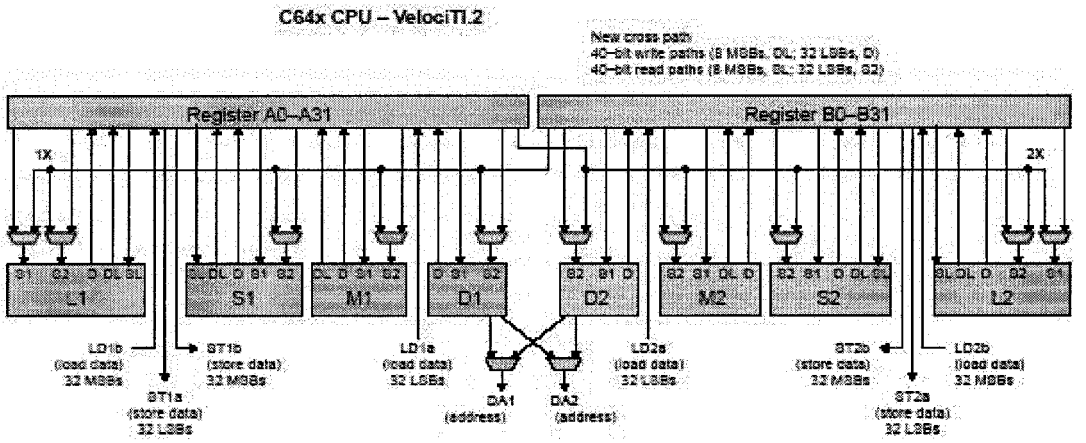


Figure 4.1 VelociTI.2 UCT du C64x [TI03]

a) Unités fonctionnelles

Chaque paire d'unité fonctionnelle performe les mêmes opérations. L'unité *M* se dédie particulièrement à la multiplication, *L* à l'arithmétique, la logique et la comparaison, *S* à l'arithmétique, la logique, la comparaison, au décalage et au branchement, et *D* à l'arithmétique et à l'accès mémoire. Ces ensembles permettent d'atteindre un degré élevé de parallélisme en exécutant jusqu'à huit instructions en un seul cycle. Chaque ensemble lit et écrit directement dans le fichier de registre de son propre chemin de donnée, donc l'ensemble *1* s'associe au fichier *A*, tandis que l'ensemble *2* s'associe au fichier *B* [TI01].

b) Chemins de croisement

Les huit unités fonctionnelles ont accès aux fichiers de registre du côté opposé via les chemins de croisement *1X* et *2X* de 40 bits de largeur. Le chemin *1X* octroie aux unités fonctionnelles du chemin de données *A* de lire leur source d'un registre du fichier *B*, de même avec le chemin *2X*, les unités du chemin *B* peuvent lire dans le fichier *A*. Comme il y a seulement un chemin de croisement pour chaque ensemble, une seule unité par ensemble peut lire une source du fichier de registre opposé par cycle. Le pipeline du C64x permet à deux unités par côté de lire la même source du chemin de croisement simultanément [TI01].

c) Chemins d'adresse

Les chemins d'adresse de données *DA1* et *DA2* sont connectée aux unités correspondantes *D*. L'unité D d'un ensemble peut se connecter à son chemin d'adresse correspondant ou se connecter au chemin de l'autre ensemble lors du chargement ou du rangement de donnée. Ceci implique l'utilisation d'un registre d'adresse appartenant au fichier associé au chemin d'adresse employé. Cependant, les accès mémoire exécutés en parallèle doivent respecter leur chemin de données respectif, ou tous les deux utiliser un chemin de croisement vers le fichier de registre opposé [TI01].

d) Chemins de données

Les chemins de données existent en deux catégories, ceux qui communiquent entre les registres et les unités fonctionnelles et ceux qui communiquent entre les registres et la mémoire. Tous ces chemins supportent le transport de dimension 40 bits. Les chemins *LD1*, *LD2*, *ST1* et *ST2* permettent des accès mémoire de 32 bits chacun. Une même unité *D* peut

utiliser deux chemins de même type pour charger ou ranger un mot de 64 bits en un seul cycle [TI01].

4.1.2 Mémoire

Le C64x utilise une architecture cache à deux niveaux de mémoire. Le premier niveau contient deux sections de cache, une dédiée au programme (*L1P*), et l'autre à la donnée (*L1D*). La mémoire de programme et celle de donnée se partagent toutes deux le second niveau, désigné par *L2*. *Ce dernier* est configurable, permettant ainsi différentes quantités de cache et de *SRAM* (*Single Random Access Memory*) [TI04].

L1P possède 16 kilooctets (*ko*) de cache à accès direct (*direct-mapped*) et *L1D* possède 16 ko de cache associatif d'ensemble à deux chemins (*2-way set associative)*. Le niveau 2 de SRAM/cache consiste en un mégaoctet (*mo*) d'espace mémoire partagée entre la section de programme et de donnée. *L2* peut opérer comme SRAM, cache ou les deux à la fois, dépendamment de son mode. La dimension totale peut être configurée en SRAM, tandis qu'un maximum de 256 ko peut être configuré en cache, laquelle est un *4-way set* associative. Les deux types de mémoire peuvent être utilisés en même temps [TI01] [TI04].

4.2 Principes d'optimisation du code du DSP

L'optimisation du code s'accomplit par l'utilisation du pipeline, lequel décroît énormément la complexité d'un algorithme. Pour utiliser le pipeline du C64x, il faut d'abord connaître les bases d'optimisation qui y sont relatives, telles le remplissage des cases de retard (*delay slots*) par des instructions utiles, le parallélisme, le déroulement des boucles et l'optimisation de la largeur des mots [GAB03].

4.2.1 Cases de retard

Les cases de retard sont inhérentes à plusieurs instructions assembleurs. Elles sont représentées par l'instruction *NOP* (*No OPeration*). On peut remplir ces cases en les remplaçant simplement par des instructions utiles, ceci se faisant par un meilleur agencement des instructions dans la fonction ou la boucle. En fait, les cases de retard équivalent à une forme de latence d'exécution ou de résultat. Toutes les instructions communes aux composants C6000 ont une latence d'unité fonctionnelle de un. Ceci signifie qu'une nouvelle instruction peut être entreprise sur l'unité fonctionnelle à chaque cycle [TI00]. Donc en plus de la latence d'unité fonctionnelle d'un cycle, une instruction peut nécessiter un certain nombre de cycle de retard avant que l'on puisse avoir le résultat. Exemple, l'instruction *load* demande 1 cycle d'instruction, mais requière en plus 4 cycles de retard avant de rendre la valeur chargée disponible, pour un total de 5 cycles.

4.2.2 Parallélisme

Le parallélisme se fait par l'utilisation de différentes unités fonctionnelles de manière simultanée. Comme chaque unité est une entité matérielle, il s'avère donc impossible d'utiliser deux fois la même unité dans une séquence d'instructions parallèles, même si l'instruction et les registres utilisés sont différents. Par conséquent, il est possible d'utiliser un maximum de huit instructions en parallèle. Il faut aussi respecter les contraintes relatives aux unités fonctionnelles, aux chemins de croisement et aux chemins d'adresse [TI00].

4.2.3 Optimisation de la largeur des mots de chargement et rangement

L'optimisation de la largeur des mots consiste à remplir les registres de chargement et de rangement d'information utile afin d'occuper toute leur largeur de 32 bits. C'est-à-dire,

même si on travaille avec des entiers de 16 bits, on ne charge pas qu'un seul entier dans la partie basse du registre 32-bit et des zéros dans la partie haute, on charge plutôt deux entiers de 16 bits, un dans la partie basse et l'autre dans la haute. Le C64x contient des instructions qui permettent d'effectuer des opérations sur les deux entiers en même temps dans le même registre. Il est aussi possible d'optimiser davantage en effectuant un chargement ou un rangement d'un mot de 64 bits à l'aide de deux registres de même fichier. En effet, une même unité *D* peut utiliser deux chemins de même type (sur chaque ensemble et fichier) pour charger ou ranger un mot de 64 bits en un seul cycle. Il ne reste plus qu'à dérouler la boucle suffisamment pour traiter autant de données qu'on en charge ou en range. [TI02].

4.2.4 Déroulement des boucles

Le déroulement des boucles consistes à répéter les instructions internes à la boucle afin d'effectuer moins de tours de boucle. Ceci à pour conséquences de réduire le nombre de vérifications de fin de boucle, d'augmenter l'utilisation du parallélisme et l'optimisation de la largeur des mots, mais aussi, d'augmenter considérablement la dimension du code [TI00].

4.2.5 Fusion de deux niveaux de boucle

La fusion de deux niveaux de boucle est l'intégration de la boucle primaire à la boucle secondaire. Ceci s'effectue en ajoutant des conditions d'exécution aux instructions de la boucle primaire, lesquelles ne sont pas exécutées au même rythme que celles de la boucle primaire. Cette optimisation a pour effet d'augmenter le degré de parallélisme et aussi d'éviter l'exécution d'une boucle, laquelle peut nuire à la performance du pipeline. En

effet, un pipeline se construit uniquement dans la boucle secondaire, si cette dernière ne comporte pas suffisamment d'itération, le pipeline se charge et se décharge trop rapidement, dégradant sa performance. En ayant deux boucles en une, on élimine la boucle primaire et permet de créer un pipeline qui s'exécute plus longuement et réduit en même temps la complexité des deux boucles. La fusion se concrétise seulement lorsque la boucle primaire n'est pas trop lourde en instructions.

4.2.6 Pipeline

Maintenant que les cinq procédures d'optimisation de base ont été traitées, le pipeline peut être abordé. Le pipeline logiciel d'un C6000 consiste en l'optimisation de l'utilisation des unités fonctionnelles disponibles en tenant compte des procédures de base d'optimisation. La génération du pipeline s'effectue en répartissant les instructions sur toutes les unités fonctionnelles possibles afin de pouvoir créer un flot d'exécution dense et constant. Ce flot devient lisiblement incohérent, mais reste fonctionnel si la synchronisation des instructions et des registres demeure non perturbée. Le flot d'instruction pipelinée comporte une section d'instructions qui se répètent indéfiniment, cette section est le noyau, lequel sera mis en boucle. La section précédente au noyau est le prologue. Celui-ci comporte le flot d'instruction préliminaire au noyau, ce flot amène les instructions une par une dans l'ordre logique pour préparer les données à passer au noyau. La section suivante à la boucle pipelinée est l'épilogue, lequel effectue le travail opposé au prologue, c'est-à-dire qu'il décompose peu à peu le noyau pour traiter les données résiduelles qu'on ne pouvait pas gérer dans la boucle. Le pipeline se trouve complexe à générer mais extrêmement performant en vitesse d'exécution [TI00].

4.3 Programmation et optimisation des algorithmes

Le logiciel d'exploitation du C6416 s'appelle *Code Composer Studio* (*CCS*). Son compilateur permet de traduire le code C, l'assembleur linéaire et l'assembleur en code machine. Le code Matlab® n'est donc pas compatible, il faut le traduire manuellement dans une des trois formes mentionnées. Le langage idéal à utiliser se détermine par l'étude des besoins préalablement identifiés et par la rencontre des normes établies pour le système. Les trois langages présentent, à des degrés différents, un niveau de difficulté de programmation versus un niveau d'exploitation des ressources matérielles. On entend par niveau d'exploitation, l'imposition des unités fonctionnelles, des registres, des chemins ainsi que des emplacements mémoire [TI02].

4.3.1 Langages de programmation

En somme, le langage C est général et standard, il s'applique à une multitude de processeurs d'architecture différente. Il se situe parmi les langages de haut niveau, il ne demande donc pas au programmeur la connaissance préalable de l'architecture visée, ce qui permet une programmation simplifiée, réutilisable et plus versatile, mais ne permet pas d'imposer le choix des ressources offertes par l'unité centrale de traitement (*UCT*). Le compilateur peut optimiser le code C et produire un pipeline [TI02].

L'assembleur linéaire s'applique directement à un processeur ou une famille donnée. Il nécessite donc la connaissance des instructions, de certaines directives dédiées à l'architecture choisie, et parfois des adresses mémoires des données. Un programme en assembleur linéaire se compose d'une suite d'instructions lesquelles n'imposent pas les unités matérielles à utiliser. La facilité à le programmer et le degré de contrôle atteint sur la

machine se situent entre ceux du C et de l'assembleur. Le compilateur peut l'optimiser et y définir un pipeline [TI02].

L'assembleur nécessite une connaissance approfondie de l'architecture utilisée. Il nécessite donc la connaissance des instructions, des directives nécessaires, des unités fonctionnelles, des chemins de croisement, des chemins de données, des chemins d'adresse et des registres. Il demande donc une gestion supervisée des ressources matérielles afin d'être efficace et d'exploiter le parallélisme et les unités fonctionnelles afin d'obtenir un code hautement efficient. Par conséquent, l'utiliser est fastidieux, mais permet la liberté totale de l'exploitation des ressources. Le compilateur ne peut pas modifier le code assembleur [TI02].

4.3.2 Implantation en C

On implante premièrement les algorithmes avec le langage C vu sa simplicité de codage et sa ressemblance structurelle avec Matlab®. La plate-forme de programmation C peut être CCS ou Visual Studio. Les objectifs de cette étape sont de s'assurer que l'algorithme s'exécute normalement, c'est-à-dire qu'il fournisse les mêmes performances que sous Matlab®, et d'obtenir une première évaluation de sa complexité.

4.3.3 Optimisation du code C

Le résultat de la compilation commune du code C fournit un code machine linéaire hautement complexe, c'est-à-dire très lent à exécuter, puisqu'il reproduit exactement la structure du code C en assembleur. Heureusement, le compilateur de CCS inclut un optimisateur très performant pouvant produire un code assembleur très rapide en générant

des pipelines aux boucles de plus bas niveau. Un pipeline ainsi généré peut-être hautement performant dépendamment de la complexité de la boucle et du nombre d'itérations que celle-ci doit subir. Plus l'outil d'optimisation reçoit d'information à propos de la boucle, plus il peut générer un pipeline rapide. Hélas, l'outil d'optimisation n'est pas en mesure de fusionner deux boucles, ce qui oblige alors le concepteur de créer son propre pipeline.

Le compilateur invoque l'outil d'optimisation lorsque le programmeur sélectionne les options appropriées. CCS fournit une interface simple et visuel pour les saisir. Les options principales de compilation et d'optimisation sont en vue d'un compromis entre vitesse d'exécution et dimension du code, la vitesse étant primordiale dans ce cas, (*no –ms*) s'impose. Ensuite on retrouve les niveaux d'optimisation, c'est-à-dire à quel niveau du code se situe l'optimisation recherchée, soit registre, locale, fonction ou fichier, la dernière (*-03*) étant la plus performante ici. D'autres options existent pour accélérer le code, telles que « pas de mauvais code d'alias » (*-mt*) et « spéculer sur le seuil » (*-mh*). Lorsque l'application nécessite beaucoup de mémoire, il faut configurer le DSP à utiliser plusieurs page de mémoire à l'aide de (*-ml0*).

4.4 Validation et évaluation

Les détecteurs multi-utilisateurs étudiés ont tous été préalablement validés sur la plate-forme Matlab® de simulation des détecteurs. Par conséquent, tous ces algorithmes sont théoriquement fonctionnels et produisent des résultats variants de satisfaisants à excellents. Afin de valider les méthodes, la plate-forme de simulation contient des modules de génération des signaux, du canal, du bruit, des codes utilisateurs et de simulation de

l'émetteur et du canal. Les données générées par ces modules peuvent être réutilisées par le DSP pour y valider les MUDs.

4.4.1 Génération des données

La génération des données utilisateurs se fait aléatoirement sur des bits signés de 1 ou –1 à l'aide du générateur de nombre aléatoire de Matlab®. On génère le canal à trajets multiples aussi à l'aide du générateur de nombre aléatoire. La génération tient donc compte de la puissance attribuée aux différents trajets et du nombre de trajets recensés. La signature (code utilisateur) se génère à partir de la valeur des paramètres représentant le nombre d'usagers connectés et le facteur d'étalement utilisé (OVSF). Le bruit AWGN (*Additive White Gaussian Noise*) est généré aléatoirement en tenant compte du rapport énergie binaire sur énergie du bruit (*Eb/No*) et de la longueur de la signature de l'utilisateur.

La génération des signaux reçus au récepteur se fait par la simulation de l'émetteur et du canal. L'émetteur se simule simplement en codant les données de chaque utilisateur selon leur signature respective. Le signal codé de chaque utilisateur se dirige vers le simulateur du canal, lequel leur effectue une convolution avec les coefficients des divers trajets, et cela en tenant compte des retards inhérents aux trajets générés. Le signal utilisateur devient donc une superposition des signaux modifiés et retardés par les trajets. Le signal reçu au récepteur représente donc une superposition des signaux de tous les utilisateurs, auquel on ajoute le bruit.

Afin de valider la performance des MUDs sur DSP, ce dernier requiert les paramètres d'entrée et les entrées associés à des résultats de simulations exécutées sur Matlab®. Par

conséquent, les paramètres, le signal d'entrée et les coefficients des filtres (si applicable) générés par simulation sur Matlab® sont enregistrés automatiquement dans des fichiers de format directement utilisables par le DSP.

4.4.2 Validation

Les programmes Matlab® servent de référence au codage des programmes implantés dans le DSP. Une première version de code fonctionnant avec des nombres à point fixe de 32 bits de résolution est produite en C. Visual Studio sert de plate-forme de validation de ces programmes. Lorsque les performances atteignent sensiblement les mêmes qu'avec Matlab®, les programmes sont modifiés pour fonctionner avec une résolution de 16 bits. Cette dernière permet d'atteindre un traitement optimal en terme de vitesse de traitement avec le C6416, dont l'architecture interne est conçue pour travailler de manière optimale en 16 bits.

Les programmes à virgule fixe de résolution 16 bits deviennent aussi des programmes de référence lorsque leur fonctionnement reflète celui de Matlab®. Visual Studio sert donc de plate-forme de référence pour valider les performances sur DSP. Les mêmes programmes C sont intégrés sur DSP par l'intermédiaire de CCS, optimisés à l'aide de l'outil d'optimisation puis validés. Ils peuvent aussi être codés manuellement en assembleur, puis validés à nouveau.

Les figures 4.2 à 4.4 montrent les résultats de l'algorithme CF-MUD implémenté dans le DSP de TI à l'aide du logiciel Code Composer Studio. La simulation est réalisée pour des données de cinq usagés. La figure 4.2 affiche le signal reçu et appliqué à l'entrée du MUD,

la composante réelle est dans le graphique du haut, tandis que la composante imaginaire est dans le graphique du bas. La figure 4.3 présente le signal obtenu en sortie du MUD des données estimées de l'usager 1, le graphique du haut présente le signal du MUD avant l'étage de décision (décision douce), tandis que celui du bas le présente après l'étage de décision (décision dure). Finalement, la figure 4.4 compare le signal original (transmis) de l'usager 1 avec le signal estimé en sortie du MUD du même usager. Nous observons l'exactitude de fonctionnement du CF-MUD dans le DSP de TI.

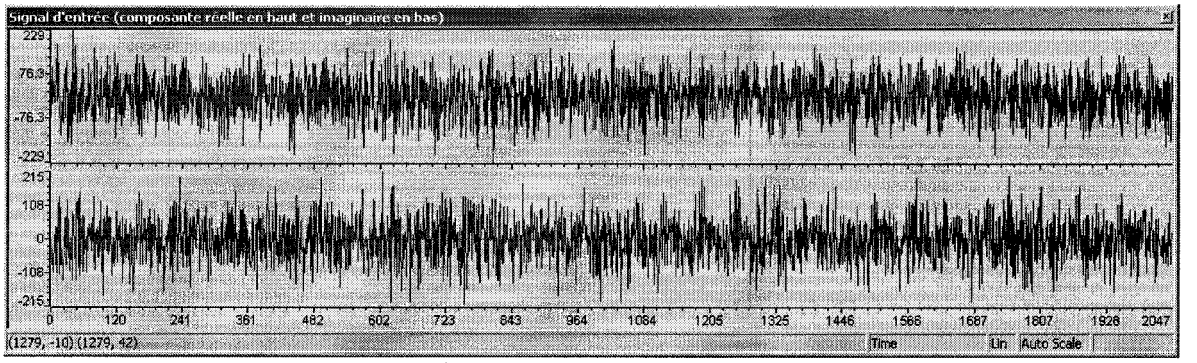


Figure 4.2 Signal d'entrée au MUD

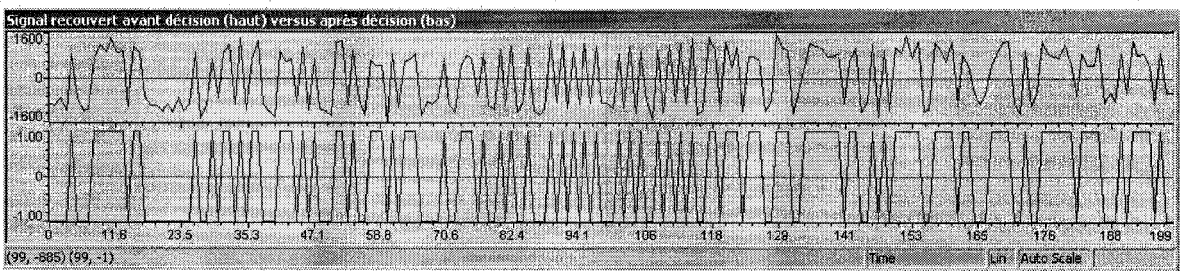


Figure 4.3 Signal recouvert par le MUD

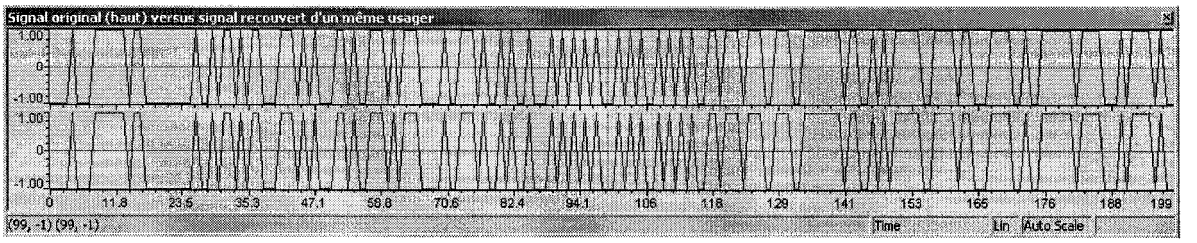


Figure 4.4 Signal original versus signal recouvert

4.4.3 Évaluation

Lorsque les programmes produisent les résultats escomptés, on évalue leur complexité de traitement à l'aide de CCS, lequel inclut quelques méthodes de comptage du nombre de cycles exécutés par le DSP. La première méthode de comptage réside en l'utilisation de l'interface de l'horloge, laquelle démarre et arrête en même temps que l'exécution du programme. L'horloge peut être remise à zéro en tout temps si le programme n'est pas en exécution. La seconde méthode, le *Profiler*, utilise l'horloge comme unité de comptage. Le *Profiler* fournit une interface plus complète permettant d'analyser plusieurs fonctions, boucles ou sections de programme de manière distinctive lors de l'exécution du programme principal. Le *Profiler* permet donc de déterminer les parties plus complexes d'un MUD. La troisième méthode réside en l'utilisation d'un *timer*. Il n'y a donc pas d'interface visuelle et il faut y ajouter du code. Cette méthode est utile pour valider l'évaluation des deux premières. Les méthodes de l'horloge ne sont pas valables lorsque le code est intégré directement sur le DSP, il faut donc l'intégrer dans le simulateur appelé « *Device Cycle Accurate Simulator* ». La complexité des programmes codés et optimisés directement en assembleur peut être évaluée directement sur papier en calculant le nombre de cycles au prologue, au noyau du pipeline et à l'épilogue. Il est toujours mieux de valider la première évaluation à l'aide d'une seconde méthode afin de s'assurer que les résultats convergent.

4.5 Conclusion

L'implantation d'un algorithme sur DSP n'est pas une tâche facile. Contrairement à la programmation conventionnelle, programmer de manière optimale un DSP nécessite une connaissance approfondie de son architecture matérielle. La maîtrise de sa structure devient

donc primordiale afin de bénéficier des caractéristiques avantageuses qu'elles procurent. On retrouve parmi ces caractéristiques huit unités fonctionnelles, 64 registres à usages généraux, différents types de chemins de donnée et d'adresse et d'autres périphériques.

L'intégration optimale d'un algorithme dans la structure d'un DSP nécessite aussi la maîtrise des principes d'optimisations. Ces principes, remplissage des cases de retard, parallélisme, optimisation de la largeur des mots, déroulement des boucles et fusion des boucles permettent d'exploiter au maximum les ressources du DSP. Ces principes se concrétisent par la génération d'un pipeline, c'est-à-dire une disposition optimale des instructions pour permettre l'utilisation maximale des ressources en tout temps.

Code Composer Studio alloue de la flexibilité au programmeur relativement au choix du langage qu'il veut utiliser. Chaque projet n'expose pas les mêmes contraintes, alors dépendamment de celles-ci, quelques langages peuvent être favorable à l'atteinte des spécifications requises. Le langage C permet un code mieux structuré, réutilisable et plus rapide à développer, mais beaucoup plus lent à exécuter. Il a l'avantage aussi de pouvoir être optimisé automatiquement par le logiciel d'exploitation CCS. D'un autre côté, le code assembleur est plus difficile à lire, peu transportable et beaucoup plus complexe à programmer, mais il permet une utilisation maximale des ressources, donc du code rapide.

La validation des MUDs s'effectue en deux étapes principales. Premièrement par la comparaison des résultats de la plate-forme Visual Studio avec ceux de la plate-forme originale Matlab®. Ensuite en comparant les performances sur DSP avec celles sur Visual Studio. Une fois validé, on évalue le code d'un MUD avec les outils fournis par CCS, qui

sont l'horloge, le *Profiler* ou un *timer*. Avec le code assembleur, il est possible de calculer à la main le nombre de cycles utilisés.

Chapitre 5
Résultats d'implémentation des MUD dans un contexte WCDMA

Ce chapitre présente les divers résultats obtenus lors de l'évaluation de l'intégration du code des algorithmes MUD étudiés (Rake, MPIC, CF-MUD). Les résultats sont exprimés dans des tableaux et sous forme de graphiques.

Les tableaux fournissent un résumé précis des résultats obtenus selon une qualité de service et de débit des données transmises dictés par un taux d'erreur binaire (BER – Bit Error Rate) et un code OVSF spécifique. Le résultat « *Maximum d'usagers* » indique le nombre maximal d'usagers que le C6416 peut supporter en exécutant uniquement l'algorithme spécifié selon une ressource précise, soit sa fréquence ou sa quantité mémoire. Le résultat « *Cycles par donnée utilisateur* » exprime le nombre de cycles requis au C6416 pour traiter une seule donnée (soit d'information ou d'adaptation) d'un seul utilisateur de l'algorithme étudié. Ce résultat sera le même pour n'importe quel nombre d'usagers si la courbe du nombre de cycles en fonction du nombre d'usagers est linéaire, mais sera spécifique au nombre maximal d'usagers supportés si la courbe n'est pas linéaire. Le résultat « *Cycles maximums par utilisateur* » divulgue le nombre maximum de cycles que le C6416 requière pour traiter une trame ou adapter une série de filtres d'un seul utilisateur. Comme pour le

résultat précédent, la valeur déterminée sera la même ou non dépendamment de la linéarité ou de la non linéarité de l'algorithme.

Les graphiques présentés montrent en détail le comportement de l'algorithme selon certains paramètres variables et ce pour une ressource spécifique. Par conséquent, ils expriment l'évolution de la complexité de traitement des algorithmes étudiés en fonction de l'augmentation du nombre d'utilisateurs au récepteur.

5.1 Paramètres et critères d'évaluation de la complexité

Les systèmes cellulaires WCDMA de la 3G transmettent leur information par l'intermédiaire de trames. Une trame contient 38400 chips. Les normes stipulent que les systèmes doivent traiter une trame dans l'intervalle de 10 ms. La ressource DSP impliquée dans la limite de temps est le nombre de cycles d'instructions maximal qu'il peut exécuter en une seconde. Par conséquent, avec un DSP fonctionnant à 1GHz, le nombre maximal de cycles s'exécutant en 10 ms est de 10 millions. L'objectif visé de nos résultats est donc de traiter dans un DSP une trame en maximisant le nombre d'usagers en utilisant moins de 10 millions de cycles. On suppose ici que le DSP n'exécute que le MUD, on ne tient pas compte des autres opérations qu'il pourrait exécuter. Les programmes sont tous chargés dans la mémoire interne.

Les tableaux et les graphiques présentent l'utilisation de symboles pour représenter les différents paramètres qui influencent la complexité de traitement. Les paramètres communs à tous les algorithmes doivent être définis pour la compréhension des résultats (voir aussi la liste des symboles). On retrouve le nombre d'utilisateurs supportés U par le DSP, le nombre

de trajets multiples L employés pour rehausser la qualité de la réception, le nombre de données d'information N contenues dans une trame, le nombre de chips N_chips émis dans une trame et le facteur d'étalement de variable orthogonale *OVSF* (64, 16, 8 et 4), représentant le débit de service (16kb/s, 64kb/s, 144kb/s et 384kb/s).

Les figures de ce chapitre présentent l'évolution de la complexité de traitement de différentes méthodes en fonction de l'augmentation du nombre d'utilisateurs dans le système.

5.2 Détecteur RAKE

Cette section présente les résultats de l'évaluation de l'algorithme RAKE codé en C [BOU06]. La méthode Rake, actuellement utilisée dans l'industrie de la téléphonie cellulaire, présente une méthode de référence pour la comparaison avec les méthodes étudiées. Le tableau 5.1 énumère les paramètres, le tableau 5.2 et la figure 5.1 présentent les résultats.

Tableau 5.1 Paramètres du RAKE et du MPIC.

Paramètres	Valeurs	Algorithmes
U	1 à 50	RAKE ET MPIC
OVSF	64, 16 ou 4	RAKE ET MPIC
L	4	RAKE ET MPIC
N_chips	38400	RAKE ET MPIC
N	N_chips/OVSF	RAKE ET MPIC
N_étage	3	MPIC

On remarque que la courbe de complexité est linéaire par rapport au nombre d'usagers présents dans le système. La complexité varie beaucoup dépendamment de la valeur de

l'OVSF. En effet, plus l'OVSF est faible, plus il y a de données par trames. Par conséquent, compte tenu que le nombre de données reçu par trame et la complexité de traitement d'une donnée sont directement proportionnels à la valeur de l'OVSF, la complexité de traitement d'une trame devrait être pour le RAKE la même quel que soit l'OVSF utilisé. Donc, sans une programmation optimale en assembleur où la boucle de traitement des données est incorporée à celle de traitement des chips en une seule boucle, il n'y a pas possibilité de construire un pipeline qui soit suffisamment efficient pour permettre l'équivalence de complexité en nombre de cycle. Ceci dit, la complexité totale augmente implicitement plus le nombre de données est élevé malgré que la complexité d'une seule donnée soit simplifiée.

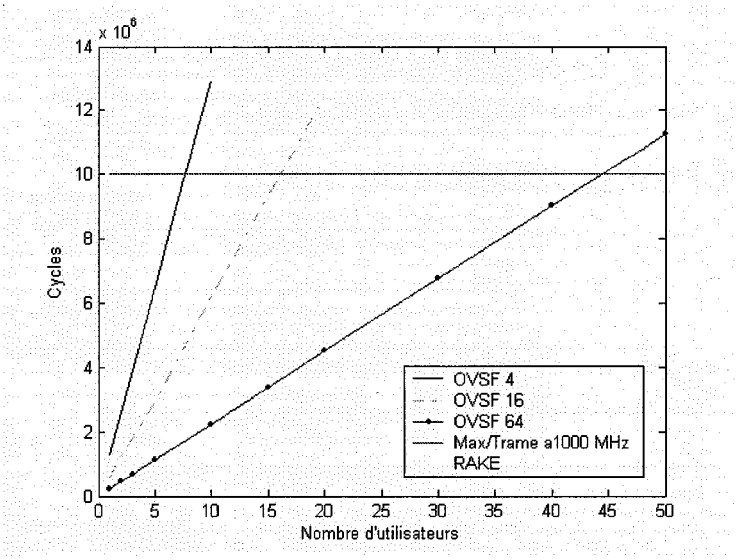


Figure 5.1 RAKE avec $N\_chips = 38400$, $L = 4$, $f = 1\text{GHz}$

Tableau 5.2 Résultats de l'évaluation du RAKE.

	OVSF		
	4	16	64
Maximum d'usagers	7	16	44
Cycles par donnée utilisateur	135	253	378
Cycles maximums par utilisateur	1300624	608338	226554

5.3 Détecteur MPIC

Cette section présente les résultats de l'évaluation de l'algorithme MPIC codé en C [BOU06]. L'intérêt d'évaluer cette méthode est qu'elle est très performante pour supprimer les interférences à accès multiples, par contre elle présente un niveau de complexité non pratique. Cet algorithme fonctionne avec plusieurs étages, par conséquent, il utilise un paramètre supplémentaire par rapport au RAKE qui est le nombre d'étages (*N_étage*). Afin d'atteindre des résultats de performance algorithmique d'intérêt, il faut considérer *N_étage*=5. Cependant, nous nous sommes limité à *N_étage*=3 dans notre évaluation sur DSP. Le tableau 5.1 énumère les paramètres, le tableau 5.3 et la figure 5.2 présentent les résultats.

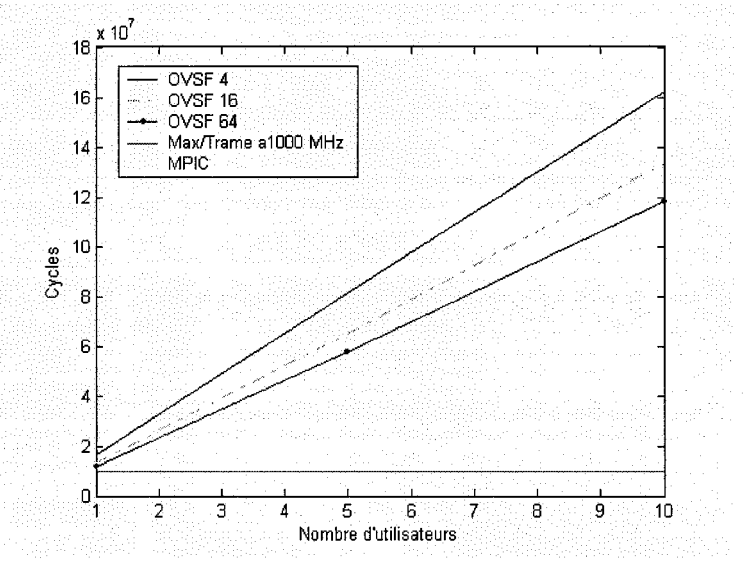


Figure 5.2 MPIC avec N_chips = 38400, L = 4, N_étage = 3, f = 1GHz.

Tableau 5.3 Résultats de l'évaluation du MPIC

	OVSF		
	4	16	64
Maximum d'usagers	0	0	0
Cycles par donnée utilisateur	1709	5504	19463
Cycles maximums par utilisateur	16409553	13210406	11678067

Tout comme le RAKE, la courbe de complexité est linéaire par rapport au nombre d'usagers présents dans le système. La complexité varie aussi considérablement dépendamment de la valeur de l'OVSF, et ce pour la même raison explicitée à la section précédente. On constate l'énorme complexité du MPIC pour le C6416 qui ne supporte aucun utilisateur.

5.4 Détecteur CF-MUD

Cette section présente les résultats de l'évaluation de l'algorithme CF-MUD. Comme il est l'algorithme d'intérêt de l'étude, on lui a porté plus d'attention en l'évaluant sur deux langages afin de comparer sa complexité avec celle du RAKE et du MPIC et d'atteindre un niveau de complexité minimal. Par conséquent les résultats suivants présentent la complexité du code assembleur et celle du code C [BOU06]. Les sous sections 5.4.1 à 5.4.7 inclusivement traitent du code assembleur, tandis que la dernière section 5.4.8 traite du code C. Le tableau 5.4 énumère les paramètres, les tableaux 5.5 à 5.11 et les figures 5.3 à 5.10 présentent les résultats.

Tableau 5.4 Paramètres généraux et paramètres spécifiques à CF-MUD

Paramètres	Valeurs	Paramètres	Valeurs
U	1 à 300	N_adapt_filtre	100
OVSF	64, 32, 16, 8 ou 4	N_adapt	100•n_SF
N_chips	38400	N_adapt_chips	N_adapt•OVSF
N	N_chips/OVSF	Nw	2•OVSF
N_SF	256/OVSF	Nv	3U

Vu que CF-MUD est un algorithme adaptatif, il comporte des paramètres de traitement qui lui sont propres. Par exemple, dans le cas de l'étude présente, les filtres du bloc Signature contiennent *Nw* coefficients. Le second bloc contient aussi un filtre muni de *Nv* coefficients. CF-MUD possède des paramètres supplémentaires liés à l'adaptation tels que le nombre de données d'adaptation par filtre (*N_adapt_filtre*), le nombre de données d'adaptation (*N_adapt*), le nombre de chips émises à l'adaptation (*N_adapt_chips*), le pas d'adaptation du bloc Signature ou bloc 1 (*mu_bloc1*) et du bloc Détection ou bloc 2 (*mu_bloc2*).

Comme les filtres de CF-MUD au bloc Signature sont bâtis sous le même principe que ceux du RAKE, le nombre de données émises et la complexité de traitement d'une donnée sont directement proportionnels à la valeur de l'OVSF, la complexité de traitement d'une trame est, pour le bloc Signature de CF-MUD codé en assembleur, la même quel que soit l'OVSF utilisé.

Les données d'information voyagent dans l'algorithme du bloc Signature au bloc Détection et leur traitement est effectué dans les fonctions de détection. Les filtres des fonctions de détection sont adaptés à partir des fonctions d'adaptation, lesquelles adaptent les filtres de

leur bloc respectif. Par conséquent, les données d'adaptation voyagent aussi du bloc Signature au bloc Détection. Par contre, les données d'adaptation traitées au premier bloc ne sont pas toutes valides pour être passées au second bloc puisque les premières données ne sont pas filtrées avec des coefficients suffisamment adaptés. Alors deux choix s'imposent. Le premier veut que l'on adapte les filtres du bloc 1, une fois convergés, on passe les données d'adaptation dans la fonction de détection du bloc 1, laquelle les émet au bloc 2. Le second veut que l'on utilise un certain pourcentage des dernières données d'adaptation, lesquelles sont filtrées par des filtres suffisamment adaptés, pour adapter le filtre du second bloc. Pour ce projet, afin de minimiser la complexité, le principe du second choix a été utilisé, mais toutes les données d'adaptation ont été utilisées pour adapter le bloc 2, en supposant qu'elles étaient toutes valides.

5.4.1 Fonction d'adaptation du bloc Signature

On remarque que la courbe de complexité est linéaire par rapport au nombre d'usagers présents dans le système (figure 5.3). Quel que soit l'OVSF, le DSP supporte toujours le même nombre d'usagers puisque le code de l'algorithme est conçu en plusieurs versions distinctes, lesquelles sont fonctions de chaque OVSF de manière à être optimale en suivant la caractéristique de complexité théorique de l'algorithme (tableau 5.5). C'est-à-dire que chaque fonction comporte un noyau de dimension inversement proportionnelle au nombre de données d'adaptation. Par conséquent, plus l'OVSF est élevé, moins on émet de données, mais plus leur traitement est complexe.

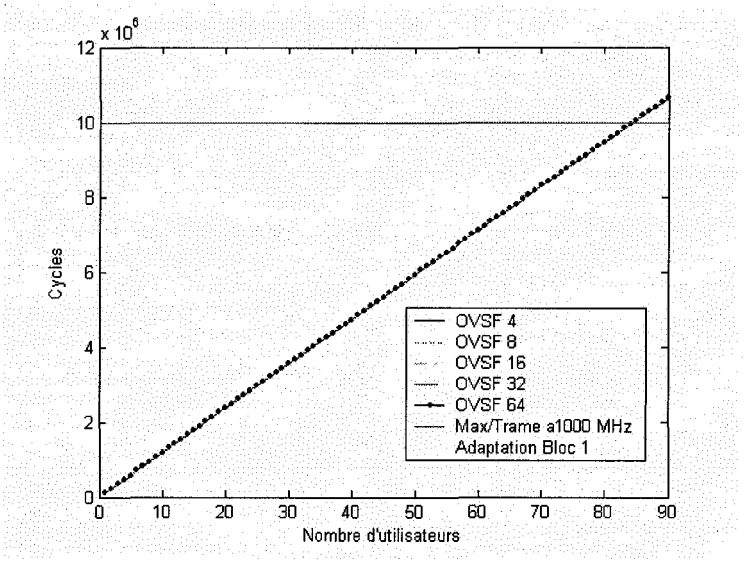


Figure 5.3 Adaptation du bloc Signature avec N_adapt_chips=256•100/OVSF, Nw=2•OVSF, f=1GHz.

Tableau 5.5 Résultats de l'évaluation de la fonction d'adaptation du bloc Signature

	OVSF				
	4	8	16	32	64
Maximum d'usagers	84	84	84	84	84
Cycles par donnée d'adaptation	19	37	74	148	296
Cycles maximums par utilisateur	118410	118410	118410	118410	118410

5.4.2 Fonction de détection du bloc Signature

On remarque que la courbe de complexité est linéaire par rapport au nombre d'usagers présents dans le système (figure 5.4). Quel que soit l'OVSF, le DSP supporte toujours le même nombre d'usagers puisque le code de l'algorithme est conçu en plusieurs versions distinctes, lesquelles sont fonctions de chaque OVSF de manière à être optimale en suivant la caractéristique de complexité théorique de l'algorithme (tableau 5.6). C'est-à-dire que chaque fonction comporte un noyau de dimension inversement proportionnelle au nombre de données d'information. Par conséquent, plus l'OVSF est élevé, moins on émet de données, mais plus leur traitement est complexe.

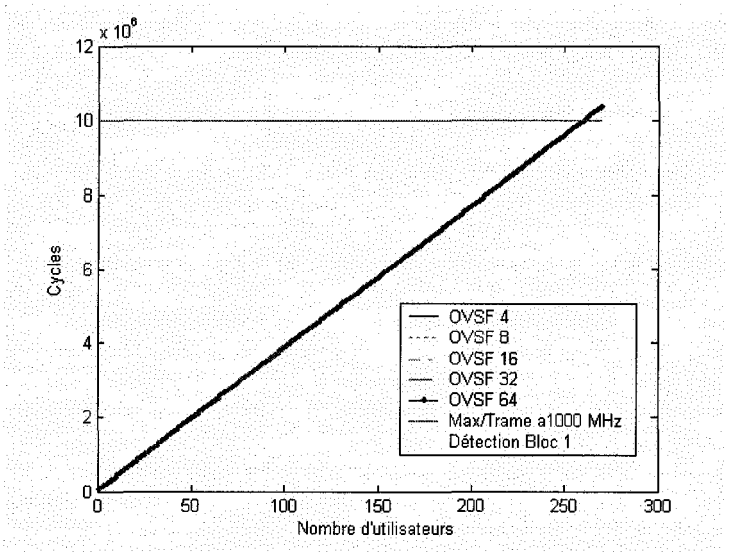


Figure 5.4 Détection du bloc Signature avec N_chips=38400, Nw=2•OVSF, f=1GHz.

Tableau 5.6 Résultats de l'évaluation de la fonction de détection du bloc Signature

	OVSF				
	4	8	16	32	64
Maximum d'usagers	260	260	260	260	261
Cycles par donnée utilisateur	4	8	16	32	64
Cycles maximums par utilisateur	38417	38393	38408	38376	38312

5.4.3 Fonction d'adaptation du bloc Détection

Pour tous les OVSFs, on remarque que la courbe de complexité n'est pas continue et qu'elle présente une tendance exponentielle par rapport au nombre d'usagers présents dans le système (figure 5.5). Par conséquent, les résultats statistiques du tableau sont relatifs au nombre maximum d'usagers supportés (tableau 5.6). Les discontinuités représentent un plateau de la complexité pour une série de quatre usagers consécutifs. Ceci est causé par l'implantation de la fonction qui traite un nombre d'usagers multiple de quatre afin d'y minimiser la complexité.

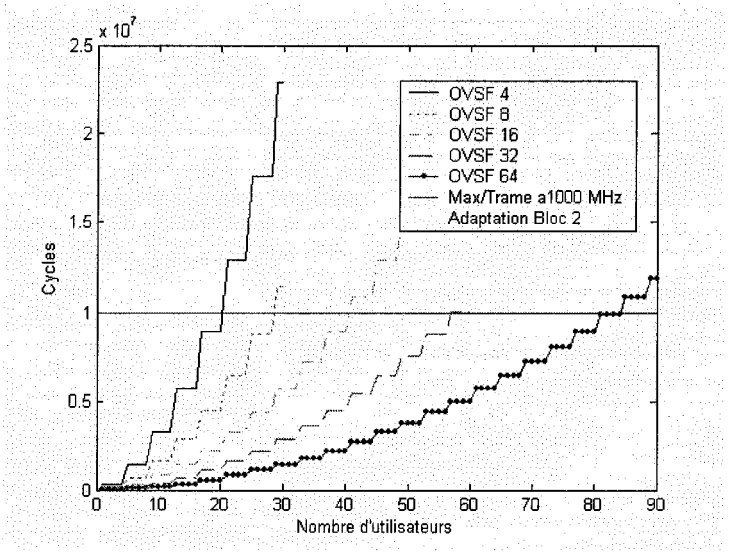


Figure 5.5 Adaptation du bloc Détection avec N_adapt_chips=256•100/OVSF, Nv=3U, f=1GHz.

Tableau 5.6 Résultats de l'évaluation de la fonction d'adaptation du bloc Détection.

	OVSF				
	4	8	16	32	64
Maximum d'usagers	20	28	40	56	84
Cycles par donnée d'adaptation	70	98	140	196	294
Cycles maximums par utilisateur	448012	313611	224010	156810	117609

5.4.4 Fonction de détection du bloc Détection

Pour tous les OVSFs, on remarque que la courbe de complexité n'est pas continue et qu'elle présente une tendance exponentielle par rapport au nombre d'usagers présents dans le système (figure 5.6). Par conséquent, les résultats statistiques du tableau sont relatifs au nombre maximum d'usagers supportés (tableau 5.7). Les discontinuités représentent un plateau de la complexité pour une série de huit usagers consécutifs. Ceci est causé par l'implantation de la fonction qui traite un nombre d'usagers multiple de huit afin d'y minimiser la complexité.

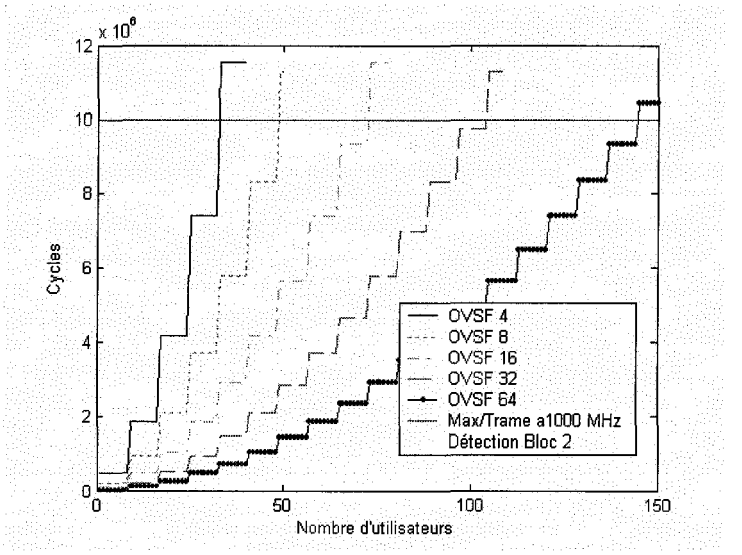


Figure 5.6 Détection du bloc Détection avec N_chips=38400, Nv=3U, f=1GHz.

Tableau 5.7 Résultats de l'évaluation de la fonction de détection du bloc Détection.

	OVSF				
	4	8	16	32	64
Maximum d'usagers	32	48	72	104	144
Cycles par donnée utilisateur	24	36	54	78	108
Cycles maximums par utilisateur	230409	172809	129609	93608	64808

5.4.5 Fonctions d'adaptation combinées

Les courbes affichées représentent la combinaison des courbes de la fonction d'adaptation du bloc Signature additionnées à celles de la fonction d'adaptation du bloc Détection (figure 5.7). Les résultats statistiques du tableau sont relatifs au nombre maximum d'usagers supportés (tableau 5.8).

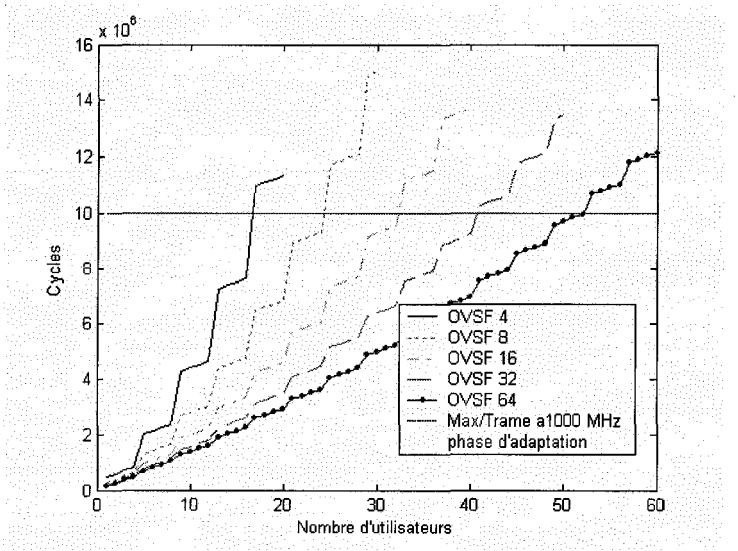


Figure 5.7 Adaptation des deux blocs avec N_adapt_chips=256•100/OVSF, Nw=2•OVSF, Nv=3U, f=1GHz.

Tableau 5.8 Résultats de l'évaluation de la combinaison des fonctions d'adaptation des deux blocs

	OVSF				
	4	8	16	32	64
Maximum d'usagers	16	24	32	40	52
Cycles par donnée d'adaptation	75	121	186	288	478
Cycles maximums par utilisateur	476823	387222	297621	230420	191220

5.4.6 Fonctions de détection combinées

Les courbes affichées représentent la combinaison des courbes de la fonction de détection du bloc Signature additionnées à celles de la fonction de détection du bloc Détection (figure 5.8). Les résultats statistiques du tableau sont relatifs au nombre maximum d'usagers supportés (tableau 5.9).

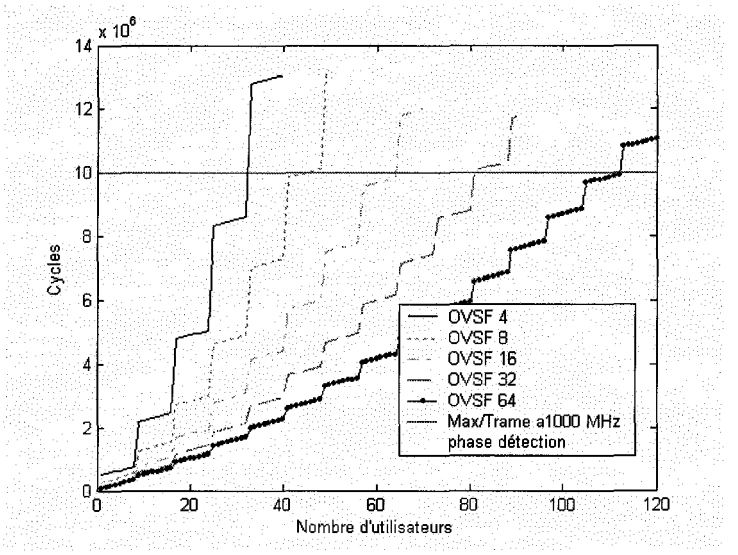


Figure 5.8 Détection des deux blocs avec N_chips=38400, Nw=2•OVSF, Nv=3U, f=1GHz.

Tableau 5.9 Résultats de l'évaluation de la combinaison des fonctions de détection des deux blocs

	OVSF				
	4	8	16	32	64
Maximum d'usagers	32	44	64	80	112
Cycles par donnée utilisateur	28	47	64	92	148
Cycles maximums par utilisateur	268826	226912	153618	110385	88721

5.4.7 Algorithme en entier

Les courbes présentées à la figure 5.9 représentent la complexité de traitement de l'algorithme CF-MUD en entier. Les résultats statistiques du tableau 5.10 sont relatifs au nombre maximum d'usagers supportés.

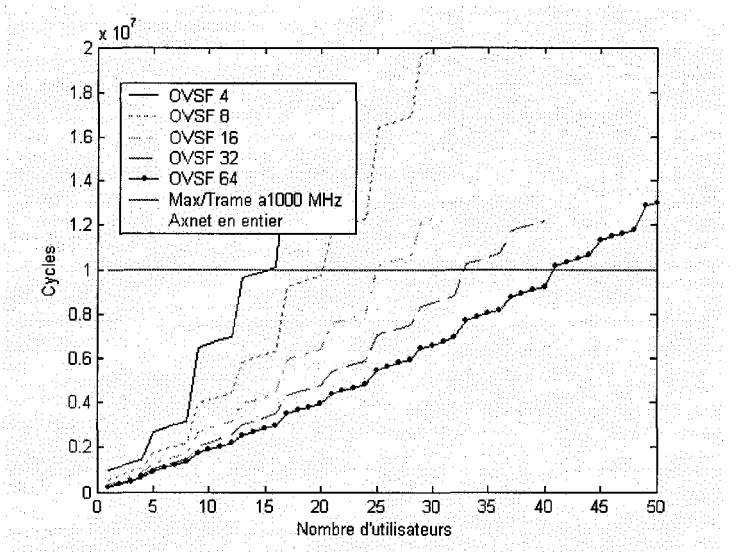


Figure 5.9 CF-MUD en entier (version assembleur) avec N_chips=38400, N_adapt_chips=256•100/OVSF, Nw=2•OVSF, Nv=3U, f=1GHz.

Tableau 5.10 Résultats de l'évaluation de CF-MUD

	OVSF			
	4	8	16	64
Maximum d'usagers	15	20	24	40
Cycles par donnée utilisateur	69	101	139	385
Cycles maximums par utilisateur	662025	484507	334442	230743

5.4.8 CF-MUD codé en C

Cette sous section présente les résultats de l'évaluation CF-MUD codé en C. L'intérêt d'évaluer l'algorithme en C après l'avoir fait en assembleur est d'étudier le rapport d'optimisation entre les deux langages. Le tableau 5.11 et la figure 5.10 présentent les résultats.

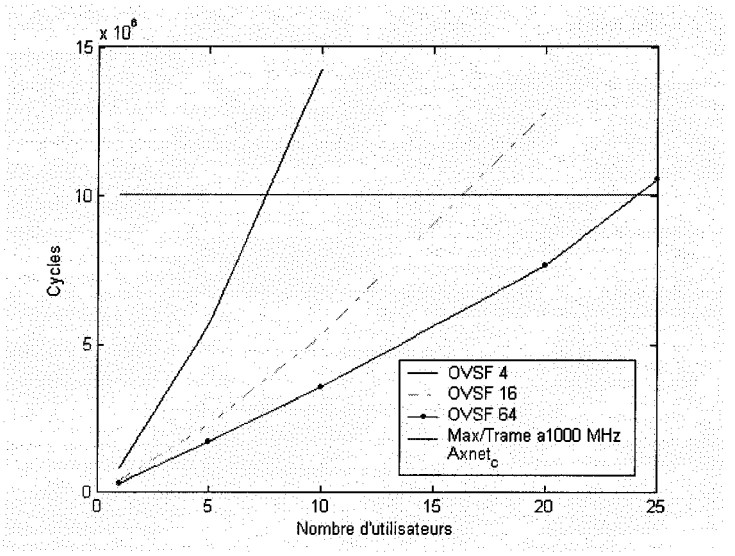


Figure 5.10 CF-MUD en entier (version C) avec N_chips=38400, N_adapt_chips=256•100/OVSF, Nw=2•OVSF, Nv=3U, f=1GHz.

Tableau 5.11 Résultats de l'évaluation de CF-MUD codé en C

	OVSF		
	4	16	64
Maximum d'usagers	6	15	23
Cycles par donnée utilisateur	156	274	714
Cycles maximums par utilisateur	1497521	656435	428155

Pour tous les OVSFs, on remarque que, contrairement au code assembleur, la courbe de complexité est continue et qu'elle présente aussi une tendance exponentielle par rapport au nombre d'usagers présents dans le système. On ne retrouve pas de plateau puisque le code du bloc détection est implanté pour effectuer un traitement d'un multiple de un usager à la fois. Les résultats statistiques du tableau sont relatifs au nombre maximum d'usagers supportés.

5.4.9 Tableaux comparatifs entre CF-MUD codé en assembleur et en C

Les deux tableaux 5.12 et 5.13 présentent un comparatif de la complexité des codes assembleur et C de l'algorithme CF-MUD. Ces tableaux permettent de visualiser facilement le rapport de complexité du code C sur le code assembleur [BOU06]. L'objectif de déterminer ce rapport est de justifier l'intérêt ou non de coder en assembleur au lieu de coder en C. Puisque le code assembleur est beaucoup plus compliqué à générer. La complexité est donnée en cycles.

Tableau 5.12 Comparaison de la complexité du code C et assembleur de CF-MUD pour 5 usagers.

OVSF 4	**Adaptation bloc 1**	**Détection bloc 1**	**Adaptation bloc 2**	**Détection bloc 2**
Assembleur	592050	192085	1433744	460900
C	2208069	1080035	1984073	384213
Rapport C/A	*3.7*	*5.6*	1.4	*0.8*
OVSF 16	**Adaptation bloc 1**	**Détection bloc 1**	**Adaptation bloc 2**	**Détection bloc 2**
Assembleur	591928	192109	358544	115300
C	1392074	290042	496073	96213
Rapport C/A	*2.4*	*1.5*	*1.4*	*0.8*
OVSF 64	**Adaptation bloc 1**	**Détection bloc 1**	**Adaptation bloc 2**	**Détection bloc 2**
Assembleur	591856	191629	89744	28900
C	1118078	415084	124073	24213
Rapport C/A	*1.9*	*2.2*	*1.4*	*0.8*

Tableau 5.13 Comparaison de la complexité du code C et assembleur de CF-MUD pour 10 usagers.

OVSF 4	**Adaptation bloc 1**	**Détection bloc 1**	**Adaptation bloc 2**	**Détection bloc 2**
Assembleur	1184100	384170	3225778	1843364
C	4416134	2160030	5888143	1824671
Rapport C/A	*3.7*	*5.6*	1.8	*1.0*
OVSF 16	**Adaptation bloc 1**	**Détection bloc 1**	**Adaptation bloc 2**	**Détection bloc 2**
Assembleur	1183954	384149	806578	460964
C	2784144	580075	1472143	456671
Rapport C/A	*2.4*	*1.5*	*1.8*	*1.0*
OVSF 64	**Adaptation bloc 1**	**Détection bloc 1**	**Adaptation bloc 2**	**Détection bloc 2**
Assembleur	1183816	383189	201778	115364
C	2236148	830149	368143	114671
Rapport C/A	*1.9*	*2.2*	*1.8*	*1.0*

5.4 Conclusion

Les résultats précédents confirment la simplicité du RAKE. En effet, parmi tous les algorithmes codés en C, le RAKE supporte en moyenne plus d'usagers que le MPIC et CF-MUD. On remarque que le nombre d'usagers supportés par CF-MUD est sensiblement le même que celui du RAKE pour de faibles valeurs d'OVSF, tandis que plus l'OVSF est considérable plus l'écart se creuse. Ceci s'explique par la courbe non linéaire du bloc Détection de CF-MUD, en effet, la méthode présente une complexité exponentielle en fonction du nombre d'usagers. Donc plus l'OVSF est élevé, plus l'algorithme devrait supporter d'usagers, mais vu l'exponentielle, le nombre d'usagers supportés est vite écrasé.

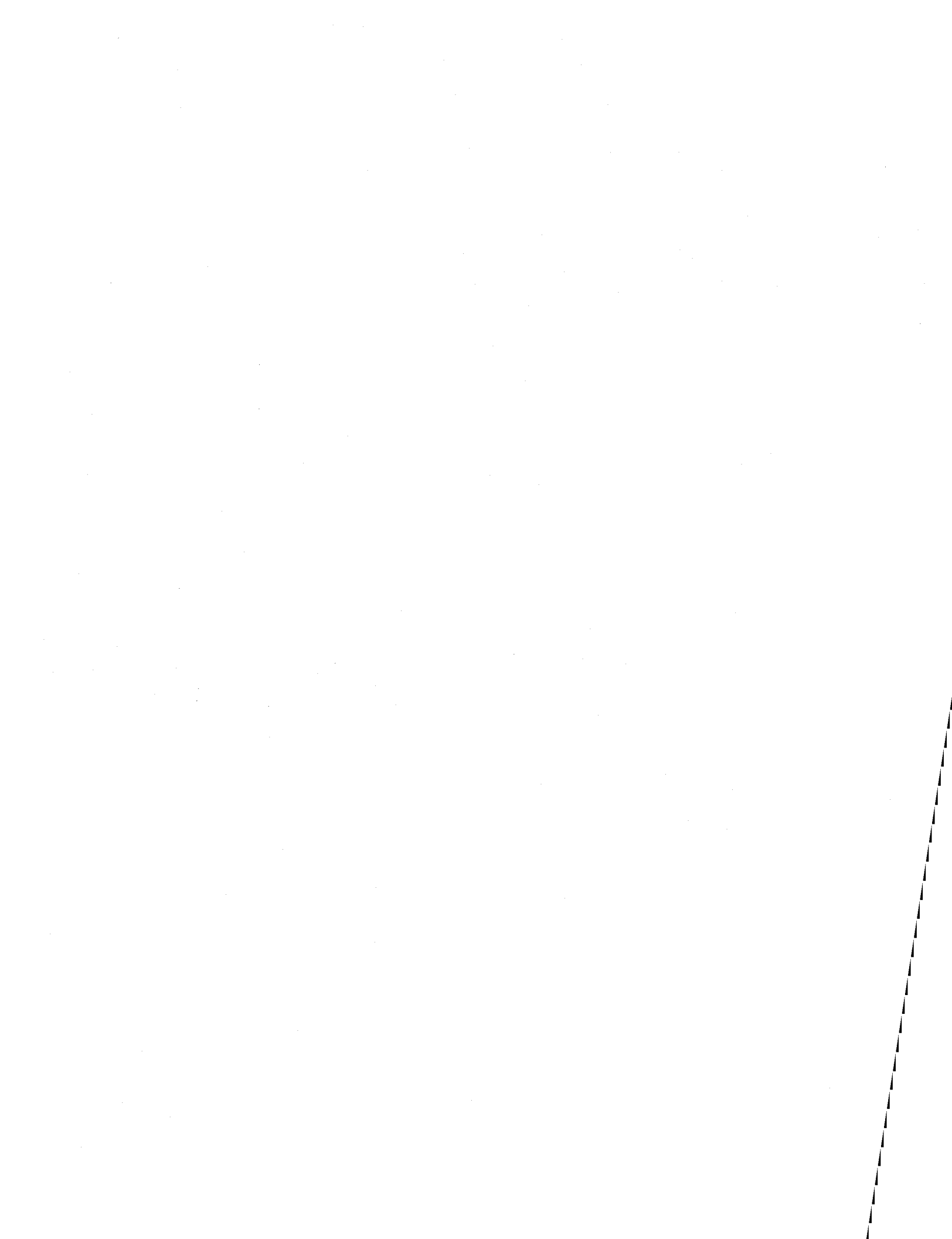
Le MPIC démontre une complexité catastrophique, ne supportant aucun usager, même à OVSF 64. Malgré que le traitement qu'il effectue soit remarquable, cette méthode demeure toujours beaucoup trop coûteuse pour permettre son implantation dans un système cellulaire. Les résultats prouvent que dans le meilleur des cas, le traitement d'une donnée avec le MPIC est 13 fois plus complexe qu'avec le RAKE ou CF-MUD.

L'évaluation de la version codée en assembleur de CF-MUD est moins complexe que la version optimisée à partir du C. Ceci dit, pour les OVSF 4, 16 et 64, la version assembleur permet de supporter 15, 24 et 40 usagers tandis que la version C en permet 6, 15 et 23, ce qui est considérable quand le temps est une ressource limitée, comme spécifier dans les normes de la 3G.

En étudiant les rapports de complexité de la sous section 5.4.9, on remarque que la programmation en assembleur est plus payante au niveau du bloc 1 (Signature) qu'à celui

du bloc 2 (Détection). Effectivement, au bloc 1, le code assembleur semble deux fois plus rapide que le C tandis qu'au bloc 2, on remarque que la complexité des deux versions se chevauche. De plus, au bloc 1, l'intérêt du code assembleur grandit plus la valeur de l'OVSF est faible, affirmant ainsi la difficulté de « l'optimiseur C » (algorithme d'optimisation du code C lors de la compilation) à construire un pipeline efficient. Aussi, au bloc 2, on remarque que plus le nombre d'usagers est élevé, plus le code assembleur à tendance à augmenter son efficience.

Bref, l'algorithme CF-MUD présente une complexité qui se rapproche beaucoup de celle du RAKE. L'effort requis pour programmer directement en assembleur se justifie seulement lorsque le temps maximal imposé au traitement ne peut être atteint avec le code C ou que le code assembleur va permettre de libérer un nombre considérable de ressource. En effet « l'optimiseur C » est très efficace et génère un code peu complexe contenant un pipeline efficient.



Chapitre 6
Conclusion

Les systèmes cellulaires de troisième génération ont pour but de remplacer leurs prédécesseurs parce que ces derniers ne permettent pas l'implantation de nouveaux services requérant un haut débit binaire et qu'ils ne supportent plus une capacité suffisante d'utilisateurs puisque la bande passante allouée est saturée. Pour contrer ce problème, on utilise la technologie d'accès multiples CDMA, laquelle sépare l'information de chaque usager d'une même cellule par un code, et cela, sur une même fréquence. Ceci permettant de libérer des fréquences sur la bande passante. L'inconvénient de cette technique est que les signaux usagers interfèrent les uns avec les autres, générant de l'interférence d'accès multiple. La détection multi-utilisateur a donc pour but d'éliminer ou de minimiser l'effet de la MAI dans un système en estimant les signaux interférant et les supprimant du signal à restituer. L'objectif de ce projet était d'évaluer la complexité de différents MUDs afin de déterminer lesquels sont les plus susceptibles d'être implantés sur DSP tout en rencontrant les contraintes temps réel des spécifications régissant les systèmes cellulaires. Les systèmes cellulaires visés par ce projet sont ceux de troisième génération. Leurs caractéristiques principales ont donc été imposées par l'Union Internationale des Télécommunications, qui a pour mission de standardiser les systèmes à travers le monde.

L'étude présentée est une évaluation comparative d'algorithmes MUD. Cette évaluation se justifie dans la problématique d'accroître la capacité des systèmes, sans augmenter le nombre d'infrastructure. Les systèmes basés sur le CDMA acceptent théoriquement un grand nombre d'usagers sur une même fréquence, mais les usagers créent de l'interférence MAI entre eux, saturant vite le système. Pour éliminer l'interférence et accroître le nombre d'usager, on utilise des algorithmes MUD. Dans la plupart des cas, les MUDs les plus performants sont aussi les plus complexes. Comme le traitement s'effectue en temps réel, les unités de traitement les plus performantes en terme de vitesse de traitement saturent aussi lorsqu'il y a trop d'usagers. Par conséquent, un compromis performance versus complexité du MUD s'impose afin d'admettre la capacité la plus élevée avec une même unité de traitement.

L'objectif principal de ce projet a été atteint. Trois MUDs de différentes performances et complexités ont été étudiés et programmer pour générer des données comparatives. Le RAKE a été sélectionné, malgré qu'il soit un récepteur peu performant, puisqu'il présente une faible complexité et qu'il a été utilisé dans les systèmes de seconde génération. Il représente donc une référence de plancher par rapport à sa complexité. Le MPIC a été sélectionné, malgré sa haute complexité, puisque ses performances sont presque idéales. Il représente donc une référence de plafond par rapport à sa complexité. CF-MUD est un nouvel algorithme présentant de bonnes performances, mais dont sa complexité n'avait pas encore été étudiée lors d'une intégration réelle sur DSP. Il représente donc l'élément principal de l'étude. Pour de très bonnes performances de traitement, il arbore une complexité supérieure au RAKE, mais beaucoup plus près de ce dernier que du MPIC.

L'évaluation des algorithmes s'est faite suivant une méthodologie de travail suivant trois étapes principales : étude de la théorie inhérente à la détection multi-utilisateur, transformation des algorithmes à virgule flottante en algorithmes à virgule fixe et programmation de ces derniers de manière optimale.

L'étude sur la détection multi-utilisateur nous a permis d'approfondir sur son principe de base qui repose sur la technologie d'accès multiple CDMA. Le CDMA repose sur l'étalement de spectre effectué sur un signal à l'aide d'un code unique permettant de différencier ce signal de celui d'un autre usager. Nous avons aussi approfondi sur les MUDs, lesquels reposent sur le principe d'estimer l'interférence MAI puis de la supprimer du signal reçu afin de permettre le recouvrement optimal du signal désiré. Enfin, nous avons étudié le principe de fonctionnement des algorithmes mis à l'étude, c'est-à-dire le RAKE, le MPIC et CF-MUD.

La plate-forme Matlab® simulait les MUDs en virgule flottante avec une haute précision de calcul. Nous avons modifié la plate-forme afin qu'elle représente mieux l'architecture de destination des programmes. Le DSP fonctionne avec une résolution 16 bits et repose sur une architecture à virgule fixe. Alors les signaux et les algorithmes ont été étudiés pour amener la plate-forme sous le même format que le DSP. Nous avons donc quantifié les signaux dans un intervalle de valeurs représentables. Les algorithmes ont aussi été modifié pour permettre la mise à l'échelle des variables et signaux résultants en cours d'exécution.

L'évaluation des algorithmes a nécessité leur programmation dans un même langage pour assurer une certaine régularité. Les programmes ainsi créés utilisent les instructions les

moins coûteuses en cycles d'instruction. Ainsi, les divisions ont été remplacées par des décalages à droite et certaines boucles ont été déroulées. Afin de réduire leur complexité au minimum, on a utilisé l'outil d'optimisation de CCS pour construire un pipeline assurant l'optimisation du parallélisme admis par l'architecture et en remplissant les cases de retard par des instructions utiles. Ceci a eu pour effet de bâtir des programmes très rapides en terme de vitesse de traitement.

Malgré les performances d'optimisation de l'outil d'optimisation de CCS, il s'avérait très intéressant de programmer les algorithmes directement en langage assembleur optimisé. Étant donné la complexité de la tâche, seulement CF-MUD a été programmé de la sorte, puisqu'il représentait l'algorithme d'intérêt de l'étude. L'optimisation manuelle assurait d'introduire tous les principes d'optimisation du code du DSP TMS320C6416. En effet, l'algorithme a été programmé selon un pipeline utilisant le parallélisme de l'architecture, les cases de retard ont été remplacées par des instructions utiles, la largeur des mots de chargement a été optimisée, les boucles ont été déroulées pour prendre avantage de l'optimisation des mots de chargement et certaines boucles ont même été fusionnées pour construire une boucle unique admettant au noyau du pipeline un plus grand nombre d'itérations sans prologue (chargement) et épilogue (déchargement).

L'optimisation manuelle et l'optimisation automatique de l'outil d'optimisation de CCS pour l'algorithme CF-MUD produisent un code relativement peu complexe. L'optimisation manuelle, dans ce cas-ci, alloue une capacité seulement deux fois plus importante que celle allouée par l'autre méthode. Ce qui permet de conclure que, pour une évaluation de

complexité, l'outil d'optimisation de CCS est suffisant, mais que lors d'une implantation réelle, il est payant de générer un code optimal.

Finalement, ce projet permet de situer la complexité de l'algorithme CF-MUD par rapport à celles du RAKE et du MPIC et en même temps de valider le taux de complexité du code générer par l'outil d'optimisation de CCS. Des recherches sont actuellement en cours afin de permettre à CF-MUD de converger plus rapidement lors de l'adaptation de ses coefficients, ce qui devrait réduire considérablement sa complexité.

Bibliographie

[ALS02] A. M. Alsolaim (2002). Dynamically Reconfigurable Architecture for Third Generation Mobile Systems. Thèse doctorale, College of Engineering and Technology, Ohio University.

[AXI04] Axiocom Inc. A New MMSE Based Cascade Filter MUD for MAI and ISI Mitigation. Technical report. Octobre 2004.

[BEL98] M. Bellanger (1998). Traitement numérique du signal. Les éditions Dunod. Paris, France.

[BOU06] S. Boucher, Annexes – Implantation sur DSP d'algorithmes de détection multi-utilisateur, Rapport technique, Laboratoire des signaux et systèmes intégrés, UQTR, 2006.

[DAH01] A. O. Dahmane et D. Massicotte. DS-CDMA Receivers in Rayleigh Fading Multipath Channels : Direct vs. Indirect Methods. Laboratory of Signal and Systems Integration., Université du Québec à Trois-Rivières. 2002.

[DAH02] A. O. Dahmane et D. Massicotte. Wideband CDMA Receivers for 3G Wireless Communications : Algorithm and Implementation Study. Verdú, S., *Multiuser detection.* 1998, New York: Cambridge University Press. [GAB03] Gabrea. (2003). Processeur numérique du signal et ses applications. Note de cours (Chapitre 5), École de technologie supérieure, Montréal.

[DAH04] Dahmane, A.O., *Méthode de détection à usagers multiples pour les systèmes de communication DS-CDMA*, Thèse doctoral en *génie électrique*. 2004, Université du Québec à Trois-Rivières: Trois-Rivières.

[GUO00] Y. J. Guo. Advanced base station technologies for UTRAN. Electronics & Communication Engineering Journal. Juin 2000.

[HAL95] A. Duel-Hallen. Multiuser Detection for CDMA Systems. IEEE Personal Communications. Avril 1995.

[HOQ06] Q.-T. Ho, D. Massicotte, A.O. Dahmane "A Low Complexity Adaptive Multiuser Detector and FPGA Implementation for Wireless DS-WCDMA Communication Systems", EURASIP Journal on Applied Signal Processing – Special Issue on Designs Methods for DSP Systems, Vol.2006, Article ID52919, pp.1-12.

[IEC98] Wideband Code Division Multiple Access (W-CDMA) Tutorial. The International Engineering Consortium. 1998.

[KOR01] J. Korhonen (2001). Introduction to 3G Mobile Communications. Norwood, MA : Artech House.

[KUN91] M. Kunt et al. (1991). Techniques modernes de traitement numérique des signaux. Presses polytechniques et universitaires romandes. Lausanne.

[MA02] Hsi-Pin Ma. Design and Implementation of an Uplink Baseband Receiver for Wideband CDMA Communications. IEICE Trans. Fundamentals, Vol.E85-A, No.12, pp. 2813-2821. Décembre 2002.

[SAN01] J. Sanchez, M. Thioune (2001) UMTS services, architecture et WCDMA. Paris, France : Lavoisier.

[TI00] Texas Instruments Inc.. (2000). TMS320C6000 CPU and Instruction Set Reference Guide, http://focus.ti.com/lit/ug/spru189f/spru189f.pdf, (16 décembre 2003).

[TI01] Texas Instruments Inc.. (2001). TMS320C64x Technical Overview, http://focus.ti.com/lit/ug/spru395b/spru395b.pdf, (16 décembre 2003).

[TI02] Texas Instruments Inc.. (2002). TMS320C6000 Programmer's Guide, http://focus.ti.com/lit/ug/spru198g/spru198g.pdf, (24 février 2005).

[TI03] Texas Instruments Inc.. (2003). TMS320C6414T, TMS320C6415T, TMS320C6416T Fixed-Point Digital Signal Processors, http://focus.ti.com/lit/ds/symlink/tms320c6416t.pdf, (23 février 2005).

[TI04] Texas Instruments Inc.. (2004). TMS320C64x DSP Two-Level Internal Memory Reference Guide, http://focus.ti.com/lit/ug/spru610b/spru610b.pdf, (24 février 2005).

[VER98] Verdú, S., *Multiuser detection*. 1998, New York: Cambridge University Press.