UNIVERSITÉ DE MONTRÉAL

DÉVELOPPEMENT D'UNE MÉTHODOLOGIE DE CODESIGN MATÉRIEL/LOGICIEL POUR DES APPLICATIONS DE COMMUNICATIONS À HAUTE VITESSE.

ISABELLE CAMPAGNA
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE ET DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE ÉLECTRIQUE)
MARS 2000

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

DÉVELOPPEMENT D'UNE MÉTHODOLOGIE DE CODESIGN MATÉRIEL/LOGICIEL POUR DES APPLICATIONS DE COMMUNICATIONS À HAUTE VITESSE

Présenté par: CAMPAGNA Isabelle

En vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées

A été dûment accepté par le jury d'examen constitué de:

M. Savaria Yvon, Ph.D., président

M. Aboulhamid Moustapha, Ph.D., membre

M. Bois Guy, Ph.D., membre et directeur de recherche

M. Houle Jean-Louis, Ph.D., membre et co-directeur de recherche

Remerciements

Je tiens à remercier mon directeur de recherche Guy Bois, pour son soutien et ses encouragements tout au long de ce projet. Son enthousiasme est communicatif. Je voudrais également remercier mon codirecteur de recherche Jean-Louis Houle. Je ne peux passer sous silence les conseils et l'aide apportés par Moustapha Aboulhamid de l'Université de Montréal.

Deux compagnies ont subventionné notre projet de recherche, Mentor Graphics ainsi que Nortel. Je voudrais remercier ces deux compagnies, et plus particulièrement Jacques Baillargé de Mentor Graphics qui a plusieurs fois fait le lien entre notre groupe de recherche et les deux partenaires industriels Il a été d'une aide précieuse pour l'installation et l'utilisation de plusieurs outils.

En terminant, je voudrais remercier Yannick Héneault et Geneviève Cyr qui ont partagé avec moi les hauts et les bas de la maîtrise. Et je ne voudrais surtout pas oublier ma famille et mes amis.

Résumé

Les systèmes embarqués étant de plus en plus gros et complexes, des méthodologies de conception sont nécessaires afin d'obtenir le système le plus performant possible, tout en gardant les coûts de développement et de fabrication les plus bas possibles. Ce sont les raisons pour lesquelles, nous avons voulu développer une méthodologie de codesign matériel/logiciel. L'objectif de ce projet de recherche est de développer une méthodologie qui sera testée et validée sur une application de communications à haute vitesse.

Le codesign matériel/logiciel est en fait la conception d'un système en utilisant des techniques de conception matérielle et logicielle. Les développements du matériel et du logiciel sont intégrés dans une méthodologie commune. La plupart des systèmes numériques utilisés comme systèmes embarqués consistent en un processeur d'utilisation générale, d'une mémoire et d'un circuit spécifique fait en matériel. Ce matériel peut être des ASICs (Application-Specific Integrated Circuit) ou des FPGAs (Field Programmable Gate Array). Des exemples de tels systèmes se retrouvent dans des instruments médicaux, des véhicules, des réseaux automatisés et des systèmes de communications.

Nous avons développé trois applications, c'est-à-dire trois modems de la famille xDSL, pour tester et valider notre méthodologie. Ces trois modems sont le ADSL (Asymmetric Digital Subscriber Line), le Universal ADSL qui est une version légère du ADSL et finalement le VDSL (Very High Bit-Rate Digital Suscriber Line) qui est une version plus performante du ADSL. Ces trois modems utilisent la même technologie, appelés DMT (Discrete Multitone), mais façons différentes. Nous avons évalué quatre critères pour les modems qui sont le temps d'exécution, les temps de communications, la surface du matériel et finalement la dissipation de puissance. Pour

fusionner les valeurs de ces différents critères en une seule représentant la qualité du modèle, nous utilisons une fonction objectif. Puis, un algorithme de partitionnement nous permet de déterminer la qualité de plusieurs modèles différents.

Notre méthodologie nous a permis de trouver un partitionnement matériel/logiciel optimal pour un modem. La différence principale entre notre méthodologie et celles développées précédemment, est que nous avons essayé d'intégrer le plus possible, des outils commerciaux existants. Pour composer avec ces nouvelles réalités, chacun des blocs des modems est évalué une seule fois et cela avant d'utiliser l'algorithme de partitionnement. Notre méthodologie est efficace mais il aurait été plus facile de le démontrer en utilisant une application moins complexe ou un processeur plus performant. Cette méthodologie pourrait être utilisée dans le futur lors de la conception de systèmes embarqués pour vérifier les choix des concepteurs d'une façon structurée et efficace.

Abstract

Embedded systems are becoming bigger and increasingly complex. Therefore, design methodologies are necessary in order to get the most effective system, while keeping development and manufacturing cost as low as possible. Consequently, the objective of this research project is to develop a methodology to test and validate hardware/software systems such as these used in high speed communications.

Hardware/software codesign is a methodology whose the goal is to integrate the hardware and the software parts of a system. Most of the digital systems are used as embedded systems. They consist of a general purpose processor, a memory and application specific circuits made in hardware. These circuits can be ASICs (Application-Specific Integrated Circuit) or FPGA (Field Programmable Gate Array). Examples of such systems can be found in medical instruments, transportation vehicles, automated networks, and communications systems.

We developed three related applications, selected from the xDSL family, to test and to validate our methodology. These three modems are the ADSL (Asymmetric Digital Subscriber Line), the Universal ADSL that is a light version of the ADSL and finally the VDSL (Very High Bit - Rate Digital Suscriber Line) which is the most effective version of the ADSL. These three modems use the same technology, called DMT (Discret Multitone), but in different ways. For the modem evaluation, we use four metrics: execution time, communications time, hardware area and power dissipation. We put the metrics values together in an cost fonction to find the model quality. Then a partitioning algorithm allows us to find the quality of different models.

Our methodology enabled us to find an optimal hardware/software partitioning for a modem. The difference between our methodology and those developed previously, is that we used as much as possible, existing tools. To compose with these new realities, each block of the modems are evaluated only once before using the partitioning algorithm. Our methodology is effective but it would have been easier to demonstrate it with less complex applications or with a more powerful processor. This methodology could be used in the future when designing embedded systems to validate the design made by the designers in a regular and effective way.

Table des matières

Liste des tableaux

Liste des figures

Liste des annexes

Liste des abréviations

ADSL	(Asymmetric Digital Subscriber Line) Ligne Numérique à Paire Asymétrique (LNPA)
ANSI	(American Standard Institute) Organisme nord Americain de normalisation membre de l'ISO.
ASIC	(Application-Specific Integrated Circuit) Circuit Intégré spécialisé. En général, ils sont produits en petites quantités, car c'est du sur-mesures.
CAD	(Computer Aided Design) Conception Assistée par Ordinateur, ou CAO en français.
CAP	(Carrierless Amplitude/Phase Modulation) Modulation Amplitude/Phase sans Porteuse, Technique de modulation developpée par "Bell Labs". Première technologie de transmetteur ADSL à être déployée commerciallement, CAP est une variation de la modulation d'amplitude en quadrature (MAQ).
FIFO	(First In First Out) Premier entré, premier sorti. C'est-à-dire une queue, une file d'attente.
HDSL	(High bit/data rate Digital Subscriber Line/Loop) Technique de transmission. Elle permet l'utilisation des fils de cuivre à paires torsadées pour transmettre des signaux à des débits allant de 784 Kbps à 2 048 Kbps selon la technique de codage et le nombre de paires utilisées.
IEEE	Institute of Electrical and Electronics Engineers
POTS	(Plain Old Telephone Service) "Le Bon Vieux Téléphone" Service téléphonique de base sur paire torsadée. Le téléphone utilise la bande 300-3400 Hz, et tout service qui veut partager le médium

	avec le téléphone doit soit utiliser une bande de fréquence supérieure, soit la voix téléphonique doit être numérisée et être groupée(multiplexée) avec les autres signaux de données.
QAM	(Quadrature Amplitude Modulation) voir Modulation d'Amplitude en Quadrature
SDL	(Specification and Description Language) voir Langage de Spécification et de Description

Avant-propos

Le codesign matériel/logiciel est de plus en plus utilisé dans l'industrie. Par contre, les concepteurs n'ont pas de méthodologie pour les guider lors de la conception de leurs systèmes. Ils utilisent plutôt leur instinct et leur expérience pour déterminer de bons partitionnements entre le matériel et le logiciel, sans vraiment vérifier et valider leurs choix. Les gens de Nortel nous ont donc demandé de développer une telle méthodologie. Après avoir fait la conception d'un modem Universal ADSL, nous devions valider et vérifier leurs choix lors du partitionnement matériel/logiciel. Nous avons également examiné d'autres modems de la même famille que le Universal ADSL, pour vérifier les possibilités de les implanter en utilisant une méthodologie de codesign.

Notre projet était également subventionné par une autre compagnie, Mentor Graphics. Dans leur cas, ils étaient intéressés par une méthodologie de codesign qui utiliserait leurs outils de conception, de simulation et de synthèse. C'est la raison pour laquelle nous avons essayé le plus possible d'intégrer des outils conçus par Mentor Graphics à l'intérieur de notre méthodologie.

Nous avons donc associé les demandes de nos deux partenaires industriels pour en faire un seul projet qui est devenu comme son titre l'indique le développement d'une méthodologie de codesign matériel/logiciel pour des applications de communications à haute vitesse.

Chapitre 1
Introduction

1.1 Définition du codesign

Les systèmes sont de plus en plus gros et de plus en plus complexes. La complexité croissante des systèmes numériques ainsi que la disponibilité de différentes technologies pour leur implantation ont complètement révolutionné le processus de conception [AdTh96]. Plusieurs systèmes sont aujourd'hui faits d'un ensemble de matériel et de logiciel. Ce type d'implantation obtenu par un processus de partitionnement matériel/logiciel aussi appelé codesign, permet de tirer partie des avantages d'une implantation en logiciel tout autant que des avantages d'une implantation en matériel. En fait, le codesign matériel/logiciel essaie d'intégrer les techniques de conception du logiciel et du matériel dans le but de concevoir une méthode de conception unique pour la création d'un système.

La plupart des systèmes numériques qui sont utilisés pour des systèmes embarqués consistent en un processeur d'utilisation générale, d'une mémoire et de circuits spécifiques fait en matériel. Cela peut être des ASICs (Application-Specific Integrated Circuit) ou des FPGAs (Field Programmable Gate Array). Des exemples de tels systèmes se retrouvent dans des instruments médicaux, des véhicules, des réseaux automatisés et des systèmes de communications.

Il existe plusieurs avantages à combiner le développement matériel et logiciel dans une méthodologie commune [Axel97]. D'abord, des spécifications détaillées de haut niveau du comportement complet sont prises avant la sélection de l'architecture et du partitionnement. Donc, plus d'information est disponible au départ et les décisions

cruciales d'implantation peuvent alors être prisesfaites avec une plus grande précision. Les décisions importantes qui influencent la qualité du système se prennent généralement au début du processus de conception. La possibilité d'avoir déjà l'information en main diminue la probabilité de faire des erreurs coûteuses. De plus, une description uniforme des modules matériels et logiciels permet de déplacer des parties du système à n'importe quel stade de son développement. Des changements dynamiques sont alors possibles entre les différentes parties implantées en logiciel et celles implantées en matériel. Un autre avantage est que cela permet de faire une évaluation rapidement, à l'intérieur du processus de conception, des importantes caractéristiques du système. Cela évite de découvrir à un stade avancé de la conception de graves problèmes. Et finalement, une méthodologie de codesign est la meilleure façon de rapprocher le développement de logiciel et de matériel, réduisant ainsi les coûts finaux d'intégration entre les différents domaines technologiques.

Les avantages que nous venons de mentionner permettent de diminuer le temps et les coûts de l'implantation en plus d'en améliorer la qualité. Par exemple, Carolyn Kuttner [Kutt96] a estimé pour l'implantation d'une FFT dans un système embarqué, une réduction des coûts de 50% et de 35% pour le temps de conception de l'étude de cas, en utilisant une méthodologie de codesign matériel/logiciel.

Il est possible pour un concepteur de déterminer lui-même la façon d'implanter les différentes parties du système. C'est-à-dire, décider sur quels composants seront implantées les différentes parties du système. Mais, à cause de la complexité des systèmes d'aujourd'hui, il est de plus en plus difficile pour un concepteur de déterminer un partitionnement matériel/logiciel conduisant à des performances optimales.

Les concepteurs de systèmes embarqués ont donc présentement très peu d'aide lors des différentes tâches de conception d'un système. Plus précisément, il n'existe

aucune méthodologie largement répandue et adoptée comme norme ou d'outils disponibles commercialement pour aider les concepteurs à créer une spécification fonctionnelle et la transformer en une architecture au niveau système [GaVa95]. Cette lacune provient des multiples dimensions que peut prendre ce genre de conception. Chaque méthodologie développée peut avoir différentes hypothèses, comme par exemples des choix de composants (matériels ou logiciels) fixes pour certaines fonctionnalités, différents types de contraintes, différents critères lors du partitionnement, etc [AdTh96]. Par conséquent, une méthodologie est nécessaire pour arriver à vérifier différentes possibilités d'implantation d'une manière optimale et trouver un juste équilibre entre le matériel et le logiciel.

Différentes universités font de la recherche dans le développement d'outils pour l'automatisation de la conception de systèmes embarqués. Voici une brève énumération des outils développés auxquels certaines références seront faites ultérieurement: Chinook [Cob95][OrBo98], Comet [KnPa96], Cool [NiMa98], Cosmos [DMVJ97][DIMJ97], Cosyma [EHB93] [HHE94], Cosyn [HeSa97], Lycos [User99], Specsyn [GaVa95], Polis et Ptolemy [poli99] et finalement Vulcan [GuDe93][GCM92][CLG96].

1.2 But du projet de recherche

Le but de notre projet de recherche est de développer et de valider une méthodologie de codesign. En fait, nous allons nous concentrer davantage sur le partitionnement matériel/logiciel à l'intérieur de la méthodologie. Nous devons déterminer en étudiant une application particulière, son partitionnement optimal.

Pour y arriver différents éléments doivent être étudiés. Tout d'abord, nous devons déterminer quelles sont les contraintes du système étudié. Dans notre cas, qui est l'évaluation d'une application de communications à haute vitesse, les contraintes étudiées sont les **temps d'exécution**, les **temps de communications**, la **surface du matériel** et la **dissipation de puissance**. Ces contraintes doivent être évaluées à partir d'une description de haut niveau. Chacune de ces contraintes est évaluée, sauf pour la surface du matériel, du côté matériel ainsi que du côté logiciel. Ces différentes métriques doivent être ensuite fusionnées en une seule valeur qui estimera la qualité d'un modèle. Un algorithme de partitionnement est également utilisé pour déterminer les différents modèles à évaluer. Les différents éléments d'une méthodologie de codesign matériel/logiciel seront développés d'une façon plus détaillée dans le prochain chapitre.

Nous voulions également déterminer le potentiel des outils commerciaux pour l'estimation de gros circuits. Les outils de codesign développés par d'autres université antérieurement n'intégraient pas d'outils commerciaux pour faire des estimations. Nous voulons donc examiner jusqu'à quel point les outils existant peuvent être intégrés dans une méthodologie de codesign et être utilisés comme outils d'estimation. Nous avons principalement utilisé les outils de Mentor Graphics et de Synopsys dans notre méthodologie. De plus, pour les métriques ne pouvant être évaluées par des outils commerciaux, nous devons trouver et valider des techniques d'estimation.

Un autre critère que nous devons considérer est la possibilité d'utiliser cette méthodologie pour d'autres types d'application. Donc, même si des métriques sont éliminées ou ajoutées, nous devons pouvoir continuer à utiliser notre méthodologie.

Finalement, nous voulons évaluer les possibilités d'implanter les modems de la famille xDSL en utilisant une méthodologie de codesign. Nous voulons également déterminer s'il est avantageux pour ce type d'application d'utiliser une méthodologie de codesign matériel/logiciel.

1.3 Application utilisée pour valider la méthodologie

Pour valider notre méthodologie, nous avons décidé d'utiliser la partie numérique de trois modems de la famille xDSL, c'est-à-dire le ADSL (Asymmetric Digital Subscriber Line), le Universal ADSL qui est une version légère du ADSL et finalement le VDSL (Very High Bit-Rate Digital Suscriber Line) qui est une version plus performante du ADSL. Pour valider notre méthodologie, une modélisation des trois modems a été faite. Une description comportementale des différents blocs composant les modems a d'abord été faite en langage C. Cette modélisation nous permet alors, d'évaluer les aspects logiciels de nos modems. Puis, cette description comportementale a été reprise en VHDL, afin d'évaluer les aspects matériels de nos modems.

Les critères que nous évaluons pour les modems sont les temps d'exécution, les temps de communications, la surface du matériel et finalement la dissipation de puissance. Le chapitre 3 contient une explication plus détaillée des trois modems, de leurs contraintes et de leurs modélisations.

1.4 Plan du mémoire

Dans ce premier chapitre, une introduction au projet de recherche a été faite. Nous allons alors voir d'une manière plus détaillée les différents éléments présentés dans ce chapitre. Le chapitre 2 contient une description détaillée des éléments d'une méthodologie de codesign. Cela comprend les différentes métriques utilisées pour évaluer les modems, la fonction objectif regroupant ces métriques et l'algorithme de partitionnement utilisé pour déterminer les différents modèles à évaluer. Le chapitre 3 présente les trois modems que nous avons utilisés pour valider notre méthodologie. Le chapitre 4 explique les techniques utilisées pour estimer les aspects décrits dans le chapitre 2, ainsi que les résultats obtenus. Et finalement, le chapitre 5 contient les conclusions du projet.

Chapitre 2
Le codesign matériel/logiciel

Nous allons pour débuter présenter une méthodologie de codesign dans son ensemble. Puis nous irons plus en détails dans les parties qui nous intéressent. En particulier dans le développement de notre méthodologie pour le partitionnement matériel/logiciel.

2.1 Les différentes étapes du codesign

Chacune des activités de conception doit être faite en fonction du partitionnement logiciel et matériel. Que ce soit dans les spécifications du système, ou dans les simulations, tous les composants sont utilisés. Vous trouverez à la figure 2.1 les activités de conception logiciel/matériel [AdTh96].

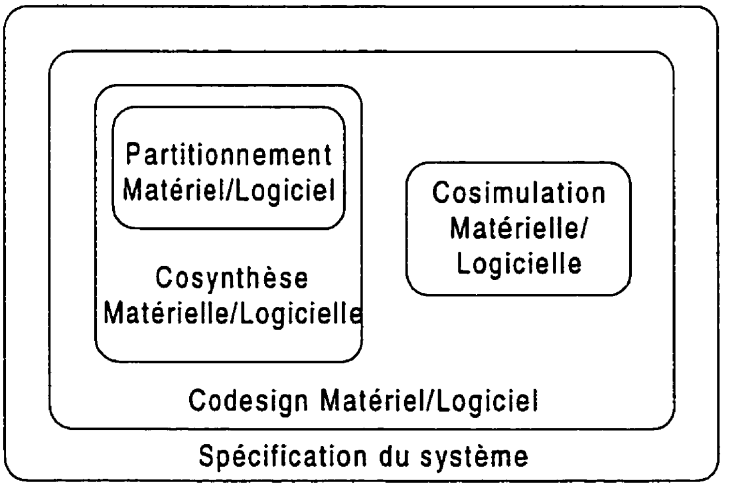


Figure 2.1 Activités de conception matérielle/logicielle

Voici une brève description de ces différentes activités.

- Spécification du système: Une description du comportement ainsi que des contraintes (temps d'exécution, surface, dissipation de puissance, etc...) pour le système hétérogène au complet sont données. Cela peut être fait avec un programme dans un langage quelconque (C, C++, ADA, Java [YMS+98]), des ensembles d'instructions, des graphes de flots de données ou des langages spécifiques de description matérielle (VHDL, HardwareC, Lustre, Esterel, Verilog, Sillage) [GaRa94].

  Les gens ont souvent tendance à choisir un langage de programmation couramment utilisé, car il a l'avantage d'être déjà connu [GuLi97]. Cela rend également possible la réutilisation de programmes déjà écrits. De plus, il est intéressant d'avoir une description qui peut être exécutable, car cela permet de vérifier si la fonctionnalité du système est bonne [DRG97].

  Il y a plusieurs façons de représenter un système et cela à différents stades du processus de conception. Il peut y avoir des représentations comportementale, structurelle, physique et hybride [GVNG94]. Nous allons nous concentrer sur les descriptions comportementales, qui sont à un plus haut niveau d'abstraction. Une description comportementale d'un système ne contient pas d'informations structurelles tels le type de composants, leurs interconnexions, le nombre de pipelines et les phases de l'horloge [GaRa94]. C'est-à-dire que le comportement du système est décrit, mais il n'y a rien de mentionné au sujet de son implantation.

- Codesign logiciel/matériel: C'est dans cette étape que le système est partitionné et que ses différentes parties sont implantées soit en logiciel, soit en matériel. Puis le système est simulé pour en vérifier la fonctionnalité et le respect de ses

contraintes. Les différentes parties du codesign logiciel/matériel sont expliquées plus en détail dans les rubriques suivantes :

- *Partitionnement logiciel/matériel*: Dans cette partie de la conception, un algorithme est utilisé pour déterminer les parties du comportement qui seront implantées en logiciel et celles implantées en matériel. Un partitionnement particulier est appelé un modèle et consiste à une des représentations possibles du système. Un modèle est un prototype virtuel d'un système à évaluer. Il n'est pas implanté, mais contient toutes les informations pour le faire.

  Afin d'utiliser un algorithme, les différentes parties du comportement doivent être décomposées en blocs. Plusieurs facteurs peuvent influencer la façon dont le partitionnement entre le logiciel et le matériel sera fait [Axel97] :

  - Les performances globales du système.
  - Le coût de l'implantation du système.
  - La facilité à modifier les parties logicielles ou matérielles (avec par exemple des composants matériels interchangeables) du système.
  - La nature des calculs, c'est-à-dire la façon dont le système traite les données.
  - Le parallélisme entre les composants.
  - Le temps de communications entre les composants.

- *Cosynthèse logicielle/matérielle:* En général, un outil de synthèse est utilisé pour ajouter des détails au niveau de l'implantation du système décrit avec

un plus haut niveau d'abstraction. Un compilateur peut être utilisé pour générer le langage machine du côté logiciel et un outil de synthèse de haut niveau pour obtenir les ASICs ou FPGAs du côté matériel à partir de la description comportementale.

- *Cosimulation logicielle/matérielle*: Une fois la synthèse du système faite, il est possible de vérifier le comportement de celui-ci en simulant la partie matérielle en même temps que la partie logicielle.

Nous allons maintenant examiner plus en détails une des activités de conception matérielle/logicielle, le partitionnement.

2.2 Les différents composants du partitionnement

Un bon partitionnement matériel/logiciel dépend de différents critères. Ces différents éléments interagissent les uns avec les autres influençant grandement les résultats qui seront obtenus. Par exemple, même si l'algorithme de partitionnement est bon, si les estimateurs ne sont pas précis, nous n'obtiendrons pas un système conçu de façon optimale. Vous pouvez voir les liens entre les différents éléments du partitionnement à la figure 2.2 [GaVa95]. La figure sera expliquée d'une manière plus détaillée dans les prochaines sous-sections.

2.2.1 Modèle d'estimation

Avant d'élaborer un modèle et de l'évaluer, il est important de déterminer la granularité de notre système. La granularité des objets est la dimension des parties qui

devront être implantées en logiciel ou en matériel. On peut appeler ces différentes parties des blocs.

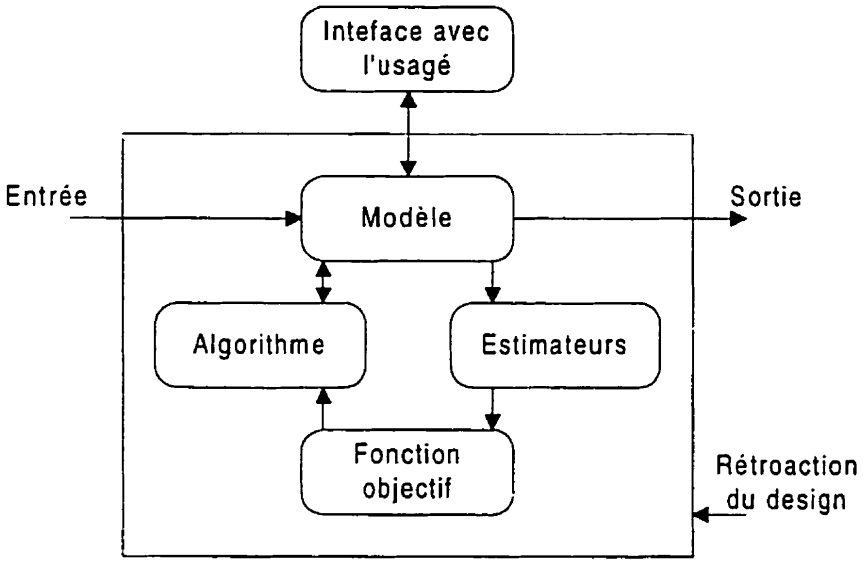


Figure 2.2 Configuration typique d'un système de partitionnement

La taille de ceux-ci est déterminée selon l'application et les contraintes du système. La grosseur d'un bloc peut aller d'un bloc de base qui est en fait la plus petite unité possible, comme une instruction, jusqu'au système complet. Lorsque le partitionnement est à grains fins, c'est-à-dire qu'il y a un grand nombre de petits blocs, les résultats obtenus seront plus précis. Par contre, vu le nombre élevé de blocs à traiter et le grand nombre de combinaisons possibles entre le partitionnement logiciel et matériel, le temps de calcul sera beaucoup plus long. On doit donc trouver un juste milieu entre la taille des blocs et la durée du temps de calcul [HBKG98].

En général, nous allons appeler un partitionnement à grain fin, un partitionnement au niveau des blocs de base ou des instructions comme l'ont fait Ernst, Henkel et Brener dans Cosyma[EHB93] [HHE94], Gupta et De Micheli avec Vulcan [GuDe93], ainsi que Knudson et Madson avec Lycos [User99]. Tandis que Srivastona

et Brodersen [SrBr91] et Athanos et Silverman [AtSi93] ont utilisé une approche à gros grains, ce qui signifie un partitionnement au niveau des tâches ou des processus [HeEr97].

Un autre point à considérer est qu'il est plus avantageux pour le matériel d'avoir de petits blocs. Cela permet un plus grand parallélisme, car plusieurs blocs peuvent être exécutés en même temps. Par contre, du côté logiciel, plus il y a de blocs, plus il y a de changements de contextes. Il est donc avantageux pour une architecture ayant un seul processeur, d'avoir de plus gros blocs de façon à réduire le temps perdu lors des changements de contextes.

La principale question est de savoir comment unir les blocs de base de façon à former les blocs utilisés lors du partitionnement. Pour faire cela, il existe des estimateurs que l'on joint ensemble pour calculer une fonction de proximité. La fonction de proximité peut être formée de différents critères et est utilisée pour déterminer les blocs à unir de façon à atteindre un meilleur partitionnement. Par exemple, il sera plus profitable d'unir deux blocs qui ont une grande quantité de communications de l'un à l'autre [VaGa95a][VaGa95b][BGN97]. Après avoir déterminé les différents blocs, il est possible de passer au partitionnement et d'obtenir un modèle.

2.2.2 Algorithme de partitionnement

Dans la recherche du modèle qui satisfera les contraintes de façon optimale, plusieurs modèles devront être considérés. Il est donc important d'avoir un algorithme qui effectuera des itérations sur les modèles de façon à explorer différentes possibilités. Pour un système de n blocs pouvant être placés sur k éléments différents, il y a k^n possibilités de partitionnement. Si le nombre de blocs n'est pas très élevé, il est

possible d'utiliser un algorithme qui trouvera le meilleur partitionnement avec certitude. Par contre, s'il est élevé, un heuristique ne trouvant pas nécessairement le meilleur partitionnement, mais qui s'exécute dans un plus court laps de temps peut être utilisé. Un compromis doit donc être fait entre le temps d'exécution de l'algorithme et la qualité du partitionnement trouvé. Les différents modèles obtenus avec l'algorithme de partitionnement seront ensuite évalués à l'aide d'estimateurs.

2.2.3 Estimateurs

Certains critères de performance doivent être considérés pour un système embarqué. Souvent ces critères sont en conflits les uns avec les autres. Par exemple la performance est proportionnelle au coût, toutes choses étant égales par ailleurs. Il est très difficile de diminuer le coût et d'améliorer la performance d'un système en même temps.

Il est important de connaître les valeurs de ces différents critères de performance afin d'obtenir le meilleur système possible. Malheureusement, le temps nécessaire à l'implantation du système en entier pour tous les modèles utilisés est beaucoup trop élevé. Nous devons donc évaluer ces critères de performance à l'aide d'estimateurs [Axel97][GaVa95]. Voici les estimateurs les plus fréquemment utilisés. Ceux qui seront utilisés pour évaluer notre application seront traités avec plus de détails dans les sections 2.3, 2.4 et 2.5.

- Performance: Le but d'un grand nombre de systèmes embarqués est de trouver une implantation dont la performance est suffisante pour rencontrer les contraintes de temps. Cet aspect est très important pour les systèmes en temps réel ou les systèmes réactifs. Les données viennent de l'extérieur à un certain rythme et le système doit être capable de les analyser rapidement.

- Coût: L'utilisation d'ASICs ou de microprocesseurs programmables est dispendieuse. Il est important de garder les coûts les plus bas possible.

- Maintenance: Certaines parties du comportement auront peut-être besoin d'être modifiées lorsque le système sera en opération. Ces parties devront donc être implantées en logiciel de façon à pouvoir être facilement modifiable ou en matériel mais en utilisant un standard permettant facilement la réutilisation des blocs comme VSIA (Virtual Socket Interface Alliance) [CHB+99].

- Distribution: Certaines parties du système peuvent être implantées dans des endroits différents. Par exemple les parties nécessitant des données provenant de capteurs peuvent être placées près de ceux-ci. Donc, les blocs devront être divisés de manière à faire partie du composant dont la location correspond à la provenance des données dont ils ont besoin.

- Consommation de puissance: Certains systèmes embarqués peuvent être portables, utiliser une source d'énergie limitée ou être situés dans un espace limité, sans la possibilité d'y placer des ventilateurs. Dans de tels cas, il est important de diminuer la consommation de puissance.

- Poids: Encore une fois, cet aspect peut être très important pour des systèmes portables. Que ce soit pour un téléphone mobile ou une application aérospatiale, le poids peut être un élément important.

- Tolérance aux fautes: Si un certain système embarqué ne tolère aucune faute, il est important d'ajouter une façon de détecter les défaillances du système. Par exemple, la fonctionnalité pourrait être dupliquée.

2.2.4 Fonction objectif

La fonction objectif utilise les valeurs obtenues des différents estimateurs pour créer une fonction unique qui quantifie la qualité du modèle. La plupart des algorithmes de partitionnement ont besoin d'un nombre unique pour comparer les différents modèles et ainsi garder celui qui rencontre de façon optimale les contraintes du système. Les façons de composer une fonction objectif seront décrites dans la section 2.7.

2.3 Description des estimateurs utilisés

Après avoir survolé rapidement les différents éléments du codesign, nous allons traiter d'une manière plus détaillée les parties que nous allons utiliser dans ce projet. Nous verrons donc les points décrits dans la section 2.2 d'une manière beaucoup plus précise et élaborée. Nous allons d'abord commencer par les estimateurs utilisés, puis la fonction objectif et finalement les algorithmes de partitionnement.

2.4 Estimateurs pour la partie logicielle

Nous allons maintenant voir la façon d'estimer ces différents critères du côté logiciel, puis du côté matériel.

2.4.1 Temps d'exécution

La première chose à vérifier dans un système est sa fonctionnalité. Une fois le comportement vérifié, on doit s'assurer que ce comportement est exécuté dans les bons délais [DRG97].

Pour calculer le temps d'exécution des parties du système implantées en logiciel, nous devons tenir compte du temps d'exécution de chacun des blocs de base et de l'ordonnancement de ces blocs. Pour une implantation logicielle la description du comportement doit être compilée dans un ensemble d'instructions du processeur cible selon un des deux modèles qui seront présentés dans la sous-section 2.4.1.1. On suppose que les variables du comportement sont logées dans la mémoire du processeur. Donc, toutes les variables du comportement sont implantées avec les opérations écriture/lecture en mémoire [GVNG94].

2.4.1.1 Modèles d'estimation

Il y a d'abord un modèle d'estimation avec un processeur spécifique. Le temps d'exécution peut être estimé de façon précise en compilant chacun des blocs dans un ensemble d'instructions pour un processeur particulier, en utilisant un compilateur spécifique à ce processeur. Une estimation à partir de ce modèle est très précise. Par contre, elle prend beaucoup de temps de calcul. De plus, l'estimation complète devra être faite pour chacun des processeurs faisant partie de la bibliothèque des composants avec des compilateurs différents.

Au lieu d'utiliser différents compilateurs et estimateurs, un modèle alternatif générique a été suggéré par Gajsky et al. [GVNG94]. Knieser et Papachristou l'utilisent également dans leur outil de conception pour le codesign Comet [KnPa96]. Le comportement est d'abord compilé dans un ensemble d'instructions génériques. Pour chaque processeur, un fichier technologique est disponible, contenant les informations au sujet du nombre de périodes d'horloge et le nombre d'octets de chacune des instructions génériques requises. Les estimations sont donc faites à partir des instructions génériques et des fichiers technologiques pour le processeur cible.

2.4.1.2 Profilage des blocs du logiciel

Il existe deux façons de déterminer le temps d'exécution d'un programme. Il y a d'abord une façon dynamique[GVNG94]. Une simulation dynamique exécute plusieurs fois les blocs du programme avec différentes données et enregistre le nombre de périodes d'horloge requises pour chacune des exécutions. Tout dépendant des données, le nombre de périodes d'horloge peut différer selon les branchements conditionnels et les boucles dépendantes des données. L'estimation dynamique demande un long temps de calcul, car l'exécution du programme doit être faite plusieurs fois. Il existe des outils qui peuvent être utilisés pour accomplir cette tâche comme QPT [Qpt99] et MIPS Pixie [Pixi99]. Ces deux programmes réécrivent le programme source en y ajoutant des instructions pour calculer la fréquence d'exécution de chaque bloc de base ou séquence de blocs de base. Un bloc de base contient des instructions étant exécutées le même nombre de fois et est en général borné par le début et la fin d'une boucle ou d'un branchement conditionnel.

Une autre façon de déterminer le temps d'exécution des blocs d'un programme est d'utiliser une méthode statique. Cette méthode ne tient pas compte des données utilisées et peut atteindre de très bons résultats si le nombre d'itérations des boucles est connu et que la probabilité des branchements conditionnels peut être prédite de façon précise. Pour le profilage statique, il existe un outil appelé Cinderella [Cind99], qui peut être utilisé pour profiler des programmes s'exécutant sur un processeur i960 d'Intel et les processeurs de la famille M68000 de Motorolla. Cinderella lit le code exécutable des programmes et construit en séparant le programmes en blocs de base un graphe de flots de contrôle (CFG) et en déduit les contraintes structurelles. L'utilisateur doit fournir les bornes supérieures et inférieures de toutes les boucles. En utilisant les bornes supérieures des boucles, il est possible de déterminer le chemin le plus long dans

l'exécution du programme. On peut ensuite trouver le temps d'exécution maximal en ordonnançant les différents blocs de base.

2.4.1.3 Ordonnancement

Le problème d'ordonnancement consiste à décider l'ordre et le temps d'exécution d'un ensemble de blocs de base ayant certaines caractéristiques connues par le concepteur (la durée déterminée avec le modèle, puis le nombre d'exécutions déterminé lors du profilage) [BLMS98] [OBE99]. L'ordonnancement est en général fait par le système d'exploitation utilisé par le processeur.

Vérifier si un ordonnancement permet de rencontrer les contraintes de temps est difficile à déterminer, même si le temps d'exécution des blocs est connu. Il existe différentes techniques d'ordonnancement pour le logiciel. Ces techniques sont en général statiques ou dynamiques. L'ordonnancement est dit statique lorsque les différentes tâches s'exécutent dans un ordre fixe et prédéterminé. Un ordonnancement statique ne demande pratiquement aucun temps CPU lors de l'exécution, ce qui implique un faible temps de service. Un ordonnancement statique est optimal lorsque les temps où les différentes tâches sont prêtes à être exécutées est bien connu à l'avance.

Dans certains cas, un ordonnancement statique n'arrive pas à rencontrer les contraintes de temps, alors un ordonnancement dynamique est nécessaire. Une exécution dynamique est basée sur la priorité. Les tâches prêtes à être exécutées sont choisies dynamiquement selon leur ordre de priorité. La priorité est la mesure de l'urgence avec laquelle une tâche doit être exécutée. De plus, une tâche peut être préemptive, c'est-à-dire qu'elle peut être suspendue lorsqu'une tâche qui a une priorité plus élevée est prête à être exécutée. Le principal désavantage est le temps de service

nécessaire à l'ordonnancement. Celui-ci est très élevé, car à chaque changement de contexte, des calculs élaborés doivent être faits, augmentant de façon significative le temps d'exécution [Axel97].

2.4.2 Dissipation de puissance

L'estimation de la puissance pour un système matériel et logiciel est de plus en plus importante [HeLi98]. Lorsque l'espace est limité ou qu'il n'y a pas de possibilité d'y placer un ventilateur, cet aspect de la conception doit être pris en ligne de compte (un téléphone cellulaire avec un ventilateur ne serait pas un très bon produit...).

La dissipation de puissance peut différer de beaucoup, tout dépendant du système implanté. La tension d'alimentation d'une carte n'est pas très représentative en tant que métrique, car les activités à l'intérieur du processeur sont en général associées à ses périphériques. De plus en plus, le processeur n'est qu'une petite partie du système.

Le processeur est fait d'un circuit CMOS. Le changement de voltage sur une capacitance nécessite un transfert de charge, et par conséquent cause une consommation de puissance. Une fois la capacitance chargée, la porte logique peut maintenir un voltage DC sans mouvement de charge additionnel et ne consomme pas de courant. Pour cette raison, la circuitrie CMOS consomme de la puissance seulement en changeant d'état.

Il est important de souligner que la consommation d'énergie d'un microprocesseur dépend de ses stimulus déterminés par le logiciel et les entrées qu'il reçoit. Donc, faire une analyse de la dissipation de puissance sans tenir compte du logiciel impliqué peut mener à des estimations très loin de la réalité. Jusqu'à maintenant, lorsqu'un client veut choisir un microprocesseur pour un système

embarqué, il a seulement la possibilité de connaître une valeur moyenne, ou une échelle allant de la plus petite valeur à la plus grande, mais sans connaître le type d'application sur lequel ces valeurs ont été calculées [STA98].

Il existe trois principales façons de calculer la dissipation de puissance et nous allons les traiter dans ce document. Ces méthodes sont: le profilage, l'estimation à partir d'un modèle matériel et l'estimation à partir des instructions.

2.4.2.1 Profilage

Il est possible d'utiliser une approche montante (bottom-up) pour déterminer la dissipation de puissance d'un processeur. On peut le faire sous forme de profilage. Il s'agit d'exécuter un programme plusieurs fois et à chaque fois en calculer la dissipation de puissance. Pour obtenir la dissipation de puissance, plusieurs techniques peuvent être utilisées. Il est possible de recouvrir le processeur et de mesurer la chaleur qui s'en dégage. Mais, une façon plus simple de procéder est de mesurer le courant utilisé (I_{LOG}) à l'entrée du processeur lors de l'exécution du programme, pour ensuite en utilisant le voltage (V_{DD}) fourni au processeur en déduire la puissance utilisée (P_{LOG}), en utilisant la formule $P_{LOG} = I_{LOG} * V_{DD}$.

2.4.2.2 En utilisant un modèle matériel

Il est possible également d'utiliser un modèle matériel du processeur pour estimer la dissipation de puissance de la partie logicielle. Pour un programme relativement simple, une façon de calculer la consommation de puissance est d'exécuter le programme sur un modèle comportemental ou RTL (Register Transfer Level) du

système et de mesurer la consommation de puissance en utilisant des techniques pour l'estimation de puissance au niveau du matériel.

Par contre, une application complexe consomme des millions de cycles machine, ce qui le rend très difficile à estimer en utilisant des techniques d'estimation pour le matériel au niveau comportemental ou RTL. Mais Hsieh et al. [MaPe98][HsPe98] ont quand même réussi à présenter une méthode basée sur cette technique, appelée *approche par synthèse de programme* pour calculer la dissipation de puissance d'un processeur. Ils utilisent un modèle transformé de celui-ci et appliquent une méthode d'évaluation de dissipation de puissance pour le matériel au niveau RTL. Les auteurs synthétisent un programme d'une telle façon que la trace résultante en terme de performance et de dissipation de puissance soit équivalente à la trace du programme originale. La nouvelle trace d'instructions est par contre beaucoup plus courte que la trace du programme originale et peut donc être simulée plus facilement par une description matérielle du processeur et donner des résultats rapidement au sujet de la dissipation de puissance.

Pour utiliser ce type d'estimation, le processeur doit être bien connu au niveau comportemental ou RTL. Souvent cette information n'est pas disponible et une autre technique doit être examinée.

2.4.2.3 Modèles de puissance au niveau des instructions

Une alternative populaire à la simulation d'un modèle matériel du processeur est d'utiliser des modèles de puissance au niveau des instructions. Ces modèles sont reliés à la consommation de puissance d'un processeur pour chaque instruction ou séquence d'instructions qu'il peut exécuter. Des raffinements supplémentaires sont faits à cette

évaluation en utilisant des statistiques tels que le pourcentage de fautes de la mémoire cache, le taux de blocage des pipelines, etc.

L'environnement de conception assistée par ordinateur pour le co-design appelé TOSCA [FGS+98] utilise cette méthode pour calculer la métrique de puissance au niveau logiciel servant à guider le partitionnement du système. Nous allons nous attarder plus longuement à cette technique, car elle nous apparaît la plus intéressante. Un compilateur ou un simulateur d'ensemble d'instructions pour un processeur cible pourraient être améliorés en utilisant un tel modèle[LRD+99].

Pour certains processeurs, il est possible de calculer la dissipation de puissance en multipliant le nombre de cycles du programme par une valeur moyenne de dissipation de puissance par cycle. Russell et al. [RuJa99] ont étudié un processeur RISC embarqué de 32 bits, le i960 de Intel. Ce modèle de processeur n'est pas très complexe et il a été démontré par les auteurs qu'ils pouvaient prédire la consommation d'énergie à l'intérieur de 8% dans 99% des cas s'ils comparaient leurs estimations à des mesures physiques. Leurs conclusions démontrent que les variations de consommation de puissance à travers les instructions assembleurs n'ont aucune signification statistique pour le processeur i960. En fait, il n'y a aucun besoin de considérer les instructions assembleurs individuelles pour prédire la consommation d'énergie réelle correctement. La puissance fluctue pendant un cycle d'horloge, mais sa moyenne est stable. Les registres sources et destinations, ainsi que le code conditionnel n'ont aucun influence significative. Cette façon est rapide et facile, mais elle ne s'applique malheureusement pas à la plupart des processeurs.

Dans [TMW94] et [TTMF97] en se basant sur les mesures réelles de quelques processeurs, Tiwari et Al. présentent le modèle suivant de dissipation de puissance au niveau des instructions:

$$Energie_p = \sum_i (BC_i N_i) + \sum_{ij} (SC_{i,j} N_{ij}) + \sum_k OC_k, \qquad (2.1)$$

où $Energie_p$ est la dissipation d'énergie totale du programme qui est divisé en trois parties. La première partie est l'addition du coût de l'énergie de base de chaque instruction ($(BC_i$ est le coût de l'énergie de base et N_i est le nombre de fois où l'instruction est exécutée). La deuxième partie de l'équation donne l'état du circuit ($SC_{i,j}$ est le coût d'énergie quand l'instruction i est suivie par j pendant l'exécution du programme et N_{ij} et le nombre de fois où ces instructions se suivent). La troisième partie de l'équation représente la contribution en énergie des autres effets d'instruction (Oc_k), tels les fautes de la mémoire cache et le blocage des pipelines pendant l'exécution du programme

Pour mesurer le coût de base d'une instruction, la même instruction ou un bloc d'instructions est exécuté plusieurs fois dans une boucle infinie. La puissance moyenne est ensuite mesurée pour plusieurs itérations de la boucle du programme. Basée sur ces essais, une estimation de la consommation de puissance moyenne est trouvée pour chacune des instructions. Puis, une table de puissance peut être pour chaque processeur, en rapportant la consommation d'énergie pour chaque instruction comprise dans l'ensemble d'instructions et pour tous les modes d'adressages associés à chaque type d'instruction. Il existe des moyens de minimiser la taille de ces tables en les plaçant dans des classes. Par exemple, si on prend les coûts de base des instructions, les instructions ayant des consommations de puissance similaires peuvent être placées dans la même classe. En utilisant cette méthode, Chunho et al. [CMP99] ont divisé les instructions d'un processeur DSP Fujitsu en six différentes classes.

En général, la puissance moyenne dissipée par un processeur en exécutant un programme est $P_{LOG} = I_{MOY} * V_{DD}$, où I_{MOY} est le courant moyen et V_{DD} est la tension

d'alimentation fournie au processeur. L'énergie est donnée par $E_{LOG} = P_{LOG} * t_{LOG}$ où t_{LOG} est le temps d'exécution du programme sur le processeur qui est exprimé par: $t_{LOG} = N_{HOR} * \tau_{HOR}$, où N_{HOR} est le nombre de cycles de l'horloge pour exécuter le programme et t_{HOR} la période de l'horloge. Pour calculer le courant moyen utilisé pendant l'exécution de chaque instruction, il est nécessaire de prendre des mesures du coût en énergie pour chacune des instructions.

Pour certains processeurs, il est important de tenir compte des effets des instructions précédentes. Les contributions supplémentaires au calcul de la dissipation de puissance globale proviennent des effets inter-instructions, ce qui n'a pas été considéré dans le calcul du coût de base de chaque instruction. L'activité de permutation dans un circuit est en fonction de l'entrée présente et de l'état précédent du processeur. Donc, le coût en énergie d'une instruction peut être un peu différent de son coût de base. Cela est dû au fait que lors du calcul des coûts de base, l'état précédent du processeur pouvait ne pas être le même. Pour certains processeurs, il est possible de considérer cet effet inter-instructions comme étant constant sans affecter la qualité de l'estimation. Mais, dans d'autres cas, c'est un élément qui doit être considéré et qui peut parfois être difficile à faire. Pour réussir à bien comprendre les effets inter-instructions, l'architecture interne du processeur, ainsi que la façon dont les instructions sont microprogrammées, doivent être connus.

Les données utilisées pour chacune des instructions peuvent également influencer la dissipation de puissance d'une instruction. Selon [TTMF97][TTMF95], les données n'ont pas tellement d'influence si l'on utilise un processeur général. Par contre, lorsqu'un processeur DSP est utilisé, les données peuvent influencer la dissipation de puissance d'une manière assez significative pour devoir en tenir compte. L'équation doit donc être modifiée pour refléter cette réalité.

Pour d'autres processeurs, il est important de tenir compte des effets des contraintes de ressources et des fautes de cache sur le budget de puissance. Par contre, ces effets peuvent être négligés dans les programmes exécutés par certains processeurs simples comme le M68000, Intel 805, Z80, etc... où des caractéristiques aussi avancées peuvent ne pas exister et dans des processeurs plus sophistiqués où le taux de succès des caches peut atteindre 98% et procurer une exploitation maximale des différents niveaux du pipeline.

Il est important de mentionner que même si le processeur possède un pipeline, cette méthodologie tient toujours. La preuve a été faite par Tiwari et al. dans [TMW94] pour un processeur 486DX2CPU. Prenons Ej_{Ik} qui est l'énergie moyenne consommée par l'étape j du pipeline, lorsque l'instruction Ik s'exécute dans cet étape. Les étapes du pipeline sont séparés les uns des autres par des bascules. Donc, les consommations d'énergie des différentes étapes sont indépendantes les unes des autres. Prenons par exemple à un cycle donné, l'instruction $I1$ est exécutée par l'étape 1, $I2$ par l'étape 2 et ainsi de suite. L'énergie totale consommée par le processeur dans ce cycle sera :

$$E_{cycle} = E1_{I1} + E2_{I2} + E3_{I3} + E4_{I4} + E5_{I5}. \tag{2.2}$$

D'un autre côté, l'énergie totale consommée par l'instruction $I1$ lorsqu'elle se déplace à travers les étapes du pipeline est $E_{ins} = \Sigma_j Ej_{I1}$. Donc, en plaçant l'instruction $I1$ qui est à évaluer, dans une boucle d'instructions, cela donne comme résultat

$$E_{cycle} = E_{ins}, \tag{2.3}$$

puisque dans ce cas $I_1 = I_2 = I_3 = I_4 = I_5$. E_{cycle} représente l'énergie d'un cycle, c'est-à-dire des différentes étapes du pipeline et E_{ins} l'énergie d'une instruction. Dans ce cas, le courant moyen est :

$$\Sigma_j Ej_{I1}/(V_{CC} * temps), \tag{2.4}$$

ce qui a été vérifié expérimentalement par les auteurs.

Une fois l'analyse de puissance complétée pour toutes les instructions, elle est étendue à tous les modules du logiciel, en calculant la consommation de puissance selon leur fréquence d'appel[FGS+98].

2.4.2.4 Autres techniques

Finalement, d'autres méthodes ont été brièvement mentionnées dans [MaPe98]. La première méthode estime la dissipation de puissance en utilisant la capacitance moyenne lorsque les modules du processeur, sont utilisés. C'est-à-dire que chaque fois qu'un module est utilisé, on ajoute sa dissipation de puissance à la puissance totale dissipée. Les activités de changement sur les bus (adresse, instruction, données) peuvent également être utilisées pour calculer la consommation de puissance d'un processeur.

2.5 Estimateurs pour la partie matérielle

Nous allons maintenant voir les différentes métriques utilisées pour évaluer le matériel.

2.5.1 Surface du circuit intégré et temps d'exécution

Un autre point majeur dans le développement d'un système est son coût. Le principal composant de ce coût est la partie implantée en matériel. Le processeur commercial utilisé coûtera le même prix quel que soit le nombre de blocs implantés sur

celui-ci en logiciel. Par contre le nombre de blocs implantés en matériel a un lien étroit avec la quantité de matériel nécessaire à sa réalisation. Il est donc important d'être capable d'évaluer la quantité de matériel nécessaire afin d'être en mesure d'estimer le coût d'un système.

Pour déterminer le temps d'exécution, ainsi que la surface du circuit intégré utilisé, nous allons prendre une technique de synthèse. Une synthèse rapide est utilisée, en omettant des tâches qui consomment beaucoup de temps, comme par exemple les optimisations de la logique [VaGa95c].

Une synthèse de haut niveau utilise une séquence de tâches qui transforme une représentation comportementale en un modèle de bas niveau en utilisant des unités fonctionnelles comme des unités arithmétiques et logiques, ainsi que des unités d'entreposage des données et d'interconnexion [GaRa94]. Cela devient en fait la représentation interne du système.

Une synthèse est effectuée en trois étapes. Il y a d'abord l'allocation, puis l'ordonnancement et finalement le partage des ressources (binding) [GaRa94]. Pour estimer la surface et le temps d'exécution, nous allons suivre les trois étapes de la synthèse qui seront décrites plus en détails dans les prochaines sections. Mais, nous allons tout d'abord présenter les façons de représenter le comportement d'un bloc pour son évaluation matérielle.

2.5.1.1 Représentation du comportement

En général, la meilleure façon de décrire le comportement d'un système est à l'aide d'un graphe de flots de données, car il permet de représenter les opérations arithmétiques et les dépendances de lecture et d'écriture qui définissent l'ordre

d'exécution. Donc, avant d'en faire la synthèse, une description matérielle, faite par exemple en VHDL, doit être transformée en graphe de flots de données ou en quelque chose d'équivalent.

Un graphe de flots de données possède des nœuds *v* et des liens *e* entre les différents blocs. Le comportement est divisé en nœuds et les dépendances entre les nœuds sont représentées par des liens. Par exemple, le lien e_{ij} relie le nœud *i* au nœud *j*, comme nous pouvons le voir à la figure 2.3.

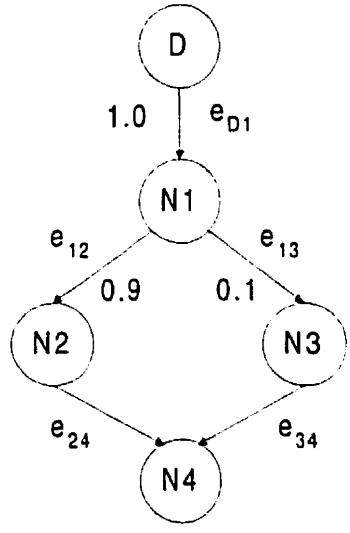


Figure 2.3 Graphe de flots de données

Il existe cinq différents types de nœuds de base : les nœuds de données, les nœuds conditionnels, les nœuds hiérarchiques, les nœuds d'emplacement et les nœuds de contrôle de boucle. Les nœuds de données représentent les opérations arithmétique et logique. Les nœuds conditionnels correspondent aux *if* et aux *case*. Les nœuds hiérarchiques contiennent les autres nœuds et liens qui correspondent aux boucles, aux fonctions et aux procédures. Les nœuds d'emplacement représentent parfois les sauts avant dans un graphe d'état ou simplement une connexion sur le graphe. Finalement les nœuds de contrôle de boucle dénotent les limites de la boucle [Knap96]. Lorsque nous

utilisons des nœuds conditionnels, une valeur représentant la probabilité $prob(e_{ij})$ doit être ajoutée aux différents liens sortant de ce nœud. Cette probabilité mesure le nombre de fois où le branchement est pris. C'est en fait une valeur entre 0 et 1. Par exemple, si une fois sur dix, le lien entre i et j est utilisé, alors ce lien aura une probabilité de 0,1.

2.5.1.2 Allocation

La tâche d'allocation détermine le type et la quantité de ressources utilisées par le système. Elle détermine également la période de l'horloge, ainsi que les pipelines. À l'intérieur de sa description comportementale, le concepteur peut également déterminer les bouts de code qui seront pipelinés ou les boucles qui seront ou non déroulées. Monet [Mone98], lit un système décrit en VHDL ou Verilog et le transforme en une base de données appelée « SIF » (Synthesis Internal Form). Les opérations du système, comme les «+» et les «-» sont remplacées par des opérateurs dans le SIF. Les opérateurs standards sont définis dans une bibliothèque [Mone98].

Dans certains outils de synthèse existant (c'est le cas avec *Monet* de Mentor Graphics et le *Behavioral Compiler* de Synopsys), la période d'horloge doit être spécifiée par le concepteur avant la synthèse. Dans d'autres outils, lorsque celle-ci n'est pas spécifiée, nous avons alors besoin d'en estimer une [GVNG94]. Il existe différentes façons de faire cette estimation et nous allons en présenter deux.

La première méthode consiste à utiliser le bloc qui a le plus long temps d'exécution pour estimer l'horloge. Le délai de chacun des blocs doit alors être estimé et celui qui a le plus grand délai est celui qui détermine la période de l'horloge. Cette méthode peut être efficace, si les temps d'exécution des blocs sont d'une longueur similaire. Ce n'est pas toujours optimal et une façon de remédier à cette contrainte est de faire une évaluation en tenant compte des marges de l'horloge. La marge de

l'horloge est la différence entre le délai du bloc et celui de l'horloge. La moyenne des marges est calculée pour différentes périodes d'horloge et la période ayant la plus petite moyenne est utilisée.

2.5.1.3 Ordonnancement

La prochaine étape ordonnance les opérations et les accès mémoires à l'intérieur de cycles d'horloge. Les algorithmes d'ordonnancement peuvent être largement classés en étant contraints par le temps ou par les ressources [GaRa94]. Un algorithme basé sur des contraintes de ressources construit habituellement l'ordonnancement un cycle à la fois. Il ordonnance toutes les opérations qui ne dépassent pas les contraintes des ressources ou violent les dépendances de données. Dans un ordonnancement basé sur les contraintes de temps, le nombre d'étapes de contrôle est fixe. Il existe également des algorithmes que l'on dit de base. Ces algorithmes peuvent être utilisés en tant que tel, mais ils sont très souvent la base d'un autre algorithme plus complet [Govi99]. Nous avons classé les algorithmes d'ordonnancement en différentes catégories tel que montré à la figure 2.4.

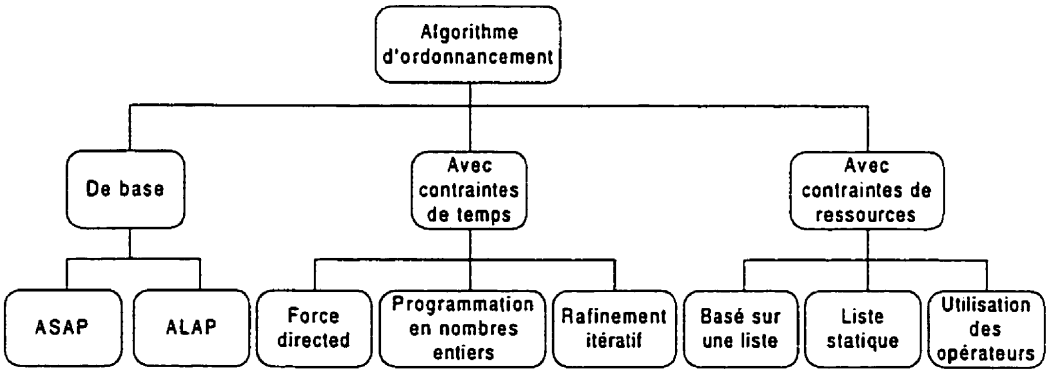


Figure 2.4 Classification des algorithmes d'ordonnancement

2.5.1.4 Les algorithmes d'ordonnancement de base

La plupart des algorithmes que nous allons décrire plus tard nécessitent des bornes pour le temps d'exécution le plus court et le plus long à l'intérieur desquelles ils peuvent être ordonnancés et cela peut être trouvé par les algorithmes ASAP et ALAP respectivement. Ces algorithmes n'ont pas de contraintes de ressources.

ASAP

Une façon simple d'ordonnancer les opérations est l'approche dite « aussi tôt que possible » (ASAP ou As Soon As Possible) [DeMi94]. C'est l'algorithme utilisé dans l'outil de conception pour le codesign Cosyn [HeSa97]. L'algorithme ASAP débute avec le plus haut nœud (qui n'a pas de parents) dans le graphe de flots de données et assigne les étapes de contrôle dans un ordre croissant, alors qu'il procède en descendant dans le graphe de flots de données. La complexité de l'algorithme est $O(V+E)$ où V est le nombre de nœuds dans le graphe de flots de données et E représente le nombre de liens dans le graphe.

ALAP

Cette approche est un raffinement de l'ordonnancement ASAP. Les opérations sont ordonnancées « aussi tard que possible » (ALAP ou As Late As Possible) en tenant compte d'un temps maximal [DeMi94]. L'algorithme ALAP fonctionne de la même façon que l'algorithme ASAP, à l'exception qu'il commence au bas du graphe de flots de données et qu'il procède vers le haut. L'algorithme donne l'ordonnancement le plus lent possible qui prend le nombre maximal d'étape de contrôle. La complexité de cet algorithme est $O(V+E)$ où V est le nombre de nœuds dans le graphe de flots de données et E représente le nombre de liens dans le graphe.

Avec contraintes de temps

Les algorithmes d'ordonnancement avec contraintes de temps sont importants pour des systèmes en temps réel, comme le traitement de signaux numériques où l'objectif principal est de minimiser le coût du matériel tout en rencontrant certaines contraintes de temps.

Dirigé par la force

L'algorithme d'ordonnancement dirigé par la force (FDS) distribue les opérations du même type uniformément selon les étapes de contrôle disponibles. Pour chaque opérateur, la force est calculée, basée sur sa mobilité. La mobilité de l'opération est calculée en utilisant les algorithmes ALAP et ASAP et indique le nombre potentiel d'étapes de contrôle auxquelles peuvent être assignés une opération sans ajouter de délai au temps d'exécution du comportement. La complexité de l'algorithme FDS est $O(CV^2)$ où C est le nombre d'étapes de contrôle et V le nombre de nœuds dans le graphe de flots de données. L'algorithme FDS ne produit pas toujours un ordonnancement optimal.

Programmation en nombres entiers

La programmation en nombres entiers, ILP (Integer Linear Programming) essaie de trouver l'ordonnancement optimal en utilisant un algorithme de recherche du type *branch-and-bound* [DeMi94] [Govi99]. Il contient également du « backtracking », c'est-à-dire qu'une décision prise antérieurement peut être changée par la suite. Une formulation simplifiée de la méthode ILP est donnée. Premièrement, les frontières de la mobilité sont calculées pour chaque opération $M= \{S_j \mid E_k \leq j \leq L_k\}$, où E_k et L_k sont les valeurs provenant des algorithmes ASAP et ALAP respectivement. Le problème d'ordonnancement par ILP peut se définir par l'équation suivante :

$$minimiser \quad \sum_{k=1}^{n} (C_k * N_k) \quad et \quad \sum_{E_i \le j \le L_i} x_{ij} = 1, \quad \forall i, \ 1 \le i \le n, \tag{2.5}$$

où $k \in [1..m]$ types d'opérations sont disponibles, i le nombre d'opérations, N_k est le nombre d'unités fonctionnelles pour les opérations du type k et C_k est le coût de chaque unité fonctionnelle. Chaque $x_{i,j}$ est 1 si l'opération i est assignée à l'étape de contrôle j, sinon sa valeur est zéro. Cet algorithme est précis, mais il a une très grande complexité de calcul.

Raffinement itératif

Dans cet algorithme, un ordonnancement initial est utilisé et chaque opération est déplacée à une étape de contrôle précédente ou subséquente, tout en gardant en mémoire les dépendances de données [Govi99]. Un déplacement au hasard est choisi, l'opération est déplacée, puis verrouillée temporairement dans cette position. De la même façon, d'autres déplacements sont faits jusqu'à ce que toutes les opérations soient verrouillées dans des positions optimales. Lorsque toutes les opérations sont verrouillées, la procédure complète est reprise avec un nouvel ordonnancement.

Avec contraintes de ressources

Dans une application où la conception est restreinte par la surface de silicium, des algorithmes d'ordonnancement avec contraintes de ressources sont très utiles. Le but de ces algorithmes est d'obtenir un ordonnancement dans le moins de temps possible, tout en utilisant un nombre limité de ressources.

Basé sur une liste (list scheduling)

Avec cet algorithme, une liste de priorité est d'abord créée. Elle contient alors toutes les opérations qui n'ont aucun prédécesseur ou dont les prédécesseurs ont été

ordonnancés. Toutes les opérations pouvant être associées à l'étape de contrôle courante sont ordonnancées et éliminées de la liste. Après l'assignation d'une étape de contrôle, la liste de priorité est mise à jour. Dans cette technique, une priorité basée sur la mobilité peut être donnée aux opérations. La complexité de cet algorithme est $O(V+E)$ où V est le nombre de nœuds et E le nombre de liens du graphe de flots de données [GaRa94][Govi99].

Liste statique

Cet algorithme utilise au départ les algorithmes ASAP et ALAP pour obtenir les étapes de contrôle minimale et maximale auxquelles l'opération peut être ordonnancée. Les opérations sont ensuite ordonnées en ordre ascendant en utilisant les valeurs obtenues par l'algorithme ALAP. La valeur de l'algorithme ASAP est utilisée comme deuxième paramètre en cas d'égalité pour la valeur de l'algorithme ALAP. Une fois la liste de priorité crée, les opérations sont ordonnancées séquentiellement, en partant avec la dernière opération dans la liste de priorité (c'est-à-dire celle ayant la plus haute priorité). À chaque itération, quand les limites du nombre des ressources ont été atteintes, le reste des opérations sont retardées à une étape de contrôle ultérieure[Govi99].

Opérateurs utilisés (Operator used)

Dans cette méthode, les énoncés sont placés dans des nœuds. Tous les énoncés d'un même nœud doivent être capables de s'exécuter de façon parallèle. Pour chaque type d'opération d'un nœud, nous calculons le nombre d'occurrences de ce nœud, que nous divisons par le nombre de ressources implantant cette opération. Cette étape est répétée pour chacune des opérations du nœud. On prend le plus grand nombre trouvé pour les opérations du nœud et cela devient le nombre d'étapes de contrôle. Nous répétons les mêmes étapes pour tous les nœuds et faisons l'addition du nombre d'étapes

de contrôle de chacun des nœuds pour trouver le nombre d'étapes de contrôle du comportement en entier. La complexité de cet algorithme est O(V) où V est le nombre de nœuds du graphe de flots de données [GVNG94].

2.5.1.5 Partage des ressources (binding)

La tâche de partage des ressources assigne les opérations et les accès mémoires à chaque cycle d'horloge aux unités matérielles disponibles. Des ressources telles des unités fonctionnelles, des unités d'interconnexions et des unités d'entreposages peuvent être partagées par différentes opérations.

Les registres

Les registres sont utilisés pour entreposer les valeurs des données des constantes, des variables et des tableaux. Une façon très simple d'estimer le nombre de registres nécessaires est d'allouer chacun d'eux à un registre particulier. C'est une façon rapide de faire, mais par contre, cela entraîne un nombre excessif de registres. Certains seront très peu utilisés et peuvent être combinés avec d'autres dans le but d'optimiser leur utilisation.

Pour ce faire, la première chose qui doit être calculée est la durée de vie des variables. En fait, cela commence lors de la définition de la variable et se termine lors de sa dernière utilisation.

Une fois la durée de vie calculée, l'allocation des variables dans un même registre peut se faire de différentes manières. Il existe l'approche du partitionnement par cliques. Toutes les variables sont placées dans des nœuds. Si deux variables ont des durées de vie qui ne se superposent pas, alors un lien est créé entre les deux nœuds. Ces

nœuds sont alors considérés comme étant des nœuds voisins. Les nœuds ayant le plus de voisins communs sont alors fusionnés dans la même variable, formant une clique. À la fin, quand il n'y a plus de fusions possibles, le nombre de cliques correspond au nombre de registres nécessaires à l'implantation de toutes les variables. La complexité de cet algorithme est $O(nc)$ où n est le nombre de variables à placer dans des registres et c le nombre de cliques

Une autre approche pour minimiser le nombre de registres se nomme « left edge ». Une liste de variables comprenant leur durée de vie est créée. Un algorithme est utilisé pour placer les variables de la liste d'une manière ascendante du temps de départ de leur durée de vie. À chaque itération, une variable de la liste est assignée à un registre. D'abord, les registres existants sont examinés et s'il existe un registre dont les durées de vie ne se superposent pas, la variable est assignée à ce registre. Sinon, un nouveau registre est créé pour entreposer la variable. La complexité de cet algorithme est $O(n \log_2 n)$ où n est le nombre de variables à placer dans des registres.

Les unités fonctionnelles

Les unités fonctionnelles sont utilisées pour implanter les opérations du comportement. Il existe encore une fois plusieurs façons d'estimer ce nombre. Premièrement, le concepteur peut spécifier explicitement le nombre d'allocations d'unités fonctionnelles, c'est ce que l'on a vu précédemment, c'est-à-dire les contraintes de ressources. Dans un deuxième cas, si le comportement a déjà été ordonnancé avec des étapes de contrôle, alors une méthode similaire à la méthode de partitionnement par cliques peut être utilisée. De la même façon, un graphe peut être construit pour les différentes opérations et des liens peuvent être créés pour indiquer les nœuds qui n'ont pas une utilisation qui se superpose.

Lorsque toutes les variables et les opérations sont assignés à des registres ou des unités fonctionnelles, il est possible d'estimer le nombre d'unités d'interconnexion.

Unités d'interconnexion

Il peut être surprenant de voir la différence entre les surfaces d'un système si l'on tient compte ou non des unités d'interconnexion. Il est donc important de les ajouter lors du calcul de la surface [MFT+96]. Des unités d'interconnexion sont nécessaires pour relier les différents registres et unités fonctionnelles. Ces connexions peuvent être faites à l'aide d'un bus ou bien de multiplexeurs. Les interconnexions peuvent être estimées directement à partir du comportement, ainsi que de l'emplacement des variables et des unités fonctionnelles. Une façon de faire est d'implanter toutes les connexions qui vont au même endroit sur un même bus ou multiplexeur. La taille des multiplexeurs peut être factorisée en différents multiplexeurs et en insérant un autre multiplexeur pour sélectionner une des entrées communes.

Une autre approche est d'estimer le nombre de connexions nécessaires en utilisant un partitionnement par cliques [GVNG94]. C'est un graphe similaire à ceux que nous avons vu auparavant pour les registres. Chaque nœud représente une connexion entre deux unités. Le lien entre les nœuds représente les connexions qui ne sont pas utilisées de manière parallèle pour le transfert de données à la même étape de contrôle.

2.5.1.6 Calcul de la surface

Une fois cette dernière étape terminée, il est possible en additionnant la surface des unités fonctionnelles, des registres et des unités de connexions de déterminer la surface totale du système. On doit d'abord connaître la surface des composants de

bases, comme les registres simples, les additionneurs, les multiplexeurs, etc. On additionne ensuite la surface de tous les composants de base utilisés.

2.5.2 Dissipation de puissance pour le matériel

Il y une très forte demande provenant des concepteurs de système œuvrant dans l'industrie des semi-conducteurs pour le développement d'outils permettant de contrôler le budget de puissance pendant les différentes phases du processus de conception matérielle. Les économies de puissance pouvant être atteintes à l'aide d'optimisation automatique s'avèrent plus importantes que les économies de puissance pouvant être réalisées en ayant recours à une technologie différente [KrRa98].

Pour des systèmes décrits dans un haut niveau d'abstraction, la cohérence est plus importante que la précision. Donc, une évaluation relative (plutôt qu'absolue) est en général suffisante [MMP96]. C'est-à-dire que l'on cherche en général à déterminer si un modèle dissipe moins de puissance qu'un autre, sans nécessairement chercher à déterminer la valeur exacte de sa dissipation de puissance.

La plupart des outils de prédiction de la dissipation de puissance de haut niveau, combinent profilage statique et simulation de façon à déterminer les dépendances de données[MMP96]. Il existe différents modèles et techniques utilisés pour déterminer la dissipation de puissance et nous allons ici présenter certaines d'entre elles.

2.5.2.1 Modèles de théorie de l'information

L'approche de la théorie de l'information pour l'estimation de puissance de haut niveau dépend de la mesure de l'activité (par exemple, l'entropie) pour obtenir une

estimation rapide [KrRa98]. Cette méthode est utilisée dans [MMP96][Najm96] [MaPe98].

L'entropie est la caractérisation d'une variable ou d'un processus aléatoire. Elle est utilisée dans la théorie de l'information comme une mesure de la capacité de transport d'information. Cette méthode est rapide et elle permet de trouver la dissipation de puissance en utilisant uniquement les variations sur les entrées et sorties du circuit. Si x est une variable booléenne avec la probabilité p que cette valeur soit à « 1 », alors $P\{x=1\} = p$, et l'entropie de x est définie par

$$H(x) = p \log_2 (1/p) + (1-p) \log_2 (1/(1-p)), \tag{2.6}$$

où log_2 est le logarithme base 2. La fonction de l'entropie $H(x)$ est montrée à la figure 2.5.

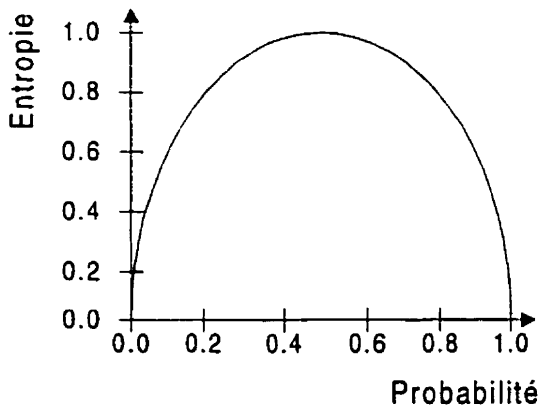


Figure 2.5 Entropie d'une variable booléenne

La fonction $H(x)$ a la valeur maximale de un à $p=0.5$. Intuitivement, si un signal a $p=0.5$ alors il peut faire le nombre maximal de transition et peut transporter le plus d'information. L'entropie représente l'activité sur les bus et peut être utilisée pour estimer la dissipation de puissance. En fait, il a été démontré dans [Najm96], que la

dissipation de puissance (P_{moy}) était proportionnelle à la surface du circuit (A) multiplié par son entropie moyenne (H).

$$Pmoy \propto A * H \tag{2.7}$$

A est une estimation de la surface du circuit qui est représentative de la capacité. Et H représente la moyenne de l'entropie de tous les nœuds du circuit.

Farid Najm [Najm96] a démontré qu'en utilisant cette méthode, il réussissait à avoir une différence inférieure à 9% entre ses estimations et ses mesures expérimentales dans 90% des cas.

2.5.2.2 Modèles basés sur la complexité

Ces modèles relient la dissipation de puissance à certaines notions de complexité de circuit. Par exemple, des paramètres qui influencent la complexité du circuit incluent le nombre et le type d'opérations arithmétiques et booléennes dans la description comportementale, le nombre de termes dans une somme de produits d'une fonction booléenne, etc...[KrRa98][FGS+98].

La plupart des modèles basés sur la complexité reposent sur l'hypothèse que la complexité d'un circuit peut être estimée par le nombre équivalent de portes logiques. Cette information peut être générée directement en utilisant des opérateurs provenant de bibliothèques. On peut utiliser l'expression suivante pour déterminer la dissipation de puissance d'un module logique.

$$Puissance = f\, N(Energie_{porte} + 0.5V^2 C_{charge}) E_{porte}, \tag{2.8}$$

où f est la fréquence de l'horloge et N est le nombre de portes logiques équivalentes. E_{porte} est la consommation interne pour les portes équivalentes (cela inclus les capacités parasitiques). C_{charge} est la capacité moyenne des entrées pour les portes équivalentes

(cela inclus les capacités de sortie, ainsi que les capacité d'interconnexion). C_{charge} est estimé de façon statistique, en se basant sur la moyenne du nombre de portes du circuit. E_{porte} est dépendant de la fonctionnalité du module. Les données sont précalculées et emmagasinées dans une bibliothèque et sont indépendantes du style d'implantation (logique statique versus dynamique, stratégie d'horloge), des paramètres spécifiques pour une bibliothèque particulière (inertie des portes, génération de « glitch » et propagation des signaux), et le contexte du circuit dans lequel le module est instantié. Ceci est un exemple d'un modèle d'estimation de puissance indépendant de l'implantation et des données.

Les estimations seront beaucoup plus précises par l'introduction de paramètres empiriques qui sont déterminés par l'analyse de données réelles. Par exemple, le modèle de puissance pour un graphe de flots de données implanté avec des cellules standards est donné par

$$Puissance = 0.5V^2 f \, (N_I \, C_I \, E_I + N_O \, C_O \, E_O) \, N_M. \tag{2.9}$$

où N_I et N_O dénotent le nombre d'entrée et de sorties respectivement pour le graphe de flots de données, C_I et C_O sont des coefficients de régression qui sont obtenus empiriquement à partir des simulations de bas niveau d'applications semblables, E_I et E_O dénotent les activités de permutation sur les lignes d'entrées et sorties en plus des lignes d'états et N_M dénote le nombre de termes produits dans une optimisation du graphe de flots de données.

2.5.2.3 Modèles basés sur la synthèse

Une autre approche est de faire une synthèse rapide de notre système exprimé sous la forme d'une description comportementale. Une fois la structure RTL de notre

système obtenue, il est possible d'estimer les caractéristiques du circuit en utilisant des techniques d'estimation de dissipation de puissance pour le niveau RTL [MMP96].

2.6 Estimation et modèle de communications

Lorsque l'on sépare un système en plusieurs blocs, des mécanismes de communications et de synchronisations sont nécessaires. Les temps de communications font en fait également partie du temps d'exécution du système entier.

2.6.1 Modèles de communication

Les modèles de communication peuvent être divisés en deux types: ceux basés sur le passage de messages et ceux qui ont une mémoire partagée.

2.6.1.1 Passage de message

Le passage de messages représente une communication directe entre les unités de traitement transmettrices et réceptrices. Il existe différentes façons de transférer les données. Il y a d'abord le transfert de données de façon synchrone. C'est le transfert de données entre deux blocs accompagné par un mécanisme assurant que le bloc transmetteur envoie les données et que le bloc récepteur soit dans un état approprié pour le recevoir. Si le bloc récepteur n'est pas dans le bon état, le transmetteur doit prendre une action appropriée soit en se bloquant jusqu'à ce que le récepteur soit prêt ou en continuant avec une autre action, qui n'est pas reliée à celle-ci. L'inconvénient avec ce type de transfert est que certains processus peuvent rester bloqués pendant une longue période de temps, augmentant alors considérablement leur temps d'exécution.

D'autre part, un transfert de données de façon asynchrone avec l'utilisation de mémoires tampons est également possible. Les données sont envoyées par le transmetteur de façon asynchrone et ces données sont retenues dans une mémoire tampon située entre le transmetteur et le receveur. L'exécution des blocs n'étant pas bloquée, le système opèrera plus rapidement. Le récepteur peut prendre les données au rythme auquel il en a besoin, car elles restent mémorisées dans le tampon. Ces données ne sont pas perdues et sont reçues dans le même ordre qu'elles ont été envoyées. Cette méthode est efficace et rapide, mais elle nécessite un coût en matériel supérieur à cause de l'utilisation des mémoires tampons.

Finalement, il existe le transfert de données de façon asynchrone. Quand un transfert de données ne se fait pas à l'aide d'un tampon ou de façon synchrone, la valeur d'une donnée peut être reçue plusieurs fois ou même pas du tout. C'est souvent le cas avec les informations de condition. Un bloc peut envoyer des données à un autre bloc sans regarder si l'information envoyée précédemment a déjà été traitée. Dans ce cas, les données envoyées vont être écrasées par réécriture, donc il est impossible pour un bloc de recevoir une autre information que la plus récente.

2.6.1.2 Mémoire partagée

Une communication par mémoire partagée utilise un élément de mémorisation partagé pouvant être accédé par plusieurs unités de traitement. Les unités de traitement transmettrices et réceptrices n'ont pas besoin d'être synchronisées. Si une synchronisation est nécessaire, elle doit être spécifiée explicitement. Par exemple, un drapeau « valide » pourrait être inclus pour indiquer que la mémoire a été mise à jour avec une nouvelle valeur. Le modèle de mémoire partagée inclut également un mécanisme de diffusion, qui s'assure que toutes les valeurs ou événements générés par

un processus ou son environnement seront immédiatement perçus par les autres processus.

2.6.2 Canaux de communication

Une fois le mécanisme de communication ou de synchronisation déterminé, un canal de communication doit être choisi pour transmettre les données ou les signaux. Une représentation des canaux possibles est proposé à la figure 2.6.

Lignes dédiées : Les lignes dédiées sont utilisées pour brancher différentes unités de traitement. Très souvent, les unités de traitement ont des mémoires locales et transfèrent des données en utilisant ces lignes. Les lignes dédiées implantent des connexions point à point entre les processus émetteurs et récepteurs qui communiquent en utilisant un protocole de passage de messages.

Bus : Un bus représente un ensemble de canaux simples et peut être utilisé pour échanger des données entre plusieurs unités de traitement. Une unité de traitement émettrice écrit un message sur le bus, en utilisant un protocole de passage de message et l'ensemble des unités de traitement réceptrices peuvent lire le message. En utilisant un bus comme canal de communication, il est possible d'implanter un mécanisme de diffusion (broadcast).

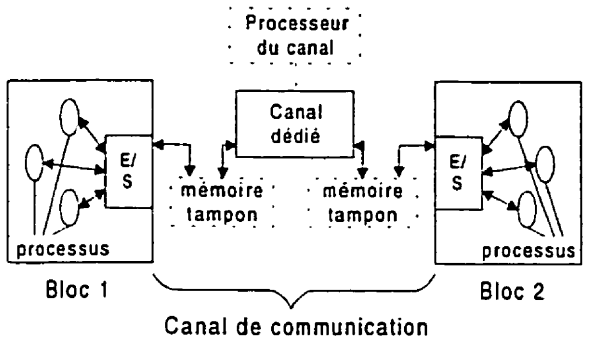
Tampon FIFO : Les tampons FIFO utilisent un protocole de passage de messages. Toutefois, l'unité de traitement peut envoyer des messages sans être synchronisée. L'avantage est qu'aucun cycle d'horloge n'est gaspillé, mais cela se fait au prix de l'ajout d'éléments de mémorisation.

Mémoire partagée : Les communications basées sur des mémoires partagées sont également des communications non-bloquantes. L'unité de traitement émettrice écrit des données en mémoire qui sont lues par l'unité de traitement réceptrice. La différence entre un tampon FIFO et une mémoire partagée est que le tampon FIFO utilise une communication point à point, tandis que la mémoire partagée utilise un bus.

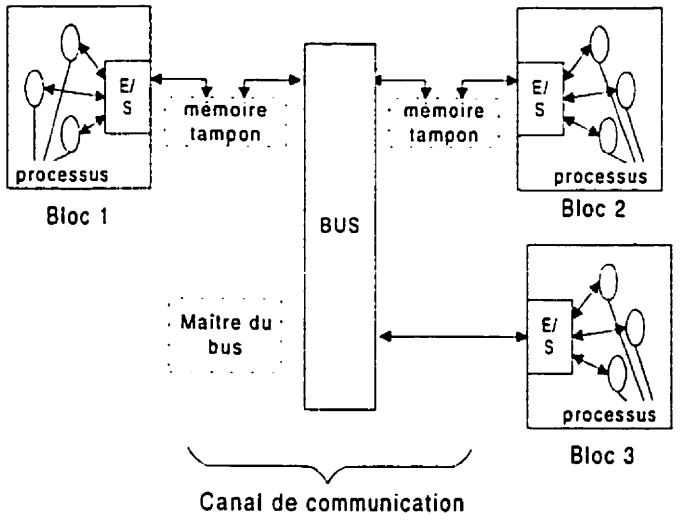
La figure 2.6 présente des exemples de canaux de communication. Les composants en lignes pointillées sont optionnels. La figure 2.6 a) représente une communication point-à-point. C'est une communication par passages de messages. Des mémoires tampons sont optionnelles et peuvent être ajoutées pour des communications non-bloquantes. Et le processeur du canal qui est aussi optionnel gère les accès au canal.

La figure 2.6 b) représente également une communication par passage de messages, mais cette fois-ci à travers un bus. Les composants du système communiquent entre eux à l'aide du bus. Des mémoires tampons peuvent être ajoutées pour empêcher les composants d'attendre que le bus se libère. Car seul un composant à la fois peut envoyer des données sur le bus. Pour gérer les accès au bus, un arbitre peut être ajouté. Ce rôle peut également être joué par un des composants du système.

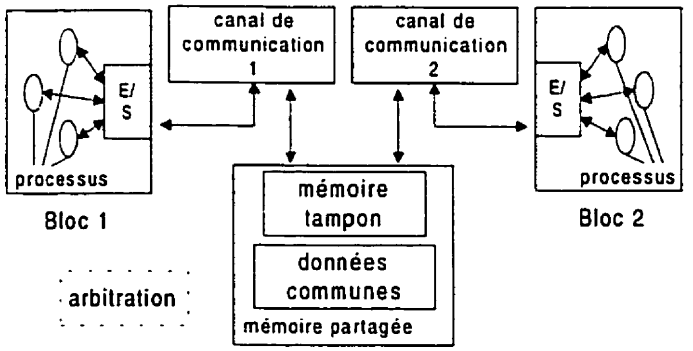
La figure 2.6 c) montre un exemple de communication par mémoire partagée. Le canal de communication est une mémoire partagée. Mais, pour atteindre cette mémoire, deux canaux de communication doivent être ajoutés. Cela pourrait être une communication point-à-point, mais en général une communication par bus est utilisée.



a) Communication point-à-point



b) Communication par bus



c) Communication par mémoire partagée

Figure 2.6 Représentation des canaux de communications

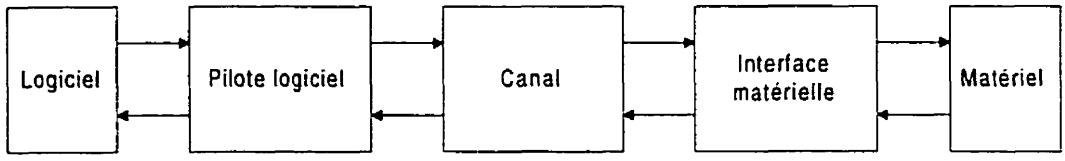


Figure 2.7 Modèle de communication sur un canal.

Un bloc pour l'arbitration de la mémoire et des canaux peut être utilisé afin d'éviter les conflits en mémoire.

2.6.3 Estimation du temps de communication

Tout dépendant des moyens de communication utilisés, les estimations des temps de communication sont plus ou moins difficiles. Nous allons voir la façon d'estimer le temps de communication entre un bloc matériel et un bloc logiciel, mais la même méthodologie peut s'appliquer que ce soit par un transfert entre deux blocs matériels ou logiciels (sur deux processeurs différents) [KnMa98].

Comme on peut le voir à la figure 2.7, un pilote doit être ajouté à la partie logicielle. C'est le pilote qui formate les données et qui envoie les signaux différemment selon les mécanismes choisis. La même chose se produit pour la partie matérielle. Une interface matérielle est ajoutée au système pour adapter la partie matérielle au mécanisme de communication choisi.

Afin de calculer le temps de communication, nous allons trouver le délai de transmission du pilote logiciel, celui de l'interface matérielle et finalement le temps de communication du canal.

Délai du pilote logiciel

Le pilote reçoit n_t mots de l'entrée et produit n_c mots pour le canal. Définissons

n_t = nombre de mots de largeur w_t reçus par le transmetteur;
w_t = largeur en bits de l'entrée:
c_{tc} = nombre de cycles nécessaires pour l'appel du pilote pour la transmission;
c_{tp}= nombre de cycles pour la transmission d'un mot; et
f_t = fréquence de l'horloge.

Nous pouvons calculer le délai de transmission du pilote logiciel (t_{td}) de la façon suivante :

$$t_{td} = (c_{tc} + c_{tp}n_t)/f_t. \qquad (2.10)$$

Délai du canal

Ensuite, si nous supposons que nous connaissons le nombre de mots à transférer, ainsi que le nombre de cycles de synchronisation, nous pouvons trouver le délai de transmission du canal. Soient

c_{cs} = nombre de cycles de synchronisation;
c_{ct} = nombre de cycles de transmission pour chaque mot sur un canal;
n_c = nombre de mots de largeur w_c;
w_c = largeur en bits du canal; et
f_c = fréquence de l'horloge.

Nous pouvons calculer le délai de transmission du canal (t_{cd}) de la façon suivante :

$$t_{cd} = (c_{cs} + c_{ct}n_c)/f_c. \qquad (2.11)$$

Délai de l'interface matérielle

Pour le calcul du délai de l'interface matérielle, nous supposons que l'interface reçoit en plus du paramètre n_c,qui est le nombre de mots à la sortie du canal. Posons

c_{rc} = nombre de cycles pour l'appel du pilote;
c_{rp} = nombre de cycles pour le traitement par le pilote des mots;
n_t = nombre de mots de largeur w_r;
w_r = largeur en bits de la sortie; et
f_r = fréquence de l'horloge.

Nous pouvons calculer le délai de transmission de l'interface (t_{rd}) de la façon suivante :

$$t_{rd} = (c_{rc} + c_{rp} n_t)/f_r. \tag{2.12}$$

Délai total

Si le transfert est fait individuellement, sans utiliser un pipeline, le délai du transfert est égal à la somme des trois délais calculés précédemment. C'est-à-dire que

$$t_t = t_{td} + t_{cd} + t_{rd}. \tag{2.13}$$

Pour un transfert fait à l'aide d'un pipeline (c'est-à-dire que les données passent à travers les trois étapes des communications de la même façon que dans un pipeline), le délai de transfert se calcule de la façon suivante :

$$\text{où } t_m = max\,(t_{td}, t_{cd}, t_{rd}). \tag{2.14}$$

$$t_t = t_m + 2\,(t_m/n_t) \tag{2.15}$$

Le premier terme indique le temps maximal pour les différentes étapes du pipeline (t_m), tandis que le deuxième terme donne une approximation du temps de départ et d'achèvement du pipeline.

2.6.4 Modèle proposé par Ralf Niemann dans l'outil COOL

Maintenant que nous avons vu les types de communications ainsi que les types de canaux utilisés, nous allons déterminer un modèle de communications global pour le système en entier. Nous nous sommes tout d'abord inspirés du modèle utilisé dans l'outil COOL, publié dans [NiMa98]. D'autres types de modèle de communication ont été implantés dans d'autres systèmes de codesign. Par exemple Chinook [Cob95][OrBo98], SpecSyn [GaVa95], Cosmos [DMVJ97][DIMJ97] et Vulcan [GCM92][CLG96].

Un modem est une application qui est dominée par les données et le modèle qui est proposé dans COOL s'intègre bien à notre système. Nous allons par contre y faire quelques modifications de façon à mieux refléter la réalité de notre application. La figure 2-8 donne un aperçu du modèle de communication.

Le contrôleur d'entrées et sorties est utilisé pour communiquer avec l'extérieur du système. Les systèmes dominés par les données, comme nos modems, sont des systèmes transformationnels qui travaillent sur des données en entrées et qui les transforment en sortie. Le contrôleur d'entrées/sorties envoie sur le bus les données provenant de l'environnement et reçoit les données à retourner à l'environnement.

Le contrôleur du système peut être fait en matériel ou en logiciel et il est le coeur du système. Le contrôleur active et désactive l'exécution des autres blocs du système.

Il indique aux blocs quand ils doivent être exécutés et reçoit des signaux des différents blocs lorsqu'ils ont terminé leur exécution.

Le contrôleur d'entrées et sorties, le contrôleur du bus, la mémoire, ainsi que les blocs matériels et logiciels ont tous accès au bus. Un arbitre du bus est alors nécessaire pour éviter les conflits. De plus, la mémoire est nécessaire pour implanter la communications entre les différentes unités de traitement en utilisant la mémoire partagée.

Les blocs matériels et logiciels sont en fait les unités de traitement de notre système. Le comportement du système est divisé en différents blocs qui sont les unités de traitement.

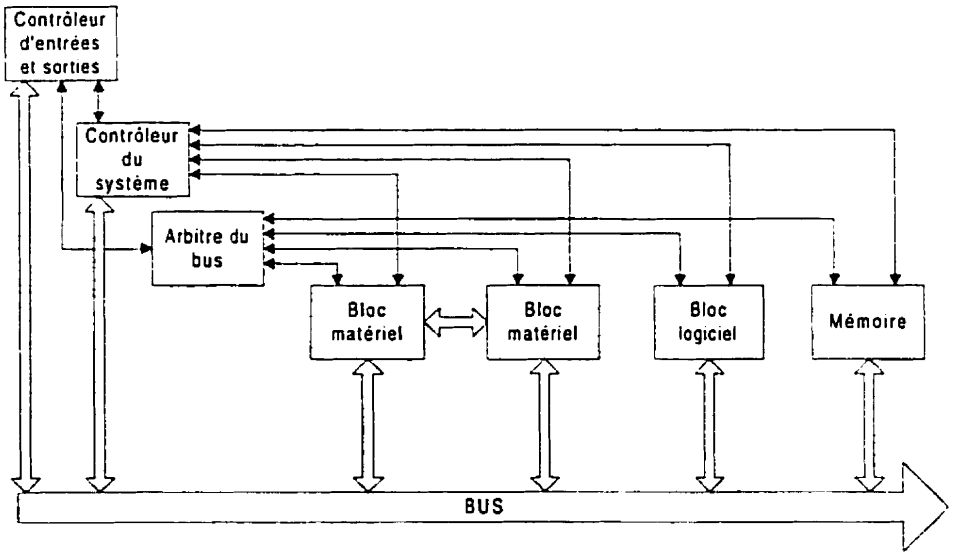


Figure 2.8 Modèle de communications modifié

2.6.4.1 Ajout au modèle

La seule chose que nous ayons ajouté au modèle est une communications directe entre les blocs matériels. Notre application est en fait implantée sous forme de pipeline.

Les données sont donc transférées d'un bloc à un autre au même moment. Si l'on utilise le bus pour transférer toutes les données, il devient le goulot d'étranglement, car cela ralentit de beaucoup la vitesse totale des communications. Pour remédier à cela, nous ajoutons des liens directs entre les blocs matériels qui se suivent dans le pipeline. Pour toutes les communications qui impliquent le processeur, nous n'avons d'autres choix que d'utiliser le bus de données pour le transfert d'un nombre important de données.

2.7 La fonction objectif

Les différents modèles ne sont pas en général évalués à partir d'une seule métrique. Jusqu'à présent, nous avons présenté quatre métriques différentes. Il faut donc être capable d'estimer une fonction objectif à partir des différents estimateurs utilisés. Ces estimateurs doivent refléter les contraintes imposées au système. La fonction objectif calcule la qualité d'un modèle et la valeur que cette fonction retourne est appelée le coût.

2.7.1 Calcul de la fonction avec des pondérations

Les estimateurs n'ont pas tous la même importance et il n'est pas toujours facile de combiner ces valeurs pour en faire une seule. La plupart des approches utilisent une pondération devant chacune des valeurs des estimateurs. Nous trouvons ces différentes approches dans [GVNG94]. Ces pondérations indiquent l'importance relative des estimateurs. Par exemple, si nous avons comme estimateurs, la surface, le temps d'exécution et la puissance dissipée, nous pourrions avoir la fonction objectif suivante :

$$FonctionObjectif = k_1 * surface + k_2 * temps + k_3 * puissance. \quad (2.16)$$

Si la valeur de k_1 est plus grande que celle de k_2 et k_3, alors la surface sera considérée comme l'estimateur le plus important. Le poids est important pour résoudre un des problèmes rencontrés lors du calcul de la fonction objectif. Toute chose étant égale par ailleurs, la relation est généralement inverse entre les estimateurs de surface et de temps d'exécution pour le matériel, donc lorsque l'un augmente, l'autre tend à diminuer. Le problème que l'on retrouve alors, est que quels que soient les changements, le coût tend à demeurer approximativement le même.

2.7.2 Calcul de la fonction objectif avec contraintes

La plupart des systèmes ont des contraintes, et la fonction objectif devrait pouvoir les refléter, de façon à ce que les modèles respectant ces contraintes soient jugés supérieurs à ceux ne les respectant pas. Par exemple, avec la fonction objectif que nous avons vu précédemment, un modèle qui a une surface optimale et qui a un poids k_1 très élevé aura un bon coût, même si ce modèle ne respecte pas les contraintes de temps. Pour résoudre ce problème, il faudrait donc calculer les différents membres de l'équation en utilisant le résultat de l'estimateur et la contrainte reliée à cet estimateur.

$$\begin{aligned} FonctionObjectif = \quad & k_1 * F(surface, contrainte\_suface) \\ & + k_2 * F(délai, contrainte\_délai) \\ & + k_3 * F(puissance, contrainte\_puissance) \end{aligned} \tag{2.17}$$

Une façon commune de calculer la fonction F est de retourner la différence entre l'estimation et la contrainte. Lorsque la contrainte est respectée, la fonction devrait retourner zéro. Cette façon de calculer la fonction objectif démontre seulement que toutes les contraintes sont respectées lorsque la fonction retourne la valeur zéro. Si le but est d'obtenir un design optimal pour un système, il n'est pas avantageux de calculer la fonction objectif de cette façon, car elle ne fera pas la différence entre la qualité de deux modèles qui respectent tous deux les contraintes. Par exemple, dans un système

temps réel, lorsque les contraintes de temps sont respectées, il n'est pas nécessaire d'améliorer la vitesse du système, car celui-ci ne fonctionnera pas plus rapidement. Donc, une fois cette contrainte respectée, l'effort est mis sur l'optimisation des autres aspects du système, comme la surface ou la puissance dissipée.

2.7.3 Calcul de la fonction objectif avec normalisation

Il est également possible de normaliser les valeurs obtenues pour les différents estimateurs [GVNG94]. Par exemple, si la surface est de 10 000 unités et que la contrainte de surface est de 9 000 unités, alors que le temps d'exécution est de 10 unités quand la contrainte en demande 1, la normalisation peut aider à comparer ces deux aspects. En prenant la valeur normalisée, on se rend compte qu'il est plus important de s'attarder à améliorer le temps d'exécution plutôt que la surface. Avec cette approche, la valeur de l'estimateur est divisée par sa contrainte. Les différentes valeurs normalisées peuvent ensuite être additionnées pour former la fonction objectif.

En général, une fonction objectif n'utilise pas seulement un de ces critères (pondérations, contraintes et normalisations), mais fait un mélange des trois.

2.8 Les algorithmes de partitionnement

Une fois la qualité d'un modèle déterminée, un algorithme de partitionnement est utilisé pour examiner différents modèles. Il existe un algorithme précis comme le « branch-and-bound », qui examine l'ensemble des modèles possibles. Mais la plupart sont des heuristiques qui essaient à partir d'un modèle, d'y faire des changements dans le but d'en améliorer la qualité. Un bref aperçu des algorithmes les plus populaires est présenté.

2.8.1 Recherche « BRANCH-AND-BOUND »

L'idéal pour obtenir le modèle qui respecte les contraintes du système de façon optimale serait de pouvoir évaluer tous les modèles possibles. Cela peut être réalisé en utilisant un algorithme tel que Branch-and-Bound [Axel97]. Cet algorithme utilise la structure d'un arbre. L'information est ajoutée graduellement de façon à pouvoir tester toutes les possibilités. Par exemple, si nous partons avec un premier bloc, ce bloc peut être implanté en matériel ou en logiciel. Nous avons donc deux branches à notre arbre. Le deuxième bloc peut également être implanté en logiciel ou en matériel. Donc, nous ajoutons deux branches à chacune des deux branches précédentes, pour un total de quatre branches. L'arbre grossit de cette façon jusqu'à ce que toute l'information ait été analysée et ajoutée. Nous nous retrouvons donc avec un nombre total de 2^n branches si nous avons un total de n blocs dans le système. Une façon de réduire le nombre de branches à analyser est de calculer des bornes sur les valeurs des solutions éventuelles situées plus loin dans le graphe à chaque nœud de l'arbre [BrBr87]. Si cette borne démontre que de telles solutions seront nécessairement pires qu'une solution déjà trouvée, on abandonne l'exploration de cette partie du graphe. La borne calculée sert non seulement à fermer des chemins, mais aussi à choisir celui qui est le plus prometteur, afin de l'explorer en priorité.

Cette technique a un temps de calcul très élevé, aussitôt que le nombre de blocs commence à augmenter. Il est donc important de trouver d'autres algorithmes qui permettent de trouver les meilleurs modèles sans devoir les évaluer tous.

2.8.2 Algorithme vorace

Un algorithme simple et rapide est l'algorithme vorace. Il part d'une partition initiale et déplace les blocs dans le groupe opposé tant qu'il y a des améliorations. Cet

algorithme est utilisé par l'outil Vulcan [GVNG94]. Il commence en créant un partitionnement complètement en matériel. Pour accepter un déplacement, on doit s'assurer qu'il y ait une amélioration de la performance et que le coût en matériel soit encore inférieur à la contrainte imposée par le système. Lorsqu'un bloc est déplacé, l'algorithme essaie de déplacer par la suite les blocs qui sont près de celui-ci, en utilisant une fonction de proximité. L'algorithme vorace est très rapide, mais par contre un des inconvénients de cet algorithme, est qu'il n'est pas capable d'échapper aux minimums locaux.

2.8.3 Recuit simulé

Cet algorithme reproduit un processus physique. Il débute avec un partitionnement initial et une température de recuit initiale [Axel97][GVNG94]. Cette température est lentement réduite. Pour chaque température, des mouvements aléatoires sont effectués entre le logiciel et le matériel. L'algorithme accepte tous les changements qui entraînent une augmentation de la qualité du système. S'il n'y a pas d'augmentation de la qualité, alors le changement peut quand même être accepté avec une certaine probabilité. Plus la température du recuit simulé diminue, moins les changements n'entraînant pas d'amélioration sont acceptées.

Un avantage, avec cet algorithme, c'est qu'il ne reste pas pris dans des minimums locaux. Des changements négatifs sont acceptés dans le but d'atteindre éventuellement un système de meilleure qualité. Chaque bloc n'est déplacé qu'une seule fois dans la séquence, diminuant ainsi la complexité de l'algorithme.

2.8.4 Algorithme génétique

L'algorithme génétique nous permet d'obtenir un ensemble de modèles à chaque itération contrairement aux autres algorithmes qui n'utilisent qu'un modèle à la fois [Axel97][GVNG94]. Il est basé sur le processus d'évolution génétique. Un ensemble de modèles est appelé une génération. L'algorithme génétique crée une nouvelle génération à chaque itération de l'algorithme en imitant trois méthodes évolutives qui se trouvent dans la nature.

Premièrement la sélection qui choisit de manière aléatoire des modèles de haute qualité qui seront recopiés dans la génération suivante. Le croisement est également utilisé. Celui-ci consiste à prendre deux modèles de la génération courante qui ont une qualité supérieure afin de les croiser, avant de les envoyer dans la génération suivante. Finalement, la mutation, c'est-à-dire le changement à l'intérieur même d'un modèle est fait de manière aléatoire. L'algorithme s'arrête lorsqu'un modèle a survécu un nombre fixe d'itérations.

Comme le recuit simulé, cet algorithme peut atteindre de très bons résultats, mais il demande un temps de calcul très long. De plus, pour conserver tous les modèles d'une génération, une importante quantité de mémoire est nécessaire lors de l'exécution de l'algorithme.

2.8.5 Recherche Tabu

Cet algorithme est inspiré d'une technique de l'intelligence artificielle dans laquelle le concept de mémoire est utilisé dans le but de rendre la recherche plus efficace [HeEr97]. Pendant la recherche, une mémoire à court terme est utilisée pour éviter que des récents déplacements soient annulés en faisant les déplacements inverses. Cette

mémoire à court terme est connue sous le nom de liste tabou et elle conserve les déplacements qui ne sont pas permis ou tabou. Cette liste a une longueur de n, qui indique le nombre d'itérations de l'algorithme pendant lesquelles un déplacement peut être interdit. Il existe également une mémoire à long terme qui garde l'information de l'évolution globale de l'algorithme. Un avantage de cet algorithme comparé aux autres est qu'il n'est pas fait d'une manière aléatoire. Les mêmes résultats seront obtenus si la même partition initiale est utilisée.

Chapitre 3
La technologie xDSL

Nous allons maintenant présenter les applications que nous avons développées. Ces applications serviront par la suite à tester et valider notre méthodologie de codesign.

Ces dernières années ont marqué le développement des communications internet à grande échelle. Les demandes en taux de transmission sont de plus en plus élevées. L'utilisation de plus d'informations multimédias sur internet par les entreprises et les utilisateurs résidentiels est un facteur de croissance important. Un autre facteur de cette augmentation est la disponibilité de réseaux à des coûts abordables, ce qui permet à un plus grand nombre d'utilisateurs d'accéder à des données à partir d'emplacements externes.

3.1 Qu'est-ce que les services xDSL?

Il existe présentement à travers le monde plus de 700 millions de lignes téléphoniques. Ce réseau est un système qui est déjà en place et qui nous permet de rejoindre une grande partie de la population mondiale. Les modems utilisés par la majorité des usagers ont présentement un taux maximal de 56 Kps. Ces modems ont une vitesse limitée à cause de la technologie analogique qu'ils utilisent. Une nouvelle technologie doit donc être utilisée pour dépasser cette limite, sans devoir ajouter de nouveaux réseaux. Ceci est possible depuis quelques temps déjà en utilisant des modems de la famille xDSL. Ils sont de plus en plus facilement accessibles et auront probablement un impact important dans les prochaines années pour supporter les accès internet/intranet à hautes vitesses, les services en-ligne, les vidéos sur demande, les

signaux TV, le divertissement interactif et la transmission de la voix. Le « x » de xDSL est utilisé pour décrire différents types de technologie, incluant le ADSL, le Universal ADSL et le VDSL.

Le principal avantage des services xDSL à haute vitesse est qu'ils peuvent tous être supportés par des fils de cuivre téléphoniques ordinaires, qui sont déjà installées dans la plupart des édifices commerciaux ou résidentiels [XDSL99]. De plus, il est possible d'utiliser le téléphone tout en restant branché avec le modem. En fait, il est possible de transmettre la voix et des données en même temps, ce qui élimine le besoin d'ajouter une ligne téléphonique ou un câble additionnel.

Il existe plusieurs types de modems dans la famille xDSL. Tous ces modems sont duplex, c'est-à-dire qu'ils peuvent transmettre dans les deux directions. Ils peuvent recevoir des données et en transmettre. Certains sont symétriques, car ils ont le même taux de transmission de données que la communication soit en amont (upstream) ou en aval (downstream). Tandis que d'autres modems de cette famille sont asymétriques. Donc, les taux de transfert des données en amont et en aval ne sont pas les mêmes.

Une autre caractéristique des modems de la famille xDSL, est que le taux de transfert des données est inversement proportionnel à la distance de transmission. Plus la distance de transmission est longue, plus il y aura d'erreurs et moins de données seront transmises par unité de temps.

3.2 Historique

L'acronyme DSL (Digital Subscriber Line) a au départ été utilisé pour se référer à la technologie utilisée pour supporter la transmission dans les réseaux numériques à intégration de service (ISDN, Integrated Service Digital Network) à travers une paire de

fils de cuivre torsadée [Hawl97]. Le taux de transfert de ISDN était alors de 144 kbs. En utilisant DSL, on parle de ligne. Pour avoir une ligne, deux modems sont nécessaires. Il peut donc y avoir une confusion lors de l'achat d'un modem, car même si nous utilisons « ligne » dans l'acronyme, celui-ci peut également référer à un simple modem.

Au milieu des années 80, une étude a été faite par les laboratoires de Bellcore pour le développement d'un transmetteur DSL ayant de plus hautes vitesses. Le HDSL (High Data Rate Digital Suscriber Line) est né. On retrouve encore cette technologie dans les lignes T1 et E1. Plus récemment, une deuxième version de ce type de modem a été réalisée, s'appelant HDSL2. Cette nouvelle version utilise les mêmes normes que le HDSL, mais elle n'utilise qu'une ligne téléphonique au lieu des deux nécessaires pour la première version du HDSL.

Puis, en 1992 le comité T1 a initié un projet de norme pour le ADSL. La norme sera officiellement acceptée en 1995. L'approche DMT a été choisie pour implanter ce type de système. Et depuis un peu plus d'un an, deux nouveaux modems ont vu le jour à partir de la norme du ADSL. Le Universal ADSL, qui est une version plus légère du ADSL et le VDSL qui est une version plus performance du ADSL.

3.3 Les différents types de xDSL

Nous allons présenter trois types de modems appartenant tous à la famille xDSL. Ces modems ont les acronymes suivants : ADSL, UADSL et VDSL [DSLK99] [Emer99][Hawl97][High99]. Ces différents modems seront expliqués plus en détails dans les sections suivantes. La raison pour laquelle nous avons choisi ces trois modems est qu'ils utilisent la même technologie, mais dans des proportions différentes. Le

tableau 3.1 démontre les différents taux de transmission et les caractéristiques des différentes modems étudiés.

3.3.1 Asymmetric Digital Subscriber Line (ADSL)

Comme son nom l'indique, le modem ADSL est asymétrique [DaPe95][DEA+99]. Ses taux de communications en amont et en aval, combinés au fait que le modem est toujours en action, fait du modem ADSL un excellent modem pour les communications internet/intranet et les vidéos sur demande. Les utilisateurs de ce type d'application téléchargent en général beaucoup plus d'information qu'ils en envoient. Mais, il est très peu pratique pour supporter les vidéoconférences, car le taux de transfert n'est pas assez élevé dans les deux directions[DSL99].

Tableau 3.1 Taux de transmission et caractéristiques des modems

Modem	Nom	Taux en aval	Taux en amont	Type	Distance (pieds)	Utilisateurs
ADSL	Asymmetric Digital Subscriber Line	9 Mbps	640 kbps	Asymétrique	18000	Utilisateur particulier
UADSL	Version légère du ADSL	1.5 Mbps	384 kbps	Asymétrique	18000	Utilisateur particulier et bureau central (CO)
VDSL	Very high data rate Digital Subscriber Line	52 Mbps	2.3 Mbps	Asymétrique et peut être symétrique	4500	Unités réseaux optiques (ONU)

L'installation du ADSL est complexe et nécessite en général l'aide d'un technicien. Cette complexité provient principalement du diviseur de ligne (POTS splitter) qui sépare la voix et les données lors de la réception des signaux. Il est possible de voir un aperçu de cette implantation à la figure 3.1. On prévoit dans le futur utiliser ce type de modem pour les changements de paquets dans les circuits, comme un

routeur IP (Internet Protocol) et éventuellement un commutateur de donnée ATM (Asynchronous Transfer Mode) ou technologie temporelle asynchrone.

3.3.2 Universal Asymmetric Digital Subscriber Line (UADSL)

C'est une version légère du ADSL, appelée également Splitterless, G.992.2, G.lite ou Universal ADSL [AbJe98][Brow99][CIAD98][Cole99]. Le UADSL est une version plus légère, donc moins complexe et rapide du modem ADSL, qui élimine le besoin de faire installer un diviseur (splitter) par un technicien. L'élimination du diviseur de ligne (POTS splitter) simplifie l'installation du modem et permet d'en réduire le coût (figure 3.1). Le Universal ADSL doit fonctionner sur de plus grandes distances que le ADSL, le rendant plus facilement disponible pour une grande quantité d'utilisateurs. Il supporte également la voix ainsi que les données. Le UADSL est en fait un compétiteur direct du modem câble [Hawl97].

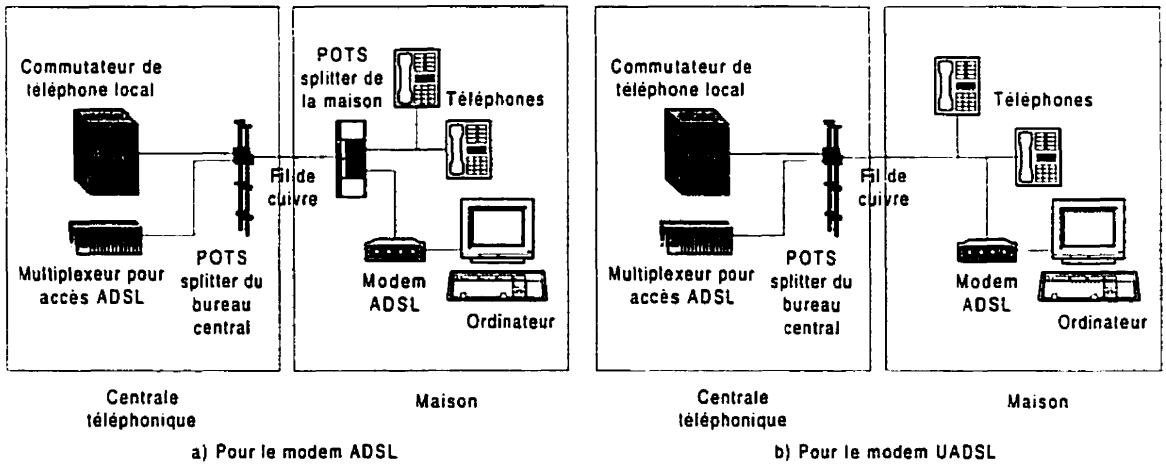


Figure 3.1 Configuration d'un système pour le ADSL ou le UADSL

3.3.3 Very High Bit-Rate Digital Suscriber Line (VDSL)

La technologie VDSL est la plus rapide des technologies xDSL. Le VDSL est un transmetteur qui peut être asymétrique ou symétrique et qui fonctionne d'une façon similaire au ADSL, mais en ayant un débit plus élevé, sur des distances moindres [DSLD99].

Un modem VDSL utilisé dans une unité de réseau optique (Optical Network Unit, ONU), doit pouvoir fonctionner avec un budget de puissance limité. Ceci est dû aux problèmes de dissipation de puissance que l'on retrouve dans les ONU. Ces limitations de puissances sont dues à un espace limité, des conditions d'environnement sévères, ainsi que la difficulté (même impossibilité) d'implanter un refroidisseur d'air aux endroits où sont localisés les ONU [QAM99]. Pour le VDSL, le premier critère à optimiser est la dissipation de puissance [VDSL99].

3.4 Les trois technologies sur le même modem

Un avantage de placer ces trois types de modems dans un même système est qu'il est alors possible pour un utilisateur de déterminer lors de sa connexion la vitesse de transfert qu'il veut utiliser. Un seul modem peut être utilisé pour faire trois types de connexions différentes.

Par contre, les caractéristiques des systèmes entraînent des optimisations différentes dans la technologie de transmission. L'utilisation de la même technologie de façon à atteindre une interopérabilité causera d'importantes dégradations de performance et surchargera les systèmes avec un coût et une complexité additionnelle [Oksm99]. Nous avons donc abandonné cette idée.

3.5 Les schémas de modulation des xDSL

Il existe plusieurs façons de modifier le signal d'une porteuse à haute fréquence. Il y a deux principales technologies, utilisant des schémas de modulation : modulation amplitude/phase sans porteuse (CAP, carrierless amplitude phase) et modulation discrete multitone (DMT). CAP et DMT utilisent fondamentalement la même technique de modulation d'amplitude en quadrature (QAM, Quadrature Amplitude Modulation) mais diffèrent dans la façon de l'appliquer. En fait, CAP opère dans le domaine temporel, tandis que DMT opère dans le domaine fréquentiel. La technologie CAP a également une implantation beaucoup plus analogique, tandis que la technologie DMT est plus numérique [Roka99].

3.5.1 CAP (Carrierless Amplitude/Phase)

CAP ou modulation amplitude/phase sans porteuse est un processus de conservation de largeur de bande utilisé dans plusieurs modems, qui permet à deux porteuses de signaux numériques d'occuper la même largeur de bande de transmission. La technologie CAP est très fortement reliée à QAM (Quadrature Amplitude Modulation). En fait, mathématiquement, les deux processus peuvent être considérés comme une simple transformation de l'un vers l'autre. Les deux processus sont des porteuses de signaux simples. Cela signifie que le taux de transfert de données est divisé en deux et est modulé sur deux porteuses orthogonales, avant d'être combinés et transmis. Les modems utilisant la technologie CAP sont en mesure de discerner s'ils doivent utiliser les nombres inférieurs ou supérieurs d'amplitude et de phases pour maîtriser le bruit et l'interférence de la paire torsadée. Au départ, le CAP test la qualité de la ligne d'accès et implante la version la plus efficace du QAM pour assurer une performance satisfaisante pour les transmissions individuelles de signaux [XDSL99].

Générer une onde modulée qui transporte une amplitude et une phase n'est pas si facile. Pour réussir ce défi, la technologie CAP emmagasine des parties du message modulé en mémoire et rassemble alors les parties dans une onde modulée. Le signal porteur est éliminé avant la transmission parce qu'il ne contient aucune information et il est rassemblé par le modem récepteur. Les symboles CAP ont une fréquence élevée (une impulsion plus élevée), mais durent très peu de temps.

3.5.2 DMT (Discrete Multitone)

DMT est différent de CAP, car il utilise plusieurs porteuses, mais de bandes étroites, transmettant simultanément en parallèle [Tomb97]. La modulation ayant de multiples porteuses nécessite une orthogonalité entre les porteuses et pour y arriver, une transformée de Fourier est utilisée.

DMT divise les fréquences disponibles en différents sous-canaux (par exemple 256 canaux pour le ADSL). Comme pour la technologie CAP, un test est fait au départ pour déterminer les capacités de transport de chacun des sous-canaux. Les données sont divisées pour former des symboles et sont distribuées selon une combinaison spécifique de sous-canaux, selon leur habileté à transmettre ces symboles. Pour vaincre le bruit, plus de données sont envoyées dans les basses fréquences et un peu moins dans les fréquences plus élevées[XDSL99]. Les quatre premiers kHz sont réservés au transport de la voix. C'est ce que l'on appelle le « bons vieux réseaux téléphoniques » ou POTS (Plain Old Telephone Service), comme on peut le voir à la figure 3.2. Les autres fréquences sont divisées en sous-canaux. DMT a de longs symboles, mais a une bande de fréquence très étroite.

DMT a été sélectionné par l'American National Standards Institute (ANSI). Le comité pour les normes T1E1.4 pour le ADSL a sélectionné cette technologie comme norme pour le modem [BaHo95].

3.5.3 Les avantages et les inconvénients des deux technologies

La technologie DMT offre une plus grande flexibilité pour le taux de transfert des données [Pryc99][DMT99a]. Avec DMT, il est facile de varier le nombre de bits par porteuse. Les systèmes CAP qui sont disponibles présentement, n'offrent pas une aussi grande flexibilité, car l'atteinte d'une telle flexibilité demanderait un temps de calcul beaucoup plus élevé, ainsi que des coûts supérieurs. La densité du spectre de puissance du DMT est complètement configurable pour les optimisations SNR (signal to noise ratio), pour le bruit de la sortie, c'est-à-dire le bruit généré par le xDSL vers le monde externe et pour la compatibilité avec les autres services (par exemple en permettant au modem de transmettre à des bandes de fréquences particulières.) Donc, CAP a une moins grande opérabilité que DMT.

La technologie DMT offre également une plus grande résistance au bruit de l'impulsion (l'envoi du symbole). Cette technologie est moins sensible à ce bruit à cause d'une plus grande durée de son symbole. Le domaine fréquentiel est en général beaucoup plus stable que le domaine temporel. CAP nécessite de large entrelaceur et démontre une tendance à propager plus facilement les erreurs [DMT99b].

D'un autre côté, la technologie DMT est malheureusement plus dispendieuse et plus complexes que la technologie CAP [XDSL99]. De plus, un modem utilisant la technologie DMT dissipe un peu plus de puissance qu'un même modem fait en utilisant la technologie CAP.

Pour l'implantation de nos modems, nous allons favoriser l'utilisation de la technologie DMT, car cette technologie est déjà utilisée comme norme pour le ADSL. Nous allons donc maintenant voir les différents éléments nécessaires pour l'implantation de la technologie DMT.

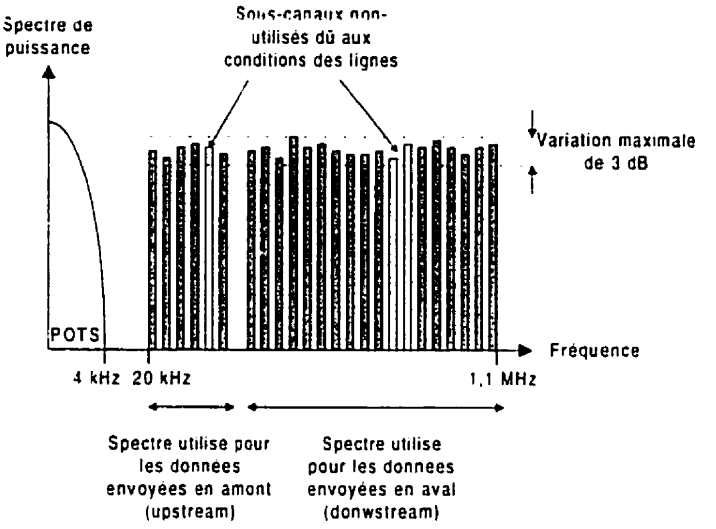


Figure 3.2 La technologie DMT

3.6 Description des différents blocs en utilisant la technologie DMT

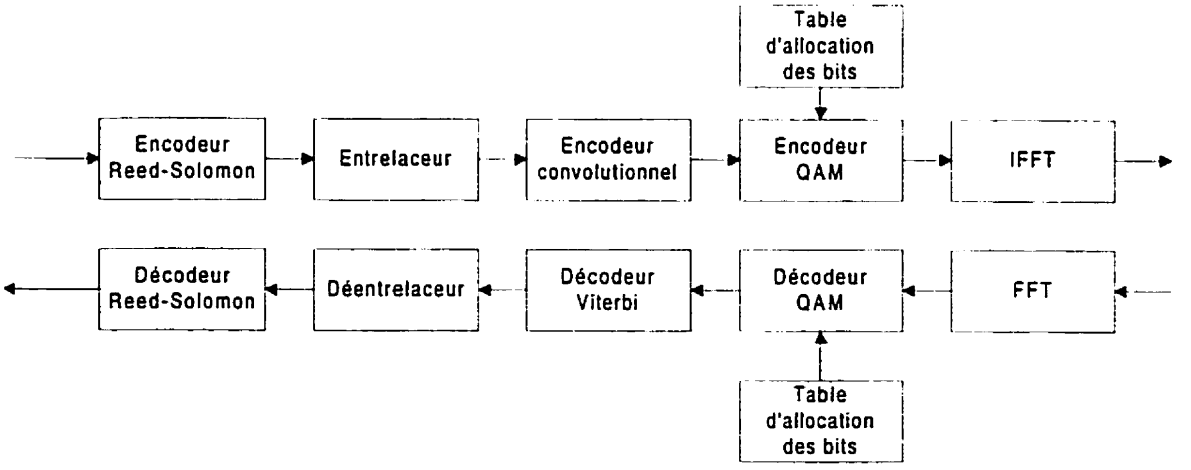


Figure 3.3 Les différents blocs de la technologie DMT

Les différentes parties de la technologie DMT sont disponibles à la figure 3.3. Ces différentes parties seront décrites plus en détails dans les prochaines sous-sections.

3.6.1 Encodeur Reed-Solomon

Le codage Reed-Solomon est une méthode de correction d'erreurs sous la forme de codage de blocs. Le codage de bloc consiste à calculer des symboles de parité à partir de plusieurs symboles d'un message. Les symboles de parités sont ajoutés à la fin des symboles du message formant un mot du code [Reed98]. Mathématiquement, les codes Reed-Solomon sont basés sur une arithmétique de champs Galois.

En résumé, voici les paramètres utilisés par un code *Reed-Solomon (n,t)* avec des symboles de $GF(2^m)$:

$n = 2m-1$	la longueur du code en symboles
$k = n-2t$	nombre de symboles de renseignement
$n-k = 2t$	nombre de symboles de vérification (ou de parité)

où t est le nombre de symboles en erreur pouvant être corrigés.

La théorie derrière l'arithmétique des champs Galois, ainsi qu'une description plus détaillée de l'algorithme Reed-Solomon pour l'encodeur et le décodeur sont disponibles en annexe A. Nous allons pour le moment donner une brève description des éléments importants de l'encodeur.

La partie de l'encodeur est assez simple. Le polynôme code $c(x)$ est composé de l'addition du polynôme information $d(x)$ et du polynôme de vérification $p(x)$. $c(x)$ doit être un multiple du polynôme générateur $g(x)$.

La complexité de l'encodeur et du décodeur Reed-Solomon est directement relié au nombre de symboles de vérification.

3.6.2 Décodeur Reed-Solomon

La partie du décodeur Reed-Solomon est plus complexe que l'encodeur. Le polynôme reçu $r(x)$ consiste en l'addition du polynôme transmis $c(x)$ et du polynôme d'erreur $e(x)$. Donc, pour trouver le polynôme transmis $c(x)$ à partir de celui reçu $r(x)$, nous devons trouver le polynôme d'erreur $e(x)$,

$$e(x) = r(x) + c(x). \tag{3.3}$$

De façon à déterminer $e(x)$, nous devons connaître l'emplacement des erreurs, ainsi que la valeur de l'erreur.

En utilisant l'algorithme Berlehamp que vous trouverez en annexe A et le syndrôme $s(x)$, nous pouvons trouver le polynôme évaluateur d'erreur, $\Omega(x)$. Le polynôme évaluateur d'erreurs $\Omega(x)$ est utilisé pour trouver la valeur de l'erreur (ou la magnitude de l'erreur) pour chaque emplacement d'erreur. Le polynôme évaluateur d'erreur est ensuite utilisé pour trouver la valeur d'erreur aux différents emplacements et ces valeurs serviront à définir le polynôme d'erreur $e(x)$. Ces polynômes sont ensuite utilisés pour retrouver le polynôme envoyé par le transmetteur en y éliminant les erreurs.

3.6.3 Entrelaceur

Lors de la transmission des données, il arrive que plusieurs symboles consécutifs soient perdus. Par exemple, un symbole QAM représente 12 bits. Si ces 12 bits consécutifs sont perdus, il est alors plus difficile de les retrouver et d'éliminer les erreurs. Un entrelaceur est alors utile. Il ne fait que changer l'ordre des bits de façon à ce que les bits perdus soient dispersés dans le code. Il est alors beaucoup plus facile de retrouver une erreur dans douze mots que douze erreurs dans un seul mot. Une représentation de l'entrelaceur est illustrée à la figure 3.4.

3.6.4 Dé-entrelaceur

Le dé-entrelaceur fait exactement l'opération inverse de l'entrelaceur. Il replace les bits aux bons endroits de façon à pouvoir les lire dans le bon ordre.

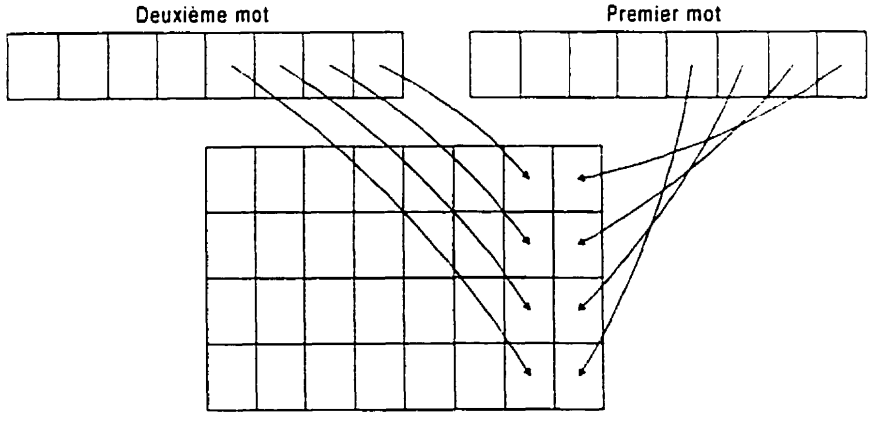


Figure 3.4 L'entrelaceur

3.6.5 Encodeur convolutionnel

Un code convolutionnel est différent d'un code bloc comme le Reed-Solomon. La différence est au niveau de la mémorisation des données précédentes pour le traitement d'un code. C'est-à-dire que les n sorties de l'encodeur ne dépendent pas seulement des k entrées, mais également des m entrées précédentes au bloc. Un code convolutionnel (n,k,m) peut implanter k-entrées et n-sorties en utilisant une mémoire de dimension m. En général, on peut voir l'encodeur comme une machine à états, où les transactions d'un état à un autre indiquent les sorties de l'encodeur.

Différentes techniques peuvent être utilisées pour encoder un code convolutionnel. Certaines de ces techniques sont disponibles en Annexe B.

3.6.6 Decodeur Viterbi

En 1967, Viterbi [QAM99] propose un algorithme de décodage à maximum de vraisemblance (maximum likelihhood decoding) qui est relativement facile à implanter avec des codes ayant un ordre de mémoire relativement petit. De plus amples informations sont disponibles sur le Decodeur de Viterbi en Annexe B.

On peut décoder un code convolutionnel en formant un arbre. C'est-à-dire que les différents chemins possibles dans la machine à états forment des branches. L'arbre peut rapidement prendre des proportions importantes et certains ensembles de branches reviennent souvent. Une façon de résoudre le problème est de créer un treillis. On avance d'une branche dans le treillis à chaque décodage de données. Vous trouverez des exemples de treillis dans l'annexe B.

Dans l'algorithme de Viterbi, lorsque le décodage se fait sur un BSC (Binary Synchronous Communication) ou en français, « communication synchrone binaire », la distance de Hamming est utilisée. La distance de Hamming peut être considérée comme le nombre de bits qui diffèrent entre les données reçues et le choix du treillis. Le but de l'algorithme est de trouver le chemin ayant la plus petite distance de Hamming à travers le treillis en comparant les distances de Hamming de toutes les branches du chemin entrant dans chaque état. Lors du processus de décodage, si à un moment donné, on trouve qu'il sera impossible d'atteindre de cette façon la plus petite distance Hamming, le chemin est éliminé. Donc, le décodeur compare toutes les distances de tous les chemins entrant dans un état et garde uniquement le survivant, qui mène au chemin ayant le plus de probabilité d'être le bon.

3.6.7 Encodeur QAM

L'encodeur prend l'ensemble des bits d'information et l'encode sur des points de constellation de N QAM. On voit dans la figure 3.5 un exemple d'encodage QAM pour 2 et 4 bits.

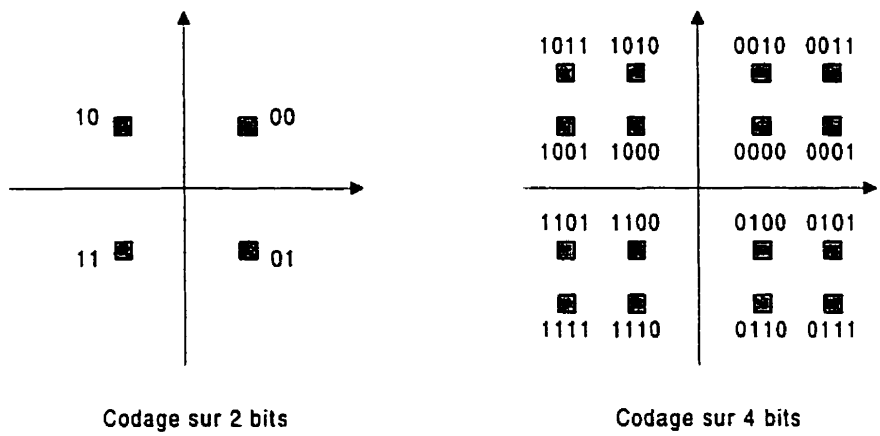


Figure 3.5 Encodeur QAM

Ce codage est fait selon la table de chargement de bits qui définit le nombre de bits portés par chaque tonalité. Les porteuses qui ont un rapport signal à bruit (SNR) élevé peuvent porter plus de bits que les porteuses ayant un faible ratio signal à bruit. En fait, la table de chargement de bits reflète la variation du SNR selon les fréquences. La figure 3.6, démontre un exemple typique du nombre de bits chargés sur chaque porteuse selon le rapport signal à bruit de chacunes des porteuses [DSL99].

3.6.8 Décodeur QAM

Le décodeur QAM fait les opérations inverses de l'encodeur QAM. Il prend les points de constellation de N QAM et les décode en bits. Le décodage se fait en utilisant la table de chargement de bits qui définit le nombre de bits portés par chaque tonalité. De la même façon que pour l'encodeur QAM, plus le rapport signal à bruit est élevé, plus le nombre de bits envoyé sur un sous-canal sera élevé.

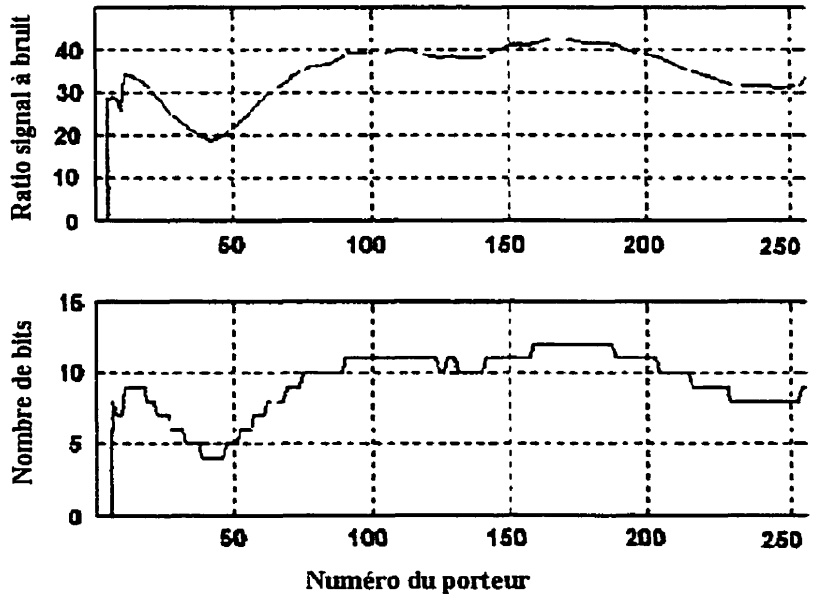


Figure 3.6: Comparaison de la charge des porteuses selon leur rapport signal à bruit

3.6.9 Table de chargement des bits

La table d'allocation des bits est calculée pendant le temps de départ (startup) avec la mesure du rapport signal à bruit (SNR) effectif, pour permettre une utilisation maximale des canaux. Lorsque nous voulons desservir un client avec un taux de transfert spécifique, nous allouons des bits aux porteuses de façon à ce que la somme de tous les bits allouées sur les porteuses correspondent au taux de transfert désiré et que la probabilité d'erreur sur chaque porteuse soit environ la même.

Lorsque nous voulons donner à l'utilisateur le taux maximal de transfert disponible, nous allouons à chaque porteuse le nombre maximal de bits qu'elle peut transmettre sans erreur, basé sur la mesure du rapport signal à bruit du signal. Ce mode est habituellement référé comment étant un mode aux taux adaptatif. De cette façon, l'utilisateur étant près d'une centrale aura un rapport signal à bruit plus élevé et par le fait même, un taux de transfert plus élevé. Tandis que l'utilisateur plus éloigné souffrira de l'atténuation de la ligne et un nombre moins élevé de bits sera assigné à chaque porteuse [DSL99][QAM99]. Les équations utilisées pour décrire la table de chargement des bits sont disponibles en Annexe C.

3.6.10 Transformée rapide de Fourier (FFT)

L'élément principal de la mise en place de la technologie DMT est la FFT/IFFT. L'IFFT est une voie élégante et efficace de créer une somme de N porteuses, chacune modulée par sa propre phase et amplitude [Odde97].

La transformée rapide de Fourier (FFT) est une réalisation rapide de la transformée de Fourier discrète (DFT) qui se fonde sur la simplification et la classification mathématiques de la séquence d'entrée pour réduire le temps d'exécution.

La définition standard de la transformée de Fourier discrète d'une séquence d'entrée $x[n]$ de longueur N est :

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{2\pi i/N}. \tag{3.5}$$

James Cooley et John Tukey ont développés leur propre formulation de la DFT où $W_N = e^{j(2\pi/N)}$.

$$X[k_1 + N_1 k_2] = \sum_{n_2=0}^{N_2-1} [(\sum_{n_1=0}^{N_1-1} x[N_2 n_1 + n_2]\ W_{N_1}^{k_1 n_1})\ W_N^{k_1 n_2})]\ W_{N_2}^{k_2 n_2}). \tag{3.6}$$

Cela permet alors pour le cas du radix-2, $N1 = 2$ et $N2 = N / N1 = N/2$ et

$$k_1=0,\ldots,N_1-1, \tag{3.7}$$

$$k_2=0,\ldots,N_2-1, \tag{3.8}$$

$$k_1=0,\ldots,N_1-1, \tag{3.9}$$

$$k_1=0,\ldots,N_1-1. \tag{3.10}$$

Nous avons donc :

$$X[k_1 + 2k_2] = \sum_{n_2=0}^{N/2-1} [(x[n_2] + (-1)^{k_1}\ x[N/2 + n_2])\ W_N^{k_1 n_2}]\ W_{N/w}^{k_2 n_2} \tag{3.11}$$

3.6.11 Transformée rapide de Fourier inverse (IFFT)

Pour obtenir la transformée de Fourier rapide inverse (IFFT), il s'agit de calculer la transformée de Fourier directe (FFT), puis de diviser chacune des valeurs obtenues par N afin de prendre le complexe conjugué du résultat. Cette dernière étape n'est pas nécessaire si le signal temporel à obtenir a des valeurs réelles, puisque le conjugué d'un nombre réel est ce nombre lui-même.

3.7 Spécifications des différents modems.

Voici les paramètres et les spécifications en temps des différents blocs des modems. Les différents paramètres ont été expliqués précédemment dans la section 3.6.

Nous avons dû déterminer nous-même les spécifications et les taux d'exécution des différents blocs. Le seul paramètre que nous possédions était le nombre de points de la FFT et de la IFFT, car ils correspondaient au nombre de sous-canaux utilisés par la technologie DMT. Ces paramètres influençaient également les paramètres de l'encodeur QAM et la table de chargement des bits. Les paramètres des autres blocs ont été déterminés selon les performances demandées pour chacun d'eux.

Les performances des différents blocs ont été déterminées à partir des taux de transferts en amont et en aval demandés par la norme des modems. Par exemple, le Universal ADSL demande un taux de transfert en amont de 384 kbps. Selon les paramètres de l'encodeur Reed-Solomon, nous devions déterminer le nombre de fois où il doit être exécuté pour traiter 384 kbits par secondes. Le nombre de bits à traiter par les blocs augmente d'un bloc à l'autre selon leur place dans l'algorithme de l'encodeur ou du décodeur. Par exemple, la sortie de l'encodeur Reed-Solomon a 128 symboles

binaires de plus que son entrée. Les blocs suivants auront donc un plus grand nombre de symboles à traiter.

La colonne *entrée* indique le nombre de bits à l'entrée du bloc et la colonne *sortie* indique le nombre de bits à la sortie du bloc. Cela n'inclut pas les signaux de contrôle. Le temps d'exécution demandé est le temps d'exécution maximal que peut prendre le bloc afin de respecter ses contraintes de temps.

Tableau 3.2 Les spécifications du UADSL

Nom du bloc	Paramètres	Entrée (en bits)	Sortie (en bits)	Temps d'exécution demandé (ms)
Encodeur Reed-Solomon	GF (2**8) m= 8 $\Rightarrow x^8 + x^4 + x^3 + x^2 + x$ n = 255 t = 8 k = 239	192 mots de 8 bits (1536 bits)	208 mots de 8 bits (1664 bits)	3,91
Entrelaceur	N/A	1664	1664	3,91
Encodeur convolutionnel	K=3 G(0)=101 G(1)=111	1664	3328	3,91
Encodeur QAM	16 symboles à la fois Maximum de 12 bits dans un symbole	192	256	0,22
Table de chargement des bits			16 * 8 bits (96 bits)	0,22
IFFT	16-points	256	256	0,22
FFT	128-points	2048	2048	0,33
Table de chargement des bits			128*8 bits (1024 bits)	0,33
Décodeur QAM	128 symboles à la fois Maximum de 12 bits dans un symbole	2048	1536	0,33
Décodeur Viterbi	K=3 G(0)=101 G(1)=111	3328	1664	0,98
Dé-entrelaceur	N/A	1664	1664	0,98
Décodeur Reed-Solomon	GF (2**8) m= 8 $\Rightarrow x^8 + x^4 + x^3 + x^2 + x$ n = 255 t = 8 k = 239	1664	1536	0,98

Le tableau 3.2 donne les spécifications du UADSL, le tableau 3.3 les spécifications du ADSL et finalement le tableau 3.4 les spécifications du VDSL.

Tableau 3.3 Les spécifications du ADSL

Nom du bloc	Paramètres	Entrée (en bits)	Sortie (en bits)	Temps d'exécution demandé (ms)
Encodeur Reed-Solomon	GF (2**8) m= 8 $\Rightarrow x^8 + x^4 + x^3 + x^2 + x$ n = 255 t = 8 k = 239	176 mots de 8 bits (1408 bits)	192 mots de 8 bits (1536 bits)	2.14
Entrelaceur	N/A	1536	1536	2.14
Encodeur convolutionnel	K=3 G(0)=101 G(1)=111	1536	3072	2.14
Encodeur QAM	32 symboles à la fois Maximum de 12 bits dans un symbole	384	512	0.27
Table de chargement des bits			32 * 8 bits (256 bits)	0.27
IFFT	32-points	512	512	0.27
FFT	256-points	4096	4096	0.17
Table de chargement des bits			128*8 bits (1024 bits)	0.17
Décodeur QAM	128 symboles à la fois Maximum de 12 bits dans un symbole	2048	1536	0.17
Décodeur Viterbi	K=3 G(0)=101 G(1)=111	3072	1536	0.17
Dé-entrelaceur	N/A	1536	1536	0.17
Décodeur Reed-Solomon	GF (2**8) m= 8 $\Rightarrow x^8 + x^4 + x^3 + x^2 + x$ n = 255 t = 8 k = 239	1536	1408	0.17

Tableau 3.4 Les spécifications du VDSL				
Nom du bloc	**Paramètres**	**Entrée (en bits)**	**Sortie (en bits)**	**Temps d'exécution demandé (ms)**
Encodeur Reed-Solomon	GF (2**8) m= 8 $\Rightarrow x^8 + x^4 + x^3 + x^2 + x$ n = 255 t = 8 k = 239	176 mots de 8 bits (1408 bits)	192 mots de 8 bits (1536 bits)	0.58 ms
Entrelaceur	N/A	1536	1536	0.58
Encodeur convolutionnel	K=3 G(0)=101 G(1)=111	1536	3072	0.58
Encodeur QAM	512 symboles à la fois Maximum de 12 bits dans un symbole	6144	8192	1.17
Table de chargement des bits			512 * 8 bits (4096 bits)	1.17
IFFT	512-points	8192	8192	1.17
FFT	4 transformées de Fourier Rapides de 1024-points	16384	16384	0.41
Table de chargement des bits			4*1024*8 bits (32 768 bits)	0.41
Décodeur QAM	4096 symboles à la fois Maximum de 12 bits dans un symbole	65536	49152	0.41
Décodeur Viterbi	K=3 G(0)=101 G(1)=111	3072	1536	0.05
Dé-entrelaceur	N/A	1536	1536	0.05
Décodeur Reed-Solomon	GF (2**8) m= 8 $\Rightarrow x^8 + x^4 + x^3 + x^2 + x$ n = 255 t = 8 k = 239	1536	1408	0.05

Une fois les spécifications données, nous pouvons maintenant effectuer des estimations pour les différents critères que nous voulons évaluer. Nous pourrons par la suite comparer les estimations obtenues aux valeurs demandées.

Chapitre 4

Méthodologie utilisée et résultats obtenus pour la réalisation d'un modem Universal ADSL.

Dans ce chapitre, nous allons examiner les méthodes que nous avons utilisées pour concevoir ou estimer les différents critères que nous avons expliqués dans le chapitre 2 et les différentes étapes de notre méthodologie. Nous allons tout d'abord présenter les outils qui seront utilisés dans nos expérimentations, puis lors de la présentation des différentes étapes de notre méthodologie, nous expliquerons comment ces outils ont été utilisés.

4.1 Description des outils utilisés

Nous avons utilisé principalement trois outils commerciaux dans notre méthodologie. En voici une introduction, ainsi que leurs inconvénients et leurs avantages.

4.1.1 Monet de Mentor Graphics

Monet est un outil de synthèse comportementale. Il permet d'explorer rapidement différentes alternatives architecturales au niveau comportemental et génère automatiquement du code synthétisable au niveau RTL. Cela permet de réduire d'une façon importante le temps de conception et d'obtenir de bons résultats. En fait, Monet combine une exploration interactive et visuelle d'une description comportementale. À l'intérieur d'une méthodologie de conception pour le matériel comme celle de la figure 4.1, Monet serait utilisé pour valider et sélectionner une architecture, ainsi que pour la transformation du VHDL au niveau RTL.

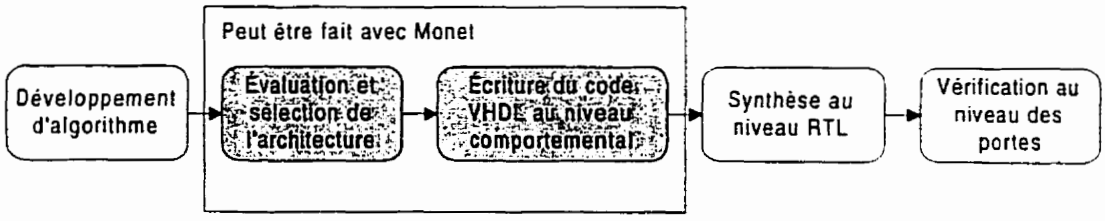


Figure 4.1 Méthode de conception matérielle avec Monet

Monet interagit avec le concepteur pour obtenir les contraintes et certaines directions pour la synthèse comportementale. La figure 4.2 représente les interactions de Monet, ses entrées et ses sorties.

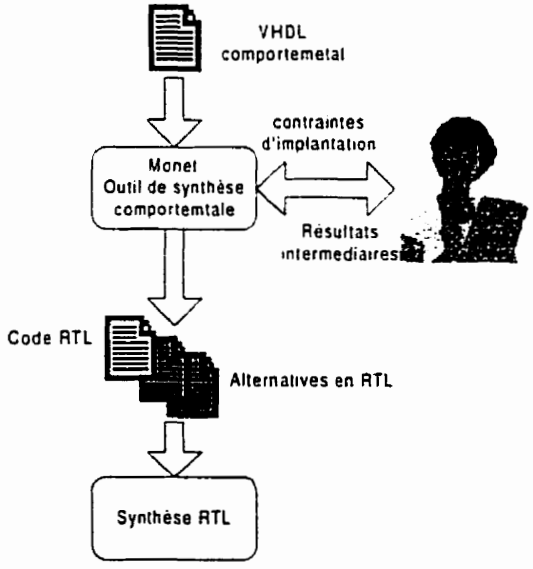


Figure 4.2 Synthèse avec Monet

Les principaux avantages de Monet, sont qu'il :

- Diminue le nombre d'implantation au niveau RTL. Le niveau comportemental permet d'explorer différentes architectures avant de choisir celle qui sera implantée au niveau RTL.

- Diminue la quantité de code à écrire en générant automatiquement du code RTL.
- Offre une interface simple et accessible permettant une meilleur compréhension des différentes étapes d'une synthèse au niveau comportemental.
- Permet une visualisation de l'ordonnancement en utilisant des diagrammes de Gantt, comme il est possible de le voir à la figure 4.3.
- Permet d'obtenir des diagrammes de transitions d'états et des diagrammes bloc des chemins de données.
- Permet de donner un poid aux boucles, influençant ainsi le degré d'optimisation des boucles.
- Offre une plus grande liberté dans le déroulage des boucles, en permettant entre autre de fixer le nombre de déroulement des boucles.
- Permet de synthétiser des tableaux à plus d'une dimension. Les tableaux contenant plusieurs dimensions sont brisés en plusieurs tableaux.
- Permet de synthétiser des blocs plus volumineux que les autres outils de synthèse comportementale connus.
- Permet une plus grande flexibilité dans le code que les autres outils de synthèse connus au niveau comportemental.
- Offre une synthèse plus rapide que les autres outils de synthèse connus au niveau comportemental.

Les principaux désavantages de Monet, sont qu'il :

- Ne peut estimer que le temps et la surface du matériel. Aucune estimation de la dissipation de puissance n'est disponible, et ceci est un élément qui devient de plus en plus important dans les systèmes.
- Ne réussit pas toujours à faire concorder une instruction VHDL et un opérateur provenant de ses bibliothèques. On doit modifier le code de façon à ce que l'outil fasse concorder l'instruction VHDL à un autre opérateur. Par exemple, un

multiplicateur aux dimensions différentes. C'est un problème majeur qui ne se retrouve peut-être pas sur les versions les plus récentes de l'outil.

- Même s'il est plus rapide que les autres outils de synthèse commerciaux, la synthèse d'un bloc contenant plusieurs boucles déroulées peut devenir fastidieuse. Par exemple, la synthèse de deux boucles imbriquées de 16 itérations chacunes prend plus d'une journée afin de passer à travers toutes les étapes de la synthèse comportementale.

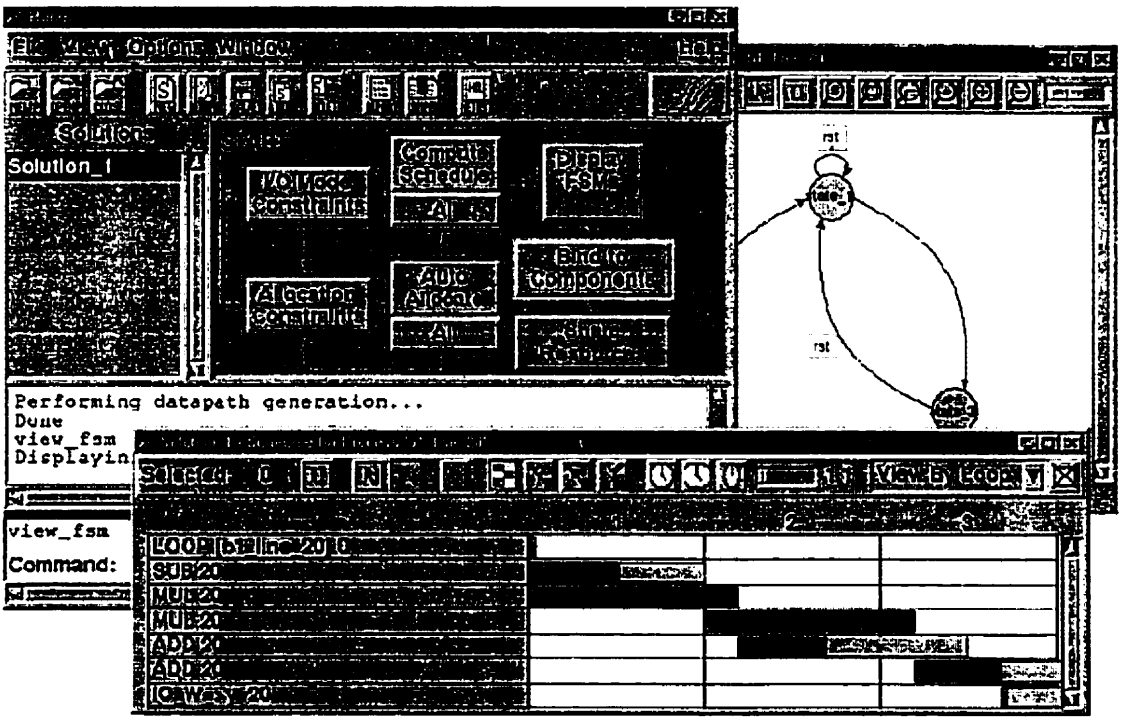


Figure 4.3 Représentation des diagramme de Gantt et d'états dans Monet

- Permet uniquement de pipeliner les boucles et non les composants
- Les appels de fonction sont fait en ligne (inline) uniquement. À chaque appel de fonction, le code de la fonction est répété à chaque fois, ce qui augmente la taille du code.

- Le code VHDL utilisé par Monet doit être compilé à l'extérieur de l'outil par une commande indépendante.
- Les rapports en format texte ne sont pas suffisamment élaborés.
- L'utilisateur doit ajouter un *reset* pour sortir d'un pipeline. Cela ne se fait pas automatiquement.

4.1.2 Description des outils de Synopsys

Nous avons utilisé deux outils de Synopsys qui utilisent une interface commune. Ces outils sont le « Behavioral Compiler » pour la synthèse au niveau comportemental et le « Design Compiler » pour la synthèse au niveau RTL. La figure 4.4 montre les moments de la conception où ces outils sont utilisés. Si nous reprenons la figure 4.1 que nous avons vu précédemment avec Monet, il est possible de voir où se situent ces deux outils de synthèse.

Le « Behavioral Compiler » a des fonctionnalités similaires à Monet. Les deux ont des entrées et sorties similaires, mais elles sont présentées un peu différemment. Nous allons donc donner les avantages et les inconvénients de ces outils.

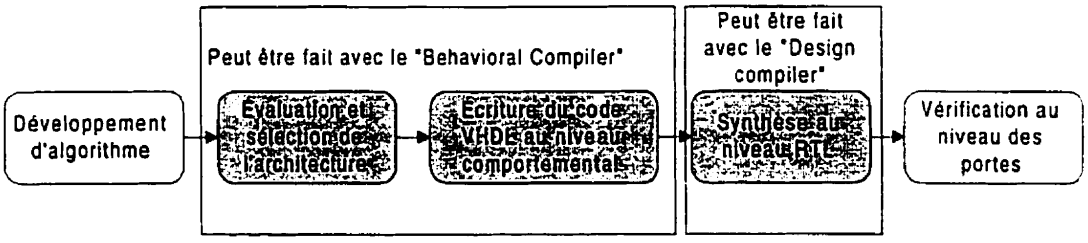


Figure 4.4 Étapes de conception avec les outils de Synopsys

Les avantages des outils de Synopsys sont qu'ils :

- Diminuent le nombre d'implantation au niveau RTL. Le niveau comportemental permet d'explorer différentes architectures avant de choisir celle qui sera implantée au niveau RTL.

- Diminuent la quantité de code à écrire en générant automatiquement du code RTL.
- Une fois le code VHDL au niveau RTL obtenu, il est possible de faire une synthèse et d'estimer la puissance utilisée.
- Font bien concorder les instructions VHDL et les opérations provenant des librairies. Il trouve toujours un composant pouvant être utilisé d'une façon pertinente.
- Permettent de pipeliner des boucles et des composants.
- Peuvent éviter les appels de fontionc en ligne (inline), c'est-à-dire qu'un pragma est utilisé pour indiquer à l'outil que la fonction peut être partagée.
- Permettent de pipeliner les boucles et les composants.
- Compilent le code VHDL à l'intérieur de l'outil. Aucun besoin pour un outil indépendant de compilation.
- Fournissent des rapports en format texte suffisament élaborés
- N'impose pas à l'utilisateur d'ajouter un *reset* pour sortir d'un pipeline. Cela se fait automatiquement.

Les inconvénients des outils de Synopsys sont qu'ils

- N'offrent pas un interface simple et accessible. Il est difficile de savoir dans quel ordre les commandes doivent être faites. La figure 4.5 montre un exemple de l'interface. Contrairement à Monet, il ne supporte pas les diagrammes de Gantt (figure 4.3).
- Les résultats ne sont pas offerts sous forme graphiques, mais uniquement textuelle. (Selon les informations de leur site internet « www.synopsys.com », la dernière version du « Behavioral Compiler » le permettrait).
- Font une synthèse comportementale très lente.

- Sont incapable de synthétiser du code comportant plusieurs boucles. Par exemple, deux boucles imbriquées et déroulées comportant 16 itérations serait trop complexe pour une synthèse dans le « Behavioral Compiler ».
- Les contraintes demandées pour l'écriture du code sont rigides. Par exemple les énoncés d'attente (Wait Statements) doivent être placé d'une manière plus rigoureuse, sinon l'outil est incapable de synthétiser.
- Ne donnent pas de poids aux boucles pour influencer leur degré d'optimisation lors de la synthèse.
- N'offre pas une grande liberté pour le déroulement des boucles.
- Ne permet pas de synthétiser des tableaux à plus d'une dimension.

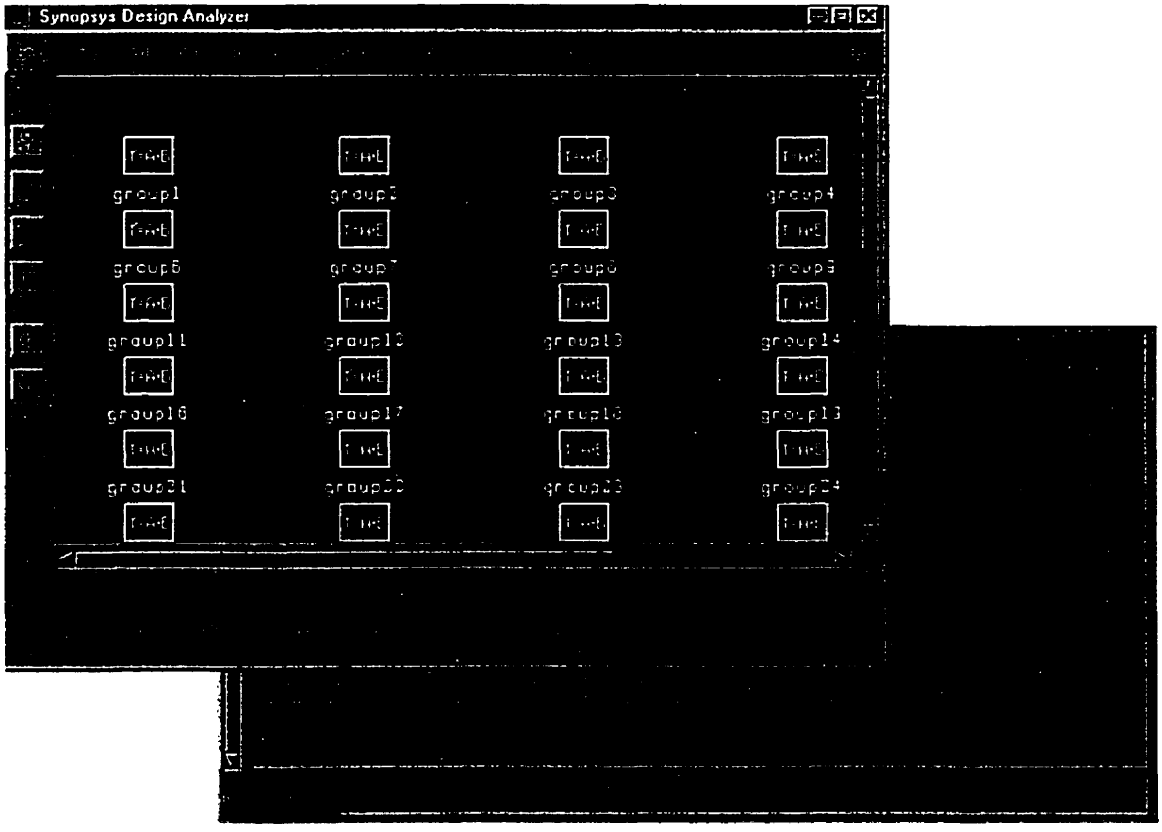


Figure 4.5 L'interface du « Behavioral Compiler » de Synopsys

4.1.3 Seamless de Mentor Graphics

Un des plus grands problèmes dans la conception de systèmes embarqués contenant du logiciel et du matériel est l'intégration de ces deux parties pour former un seul et même système. Dans les méthodologies de codesign matériel/logiciel traditionnelles, le logiciel et le matériel sont développés séparément. Le logiciel ne peut être testé, tant que le matériel n'est pas validé et fonctionnel. Cela peut entraîner des erreurs complexes et longues à isoler et à réparer et cela principalement au niveau des interfaces. La cosimulation peut être utilisée à toutes les étapes de la conception, c'est-à-dire à différents niveaux d'abstraction. Car, il permet de tester le logiciel sur un modèle matériel, même si celui-ci n'est qu'au niveau comportemental ou RTL.

Il est possible de faire de la cosimulation matérielle/logicielle en utilisant uniquement un simulateur matériel avec un modèle fonctionnel du processeur (ISS ou Instruction Set Simulator en anglais). Par contre, les vitesses de simulation sont très faibles et il est difficile de déverminer le logiciel. On pourrait également créer un modèle en C ou en C++ du processeur et le simuler sur le simulateur de jeux d'instructions, mais la précision de ces modèles est discutable [Boll99].

Comme il est possible de le voir dans la figure 4.6, Seamless CVE est composé d'un simulateur de jeu d'instructions qui permet d'exécuter et de déverminer la partie logicielle du système. Il comprend aussi un simulateur logique qui permet d'exécuter et de déverminer la partie matérielle du système. Ces deux simulateurs sont reliés et communiquent ensemble dans Seamless. Les mémoires utilisées sont également définies et configurées à l'intérieur de Seamless CVE, car elles doivent être accessibles autant à partir du simulateur logiciel, que du simulateur logique. En fait, à chaque fois que le processeur a besoin d'exécuter une instruction ou d'accéder à la mémoire, le simulateur de jeu d'instructions envoie une requête par l'intermédiaire de Seamless

CVE vers le simulateur HDL. Le modèle d'interface de bus répond en exécutant les cycles de bus correspondants, l'accès à la mémoire s'effectue à travers le modèle matériel, puis les données ou instructions sont retournées au simulateur de jeu d'instructions.

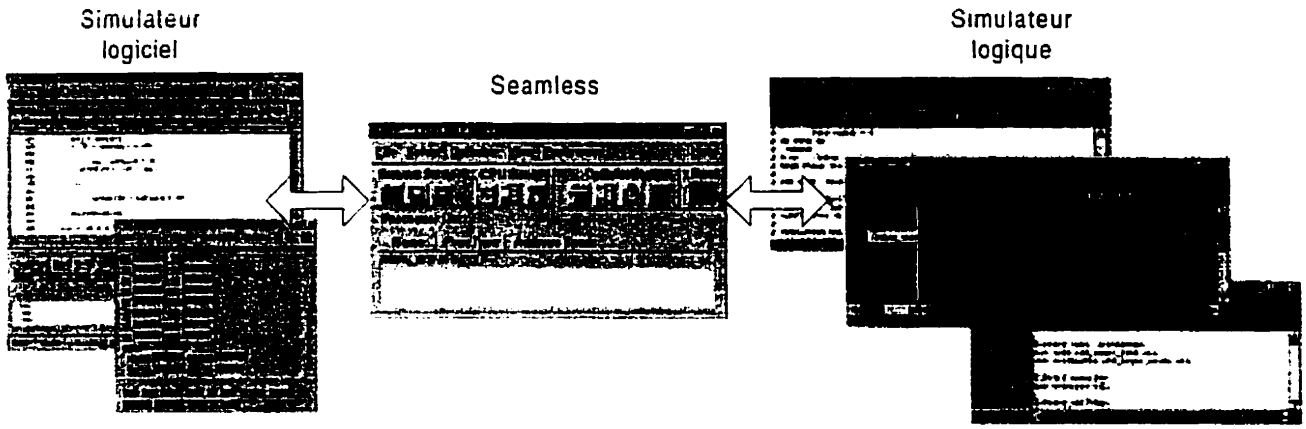


Figure 4.6 Seamless CVE pour la cosimulation matérielle/logicielle

Voici les principaux avantages que nous avons trouvés pour l'utilisation de l'outil Seamless CVE pour co-simuler le matériel et le logiciel à un haut niveau d'abstraction :

- Il réduit le temps de développement des systèmes embarqués;
- Il réduit le nombre d'itérations du prototype matériel;
- Il permet d'exécuter du vrai logiciel sur le design matériel avant que celui-ci soit implanté en silicium.
- Il accélère le déverminage des pilotes des périphériques et des diagnostiques matériels;
- Il ne nécessite pas de modification du système du côté matériel ou logiciel;

- Des optimisations permettent d'atteindre de hautes performances pour la validation du système (temps de simulation);
- Il permet d'offrir un produit de qualité plus rapidement sur le marché;
- Il permet de simuler 10 000 fois plus rapidement qu'avec un simulateur logique;
- Il supporte plus de 35 processeurs et 10 simulateurs logiques;
- Il utilise des simulateurs de façon indépendantes. Donc, en apprenant à utiliser Seamless, on apprend également une méthodologie pour simuler et déverminer à l'aide d'un simulateur de jeux d'instructions et d'un simulateur logique.

Voici les inconvénients de Seamless :

- Il possède un environnement complexe, utilisant deux simulateurs différents et une grande quantité de fenêtre.
- La synchronisation entre les différents simulateurs est difficile. Il est difficile d'en arrêter un et de le repartir sans le désynchroniser avec l'autre.
- Il ne possède pas de point de contrôle ou de redémarrage. Après chaque modification, il faut revenir au point de départ et recharger le projet.
- Il oblige à utiliser les modèles de processeurs et de mémoire fournit par Seamless. La création de nouveaux modèles demande une grande expérience et est très complexe.
- Le changement de processeur demande également un changement de la logique (glue logic) entourant le processeur. Donc, l'interface doit être refaite pour chaque processeur utilisé.

4.2 Description de notre méthodologie

La figure 4.7 représente les différentes étapes de notre méthodologie. Seulement les résultats obtenus avec le Universal ADSL seront donnés dans cette section dans le but contraindre la longueur de ce document. De plus, parmi les trois modems étudiés, il est le plus intéressant au point de vue codesign matériel/logiciel.

4.3 Spécifications du système et définition des blocs

La première étape comporte les spécifications du système. Les temps d'exécution des différents modems ont été donnés dans le chapitre 3. La section 3.7 contient les spécifications au niveau des blocs. Comme on peut le voir à la figure 3.3, nous avons divisé les modems en douze blocs distincts qui représentent chacun une partie de la fonctionnalité du modem. Ceci constituue un partitionnement à gros grains. Chacun de ces blocs a une fonctionnalité distincte. C'est-à-dire que l'exécution d'un bloc ne dépend pas de celle des autres. Ils pourraient être utilisés indépendamment dans une autre application, car à eux seuls, ils effectuent une opération complète. Nous avons choisi ces blocs de façon à ce qu'ils puissent être exécutés en parallèle, de la même façon qu'un pipeline. Par contre, les blocs ne contiennent pas tous la même quantité de comportement. Par exemple, le décodeur Reed-Solomon est beaucoup plus gros (contient une plus grande partie de la fonctionnalité du système) que l'encodeur QAM. Un partitionnement plus fin aurait pu être fait, mais pour une première étude du modem, nous avons préféré utiliser de gros blocs pour diminuer le nombre de blocs à traiter et minimiser les communications.

4.4 Description des blocs pour le logiciel et le matériel

Tout d'abord, pour faire des estimations sur les différents blocs de nos modems, nous avons fait une description matérielle et une description logicielle de tous les blocs. Idéalement, nous aurions utilisé un seul langage pour avoir une description unique de

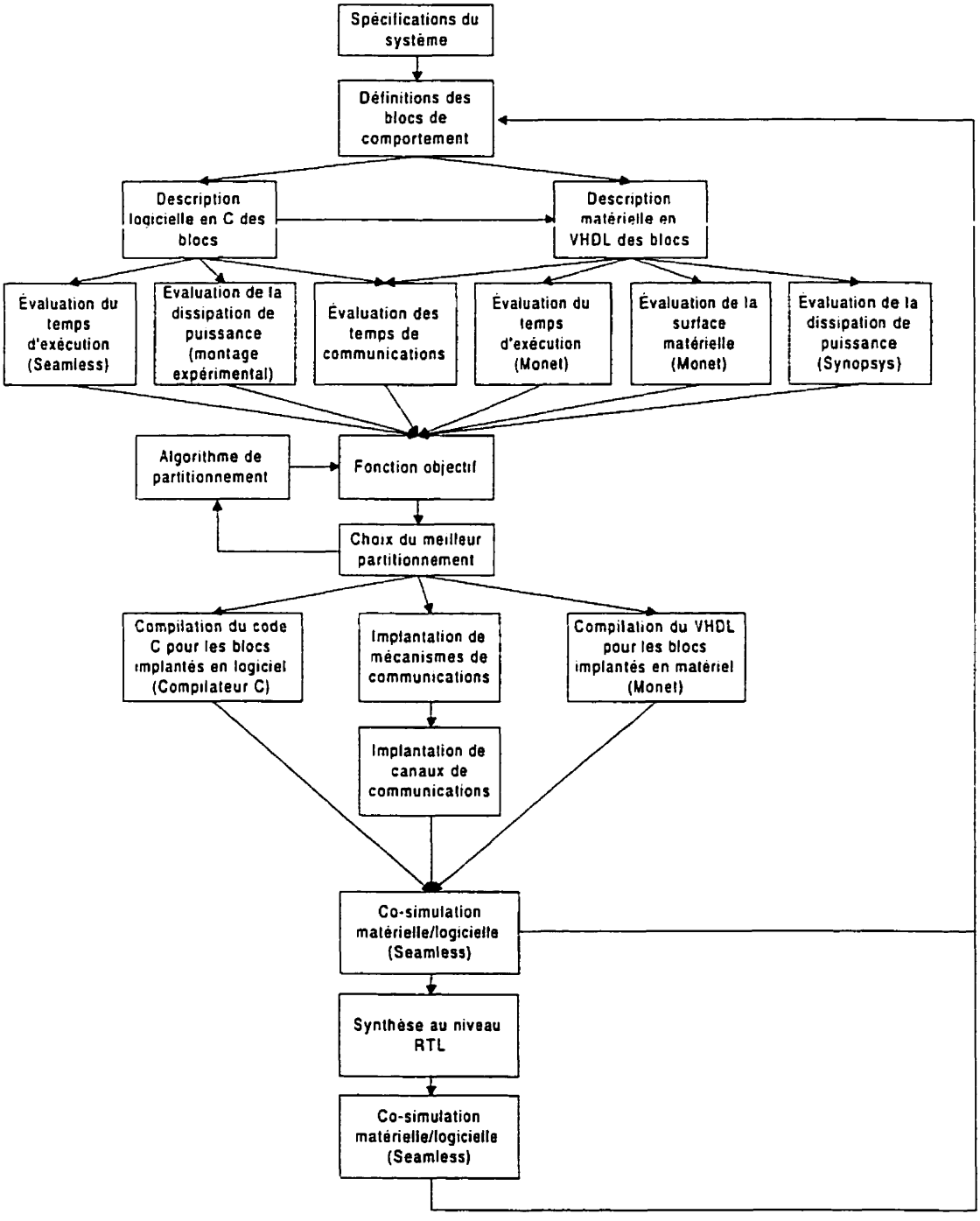


Figure 4.7 Les étapes de notre méthodologie de codesign

nos blocs. Par exemple, nous aurions aimé utiliser le C comme langage de description commun, mais cela aurait nécessité un traducteur, par exemple du C vers le VHDL, ce que nous ne possédions pas. C'est la raison pour laquelle il y a une flèche entre la description logicielle et la description matérielle dans le schéma de la figure 4.7. Comme cela n'était pas disponible, nous avons tout d'abord fait une description des différents algorithmes en C pour évaluer les aspects logiciels, puis cette description a été transformée en VHDL à partir de la description logicielle en C des blocs et des définitions des blocs de comportement, afin d'évaluer les aspects matériels.

4.5 Développement de la dissipation de puissance pour le logiciel

Notre premier choix pour l'implantation des modems de la famille xDSL a été le TMS320C54 de Texas Instruments, à cause de sa faible dissipation de puissance, sa rapidité d'exécution, ses multiples bus (parallèle et HPI) et finalement ses instructions spéciales permettant d'optimiser les blocs FFT, IFFT et le décodeur Viterbi des modems.

Nous n'avons trouvé aucune étude sur la dissipation de puissance pour le processeur C54 de Texas Instruments, ayant une source d'alimentation de 5 V. Nous avons donc décidé d'élaborer une méthode d'estimation de dissipation de puissance, en utilisant les modèles de puissance au niveau des instructions, comme il a été expliqué dans la section 1.4.1. Ces expériences d'abord élaborées par Tiwari et al. [TMW94] ont été par la suite reprise pour différents processeurs. Nous avons donc voulu utiliser cette méthodologie avec le TMS320C54, dans le but de déterminer la fiabilité de cette technique et de découvrir les dépendances des différentes instructions sur la dissipation de puissance. Une partie de ce travail a été effectuée par un stagiaire, Gwénaël Poitau provenant de ISTASE (Institut supérieur des techniques avancées de Saint-Étienne) [Poit99], sur une carte d'évaluation du TMS320C54 de la compagnie Texas Instrument.

4.5.1 Dissipation de puissance pour chacune des instructions

Nous allons tout d'abord donner et expliquer les résultats qu'il a obtenu, puis la suite des expérimentations effectuées. Les mesures des coûts de base des instructions ont été réalisées à l'aide du montage de la figure 4.8, en utilisant une carte d'évaluation DSKPLUS de Texas Instruments, intégrant un TMS320C542 fonctionnant à 40 MHz, à l'aide d'une alimentation programmable Hewlett-Packard 6623A fournissant 5 V et un courant variable. Les mesures ont été effectuées à la température ambiante (environ 20° C).

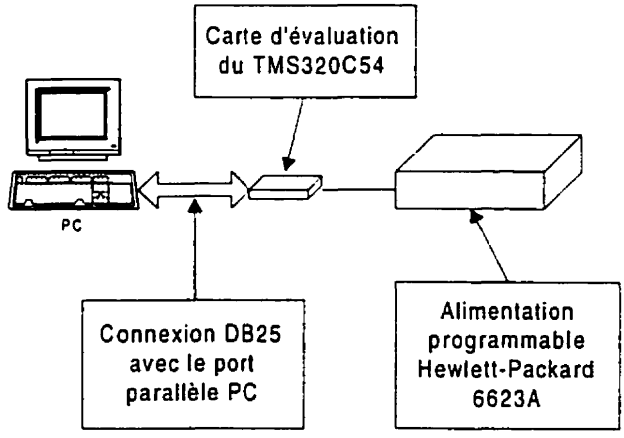


Figure 4.8 Montage pour mesures expérimentales

Le problème sur cette carte est que l'alimentation ne fournit pas uniquement le processeur, mais également ses périphériques. Nous avons donc testé le courant utilisé par la carte lorsque le processeur était en mode Idle3. Cela signifie que le CPU, ainsi que les périphériques internes sont arrêtés. Il est donc possible dans ce mode de calculer le courant utilisé par les périphériques extérieurs au processeur sur la carte, lorsque le processeur est arrêté. Il est bon de préciser que les périphériques de la carte externes au processeur ne sont pas utilisés lors de l'exécution des différents programmes. Lorsque la carte est dans le mode Idle3, un courant de 0.179 ampère

circule. Donc, à chaque fois que nous allons prendre des mesures sur la carte, 0.179 A devront être soustrait du courant obtenu pour calculer uniquement le courant utilisé par la carte. Il est bon de remarquer qu'une fois le courant connu, il est facile de déterminer la puissance *(P)* dissipée en multipliant le courant *(I)* par le voltage *(V)* fournit au processeur.

$$P = V I \tag{4-1}$$

Toutes les mesures sont effectuées deux fois. La première dans le mode « Repeat », c'est-à-dire qu'une boucle est utilisée pour répéter la même instruction plusieurs fois. Le deuxième mode utilisé est le mode « Inline ». Dans ce mode, les instructions sont écrites plusieurs fois sans utiliser une boucle. Par exemple, si nous voulons exécuter une instruction 120 fois, elle sera écrite 120 fois. On peut trouver des exemples de routines de test en assembleur algébrique pour le C54, ainsi que les résultats pour les différentes instructions en annexe D. Le tableau 4.1 montre le coût de base de certaines instructions.

Tableau 4.1 Coût de base des instructions

Instructions	Mode « Repeat » (mA)	Mode « Inline » (mA)	Commentaires
Abs	66	91	A = |a|
Cmps	77	102	Cmps(a,*ar3)
Add	67	94	A=a+#01h
Mul	87	107	A=a+*ar5+*#1234h
Or	77	103	A=a|*ar3
Load	66	82	B=#248

4.5.2 Les dépendances de la dissipation de puissance

Après avoir étudié la consommation de puissance de chacune des instructions et constitué un modèle de premier ordre, nous donnons les différents facteurs que nous avons mis en évidence et qui influencent cette consommation de puissance.

4.5.2.1 Dépendance vis-à-vis des données

Le tableau 4.2, donne le courant utilisé par le processeur lors de l'exécution de différentes instructions ayant des opérandes différentes. En fait, les dépendances de données sont examinées. Nous pouvons alors constater que pour certaines opérations, il y a une dépendance de données sur la consommation de puissance. Les calculs ont été faits en mode « Repeat ». Les valeurs dans le tableau sont en fait les différentes données utilisées pour tester une instruction.

Tableau 4.2 Dépendances des données des instructions

Instuctions	Courant pour 5555h (mA)	Courant pour aaaah (mA)	Courant pour 0h (mA)	Courant pour 1234h (mA)
Abs	65	65	65	65
Add (A)	79	79	79	79
And (A)	77	77	77	77
Firs	63	63	110	99
Or (A)	77	77	77	77
Square (A)	66	66	65	66
Sub (#lk)	71	70	66	69

C'est cette dépendance qui peut entraîner le plus d'incertitude à l'intérieur d'une estimation utilisant le coût de base de chacune des instructions. Cela est dû au fait que si nous utilisons un programme en temps réel, les données ne sont pas en général connues à l'avance ou bien elles sont trop nombreuses pour vraiment en tenir compte. À ce moment-là, le meilleur moyen de fonctionner est d'inclure une incertitude dans le calcul de la dissipation de puissance. Pour arriver à effectuer une bonne estimation de la variation selon les données, nous aurions besoin de connaître avec exactitude l'architecture interne du processeur et la façon dont les instructions sont microprogrammées.

4.5.2.2 Dépendance inter-instructions

Lorsque deux instructions sont exécutées l'une à la suite de l'autre, la consommation du processeur est généralement différente de la moyenne des consommations des deux instructions prises séparément et pondérées par le nombre de cycles nécessaires à leurs exécutions. En effet, la commutation des circuits est en fonction des entrées actuelles et de l'état précédent du circuit, ainsi la consommation de puissance sera en fonction de l'instruction précédemment exécutée. Le tableau 4.3 contient certains des résultats obtenus démontrant les effets inter-instructions. Un tableau plus complet est disponible en annexe D. La moyenne des consommations pondérée est en fait le courant moyen lors de l'exécution des 2 instructions à l'intérieur d'une boucle. La consommation réelle est en fait l'addition du courant utilisé par les deux instructions de façon indépendante.

Tableau 4.3 Les dépendances inter-instructions

Première instruction	Seconde instruction	Moyenne des consommations pondérée (mA)	Consommation réelle (mA)	Écart (%)
Abs (91 mA)	Cmps (102 mA)	96.5	111	13
	Add (94 mA)	93	102	9
	Prog (90 mA)	90.5	91	1
Max (90 mA)	Sub (107 mA)	101	101	0
	Sat (64 mA)	90	108	17
	Test (110 mA)	100	111	10

4.5.3 Classes d'instructions

Les consommations de puissance des instructions ont été placées en ordre croissant dans le graphique de la figure 4.9. Les courbes du graphique présentent 3 paliers chacun, à 65, 80, 90 mA pour le mode RPT, à 90, 105, 115 mA (paliers du mode REPEAT majorés de 25 mA) pour le mode INLINE. Si l'on s'attache à la signification de ces paliers (mode REPEAT), on observe que le premier palier à 65 mA correspond à

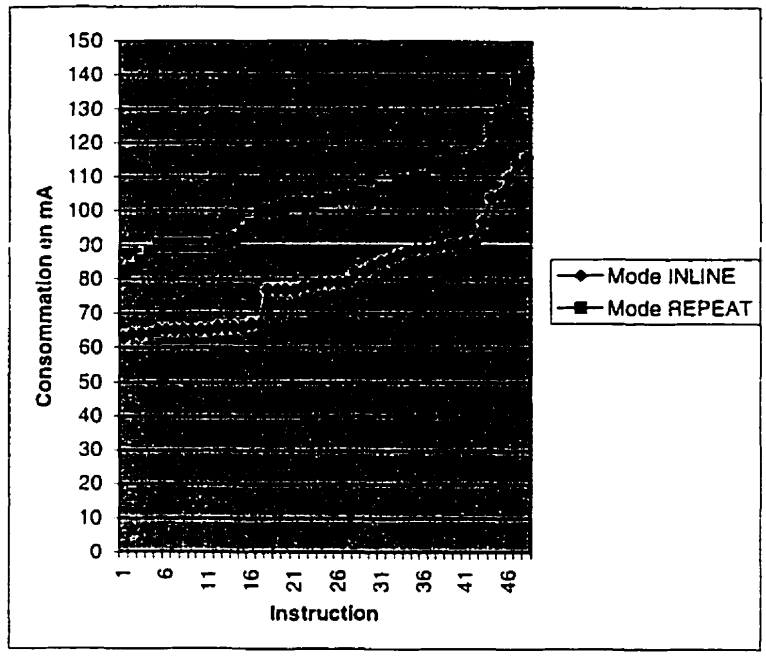


Figure 4.9 Consommation de puissance des différentes instructions en ordre croissant

une instruction simple, le palier à 80 mA correspond aux instructions utilisant un adressage indirect (utilisation d'un registre arx), enfin le palier à 90 mA correspond à des instructions comportant une post-incrémentation ou une post-décrémentation d'un registre. Cette analyse est vérifiée pour les opérations arithmétiques, logiques et les fonctions de chargement et stockage. Les fonctions de multiplication, les fonctions spéciales de multiplication et les fonctions complexes (multiples) sont à analyser indépendamment et aucune corrélation n'a été trouvée pour l'instant. Nous proposons donc le système de la figure 4.10.

Pour l'instant, le système de classes est limité aux effets de premier ordre, nous n'avons pas trouvé de corrélation entre les effets inter-instructions et l'appartenance à une classe.

	Consommation de classe en mA (mode RPT)	Consommation de classe en mA (mode INLINE)
Instructions simples	65	90
Instructions utilisant un adressage indirect	80	105
Instructions utilisant une post-incrémentation ou une post-décrémentation du registre	90	115
Fonctions de multiplication, fonctions spéciales de multiplication, fonctions complexes (multiple)	À analyser indépendamment en fonction des consommations données en annexe D.	

Figure 4.10 Système de classes de consommation

4.5.4 Calcul de la dissipation de puissance pour un programme

Nous allons utiliser les coûts de base des instructions (section 4.5.1) afin de déterminer la dissipation de puissance d'un programme. Nous ne tiendrons pas compte des effets inter-instructions et des dépendances de données. Ces aspects pourraient être évalués dans des travaux ultérieurs. Nous devons être capables de déterminer le courant à un temps précis et non seulement de prendre une valeur moyenne obtenue dans une certaine période de temps. Pour ce faire, nous avons utilisé le montage de la figure 4.11. Une résistance est placée entre la source de 5 V et la carte DSKPLUS contenant le processeur TMS320C542. Cette résistance a une valeur de 0.1 Ω. Le voltage est ensuite mesuré aux bornes de la résistance afin de déterminer le courant traversant celle-ci. Le voltage étant trop petit pour vraiment en voir les variations sur l'oscilloscope, nous y avons ajouté une amplification de 23 fois. La figure 4.12 montre les éléments de l'amplificateur. L'amplification a été faite avec un amplificateur opérationnel TL084 de Texas Instruments. La bande passante maximale de cet amplificateur opérationnel est

de 4 Mhz. Le processeur fonctionne à une fréquence beaucoup plus élevée, donc les résultats obtenus sont fortement filtrés par un filtre passe-bas.

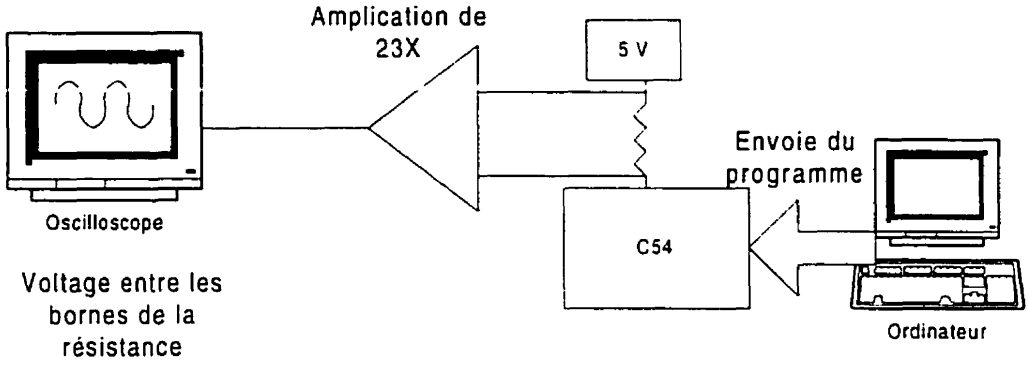


Figure 4.11 Montage pour le calcul de la dissipation de puissance

En obtenant le voltage aux bornes de la résistance, il est alors possible en divisant cette valeur par la résistance d'obtenir le courant qui y passe et qui est également le courant à l'entrée du processeur. En utilisant l'équation :

$$V=RI, \tag{4-2}$$

où V est le voltage, R la résistance et I le courant passant dans la résistance. Le voltage est trouvé en calculant l'aire sous la courbe. Comme l'aire sous la courbe est en fonction du temps, la valeur que nous trouvons en multipliant cette valeur par le voltage de la carte est l'énergie et non la puissance. Car l'équation pour trouver l'énergie est :

$$E=Pt, \tag{4-3}$$

où E est l'énergie, P la puissance et t le temps. Il ne nous reste qu'à diviser la valeur obtenue par le temps d'exécution du programme pour obtenir la dissipation de puissance

moyenne. La figure 4.13 montre un exemple d'une FFT de 1024-points placée dans une boucle. Quatre itérations sont montrées dans la figure.

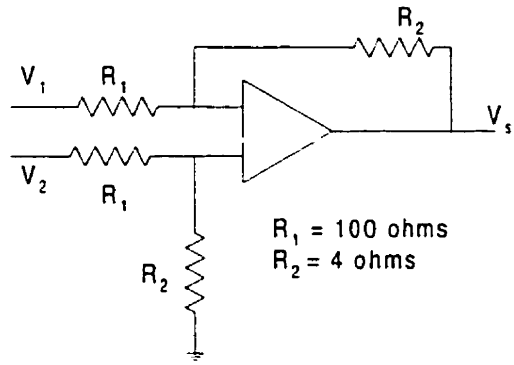


Figure 4.12 Amplificateur 23 X du montage

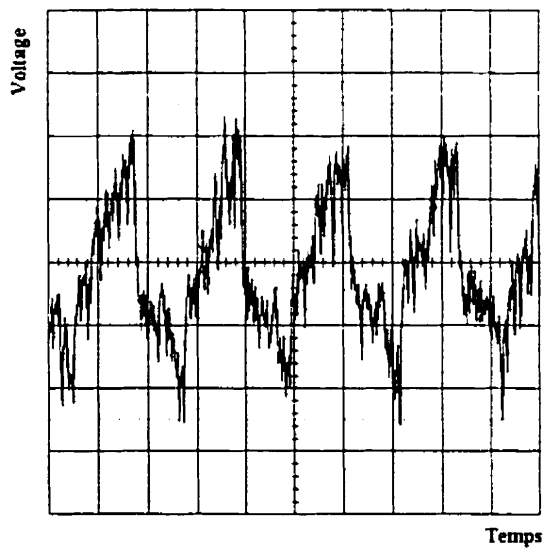


Figure 4.13 FFT de 1024-points dans une boucle

Le tableau 4.4 montre les résultats de l'expérimentation, ainsi que les résultats obtenus en additionnant le coût de base de chacune des instructions des programmes exécutés. Ces résultats ne tiennent pas compte des dépendances de données ou d'instructions. On trouve les programmes testés en annexe E.

Tableau 4.4 Comparaisons des résultats et des coûts de base			
Programmes testés	**Résultats avec oscilloscope (W)**	**Résultats avec estimations (W)**	**Différence**
Test1.asm	5,3	4,8	9 %
Test2.asm	3,9	4,3	9 %
Test3.asm	5,2	5,1	2 %
Test4.asm	5,9	6,3	6 %
Test5.asm	4,6	5,3	13 %
Test6.asm	4,2	4,8	12 %
Test7.asm	4.8	5,8	17 %
Test8.asm	4.2	5,0	10 %
Test9.asm	5.0	4,7	6 %
Test10.asm	3.1	3,8	18 %

Dans nos calculs, nous n'avons pas tenu compte des différentes dépendances. Donc, nos résultats sont probablement un peu moins précis que si nous en avions tenu compte. Comme nous pouvons le voir dans le tableau 4.4, les différences relatives entre les puissances peuvent varier de 2 à 18%, ce qui n'est pas très précis. Par contre, cela donne quand même un ordre de grandeur de la dissipation de puissance d'un programme. L'utilisation de cette méthode pour estimer la dissipation de puissance pourrait être utile si nous voulons comparer la dissipation de puissance de deux modules ayant la même fonctionnalité.

Il y aurait également la possibilité de diminuer la dissipation de puissance d'un programme en choisissant des instructions moins coûteuses en dissipation de puissance. Par exemple, des adressages directs sont à privilégier par rapport aux adressages utilisant une post-incrémentation.

4.6 Estimation du temps d'exécution logiciel

Comme nous l'avons mentionné précédemment, notre premier choix de processeur était le TMS320C54. Pour les étapes suivantes, nous avions besoin d'un modèle du processeur en VHDL, afin d'utiliser l'outil de cosimulation matériel/logiciel

Seamless. Comme nous n'avons pu obtenir ce modèle pour le TMS320C54, nous avons donc opté pour un autre processeur le ARM7TDMI. Par contre, nous aurions aimé reprendre les expérimentations sur la dissipation de puissance avec ce nouveau processeur. Mais nous n'avons pu avoir de carte matérielle pour ce processeur. Pour les étapes suivantes, nous avons donc travaillé avec le ARM7TDMI qui avait quand même des caractéristiques intéressantes. L'architecture ARM7TDMI intègre le jeu d'instructions ARM 32 bits et le jeu d'instructions 16 bits Thumb, un pipeline d'exécution de trois étages qui reste masquée pour le logiciel d'application. Elle offre également des instructions d'un seul cycle, des instructions de branchement et d'accès mémoire de deux cycles, des accès par octets, par demi-mots ou par mots, les modes d'interruptions rapides et une conception entièrement statique pour réduire la consommation [Boll99].

Pour estimer le temps d'exécution des différents blocs logiciels, nous utilisons une technique que l'on pourrait situer entre le profilage dynamique et le profilage statique. Tout d'abord, il est important de mentionner que le profilage s'est fait en utilisant un modèle d'estimation pour processeur spécifique, comme il a été présenté dans la section 2.4.1.1.

Nous avons manuellement fait un partitionnement de chacun de nos douze blocs en les divisant en de plus petits blocs. Ces petits blocs sont appelés blocs de base. Ces blocs de base sont bornés par le début et la fin d'une boucle ou d'un branchement conditionnel. Nous avons pour chaque boucle déterminé à l'aide des spécifications de l'algorithme le nombre de fois maximal où la boucle pouvait être exécutée. Par exemple, si nous prenons l'encodeur QAM, la boucle principale devait être exécutée seize fois, car seize symboles doivent être encodés pour rencontrer les demandes en taux de traitement de l'algorithme. Donc, nous avons exécuté de façon dynamique le corps de la boucle en y envoyant différentes données, choisies de façon aléatoire. Une fois le

temps d'exécution d'une boucle obtenue, nous avons multiplié ce temps par le nombre de fois où la boucle est exécutée, c'est-à-dire par seize. Pour les branchements conditionnels, nous avons fonctionné un peu différemment. Nous avons calculé le temps d'exécution des blocs de tous les branchements possibles et le temps du branchement le plus lent a été utilisé. Par exemple, si nous prenons à nouveau l'exemple de l'encodeur QAM. Chacun des symboles de l'encodeur utilisant la technologie DMT peut contenir de 0 à 12 bits. Nous avons donc 13 possibilités différentes pour l'encodage d'un symbole. En code VHDL, ce branchement conditionnel est représenté par une boucle *if* et des boucle *elsif*. Une seule de ces 13 conditions sera choisie à la fois. Les instructions examinent en premier la condition pour encoder 13 bits, puis 12 bits et ainsi de suite jusqu'à 0 bit. Pour calculer le temps d'exécution de ces blocs, nous avons calculé le temps d'exécution des 13 blocs pour ainsi déterminer que le bloc le plus long était celui encodant un symbole de 2 bits. La raison à cela est que nous devons passer à travers la logique de contrôle de tous les autres blocs avant d'obtenir la bonne condition et que son temps de traitement est plus long que celui de l'encodage des symboles de 1 et 0 bit. Donc, le temps de calcul maximal serait obtenu si tous les symboles étaient codés à partir de 2 bits. Ce temps sera alors multiplié par le nombre de symboles à traiter pour obtenir le temps d'exécution total du bloc.

Une fois le temps d'exécution de chacun des blocs connu, il est possible en les additionnant d'avoir une bonne approximation du temps d'exécution totale de l'algorithme (bloc du système). Pour obtenir le temps d'exécution des blocs de base, nous avons utilisé un outil qui au départ a été créé pour faire de la cosimulation matérielle/logicielle, de Mentor Graphics. Le fonctionnement de cet outil a été expliqué à la section 4.1.3. Pour ce faire, nous avons exécuté les différents blocs de base sur le modèle fonctionnel du processeur, puis à la fin de l'exécution du bloc, nous avons regardé sur la trace du circuit logique pour obtenir le temps d'exécution du bloc. Nous

avons utilisé la carte virtuelle fournie par Mentor Graphics pour le ARM7. Ce processeur peut fonctionner jusqu'à une vitesse de 25 MHz, mais sur la carte virtuelle de la compagnie, il était installé pour fonctionner à une vitesse de 2,5 MHz. À cause des nombreux problèmes que nous avions lors de la simulation, nous n'avons pas modifié la carte virtuelle. Le problème était au niveau du modèle fonctionnel du processeur. La compilation se faisait correctement, mais lors de l'exécution, l'instruction en code assembleur « load register » n'était pas exécutée correctement. En fait, un déplacement d'un octet se produisait vers la droite après avoir placé la bonne valeur dans le registre concerné. Nous avons quand même pu obtenir des résultats en contournant le problème et en divisant nos programmes en plus petits blocs.

Le programme peut être exécuté et testé dans cet environnement, de la même façon qu'il peut l'être avec n'importe quel compilateur commercial. Le code a d'abord été développé en utilisant Microsoft Visual Studio 6.0, puis quelques modifications ont été apportées pour exécuter le programme des différents blocs sur le modèle fonctionnel du ARM7.

Nous n'avons pas utilisé de système d'exploitation, ni d'algorithme d'ordonnancement logiciel. Chacun des blocs était un programme qui était exécuté d'un façon séquentielle d'un bout à l'autre sans interruption, ni changements de contextes. Vous trouverez les estimations des temps d'exécution logiciels pour les différents blocs dans le tableau 4.5.

Si l'on compare les résultats du tableau 4.5 avec les contraintes demandées dans le tableau 3.2, nous pouvons voir que seul deux blocs rencontrent les contraintes de temps lorsqu'ils sont placés dans une partition logicielle, dans ce cas-ci sur le ARM7.

Tableau 4.5 Temps d'exécution des blocs logiciels					
# du bloc	**Nom du bloc**	**Temps d'exécution**	**# du bloc**	**Nom du bloc**	**Temps d'exécution**
1	Encodeur RS	0,0359 s	7	FFT (128-points)	0,0419 s
2	Entrelaceur	0,0143 s	8	Table d'alloc. des bits	0,0223 s
3	Encodeur Treillis	0,3098 s	9	Décodeur QAM 128-points	0,0119 s
4	Encodeur QAM	$0,7328 \times 10^{-4}$ s	10	Décodeur Viterbi	1,1050 s
5	Table d'alloc. des bits	$0,2621 \times 10^{-4}$ s	11	De-entrelaceur	0,0148 s
6	IFFT (16-points)	0,0013 ns	12	Décodeur RS	1,620 s

4.7 Estimation de la surface et du temps d'exécution du matériel

Pour faire une estimation de la surface du circuit intégré, ainsi que le temps d'exécution du matériel, nous avons utilisé une technique de synthèse rapide (section 1.7.1.) Puisqu'il existe des outils commerciaux permettant de faire des synthèses au niveau comportemental, nous n'avons donc pas eu à développer nos propres algorithmes et outils. Nous avons utilisé Monet de Mentor Graphics (section 4.1) pour faire la synthèse de nos circuits. Nous allons maintenant examiner les différentes étapes de la synthèse en utilisant Monet.

Il faut d'abord ouvrir un fichier VHDL, compilé comme présenté à la figure 4.14. À l'intérieur de nos blocs, nous avons utilisé des mémoires pour conserver les variables plutôt que des registres. Les bibliothèques contenant les mémoires, ainsi que la technologie utilisée doivent être incluses dans ce fichier d'initialisation. Ces bibliothèques seront par la suite utilisées pour déterminer la surface et le temps d'exécution du matériel. Nous avons utilisé les bibliothèques *mgc_lca300k_dmag_dc.lib* et *mgc_lca300k_comp_dc.lib* pour le choix de la technologie et les bibliothèques *ram_singleport_lib* et *ram_mgc.lib* pour les mémoires.

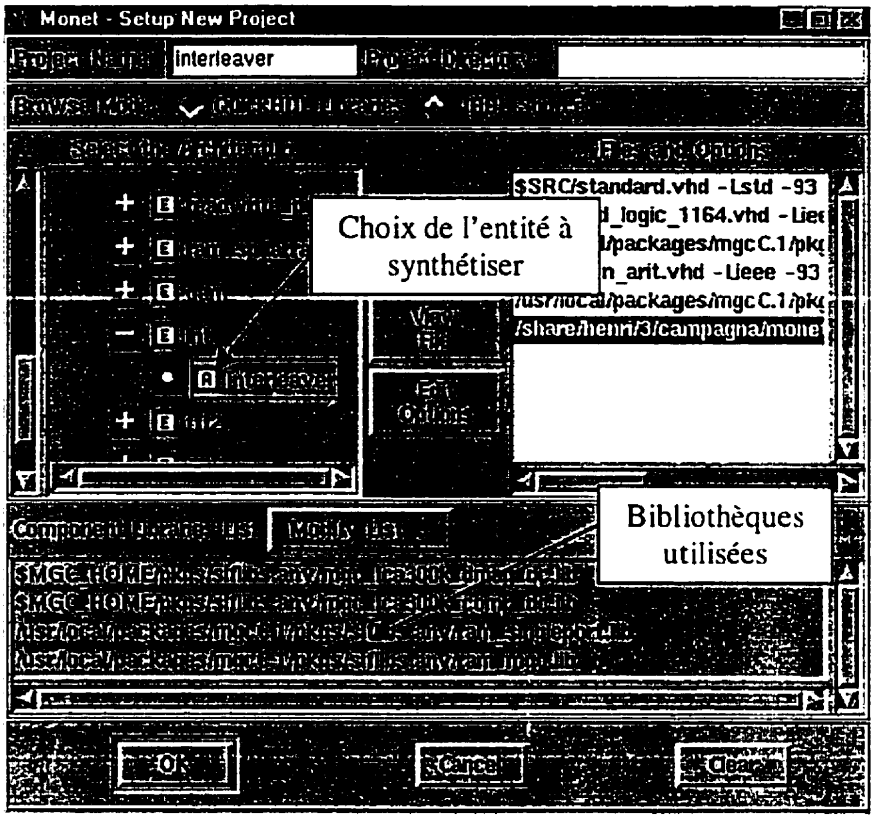


Figure 4.14 Sélection du code à synthétiser et des bibliothèques.

Il est facile de déterminer les différentes étapes en regardant l'interface de Monet (figure 4.15). Chacune des étapes est représentée par un rectangle et les flèches entre les rectangles montrent l'ordre dans lequel sont exécutées les différentes étapes. Par exemple, il est possible de voir que l'ordonnancement se fait après l'allocation.

La première étape de la synthèse est le choix des modes pour les entrées et sorties. Il existe trois modes qui sont « Cycle-Fixed », « Superstate » et « Free ». Le mode « Cycle-Fixed » permet d'avoir une concordance exacte cycle par cycle avec la simulation de pré-synthèse. Le mode « Superstate » permet à l'outil d'ajouter des énoncés d'attente (Wait statement) aux super états et finalement le mode « Free » permet à Monet d'ajouter d'enlever et de déplacer les énoncés d'attente comme il le

désire. Nous avons fait nos estimations en utilisant le mode Free, ce qui nous permettait une plus grande flexibilité dans la façon d'écrire notre code VHDL au niveau comportemental.

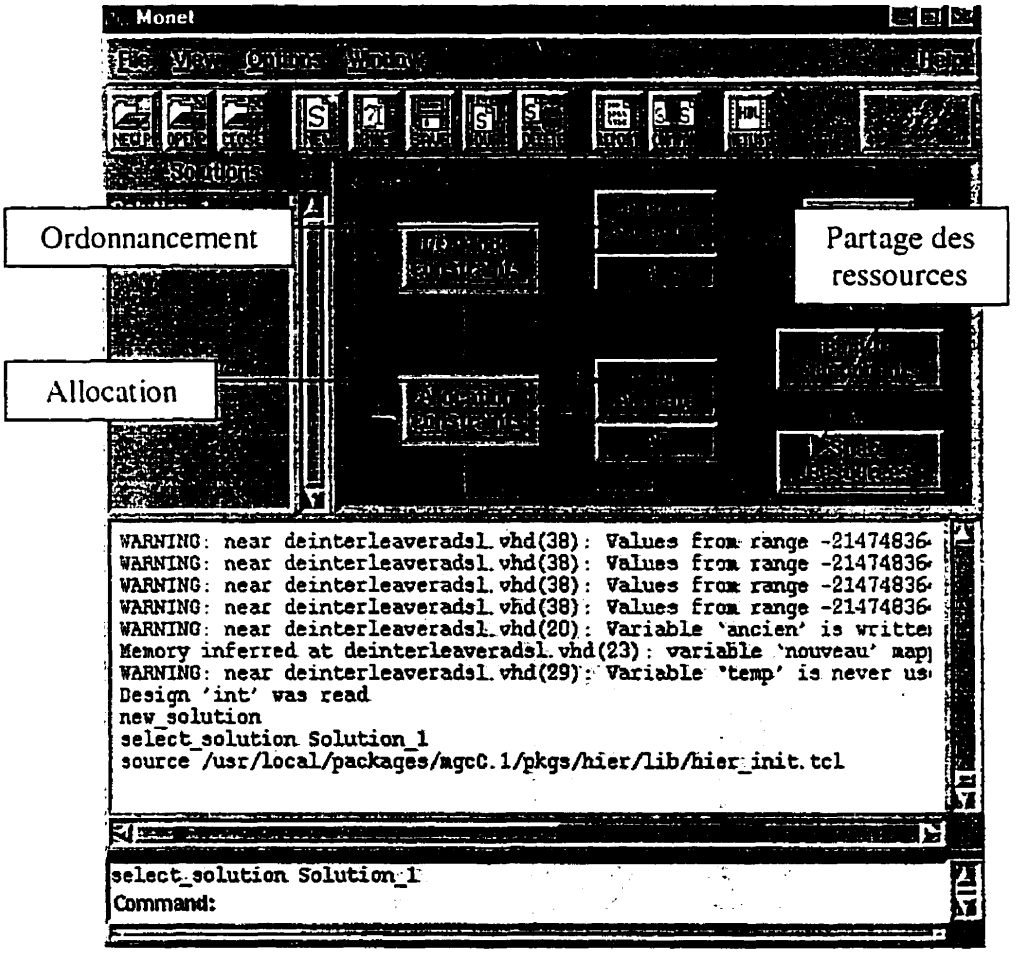


Figure 4.15 Différentes étapes de la synthèse avec Monet

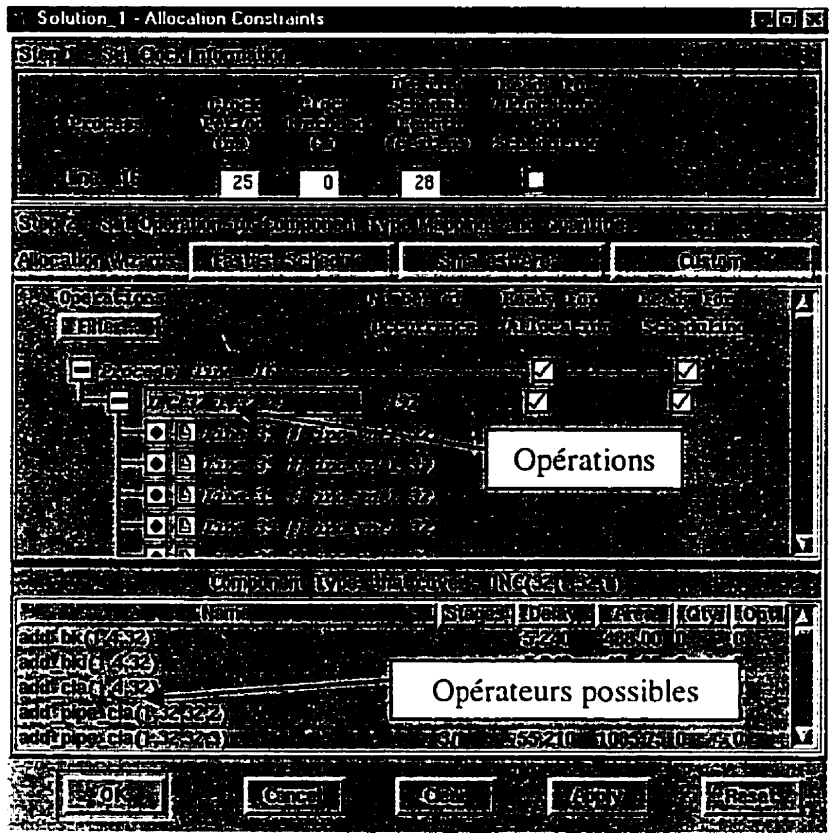


Figure 4.16 Contraintes d'allocation

Comme nous étions certains de réussir à respecter nos contraintes de temps en matériel, nous avons fait les synthèses en optimisant pour minimiser la surface du circuit intégré (figure 4.16). Nous avons utilisé une horloge de 25 ns, ce qui correspond à 40 MHz. La longueur de l'ordonnancement désiré a été placé initialement à 0, puis l'option « Smallest Area » permettant d'optimiser la surface du matériel a été sélectionnée. Chacune des opérations est alors allouée à un opérateur provenant de la bibliothèque que nous avons indiqué au début, comme il a été expliqué à la section 2.5.1.2. Il est possible de laisser l'outil déterminer lui-même les opérateurs correspondants, mais il est également possible pour le concepteur d'indiquer lui-même à l'outil les opérateurs à utiliser parmi un certain nombre d'opérateurs possibles.

Après avoir établi les contraintes de l'allocation, cette dernière peut être effectuée. L'outil donne alors une estimation du nombre d'étapes de contrôle et la surface utilisée par le circuit intégré, comme il est possible de le voir à la figure 4.17. Une description plus détaillée des allocations effectuées peut être obtenue.

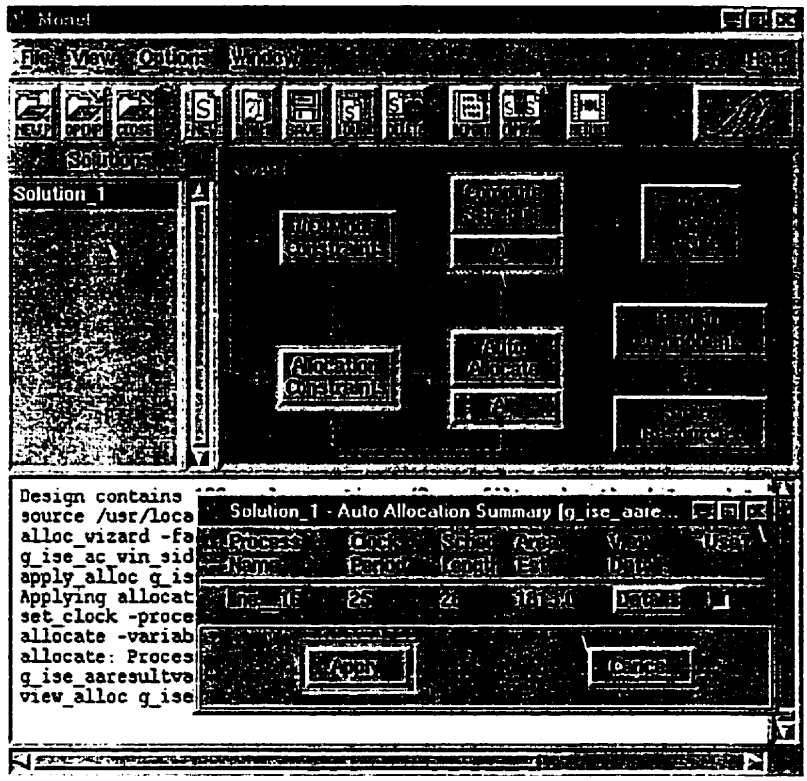


Figure 4.17 Allocation avec Monet

La prochaine étape à suivre en regardant le flot de Monet est l'ordonnancement. Tout dépendant des contraintes données précédemment, l'algorithme utilisé pour l'ordonnancement est différent. Par exemple si lors de la détermination des contraintes pour l'allocation, « Fastest Schedule » c'est-à-dire l'ordonnancement le plus rapide a été sélectionné, un algorithme avec contraintes de temps sera utilisé. Par contre, si « Smallest Area » c'est-à-dire la surface la plus petite a été sélectionnée, alors un algorithme avec contraintes de ressources sera utilisé. Ces différents algorithmes (*Basé*

sur une liste, liste statique, utilisation des opérateurs) ont été décrits dans la section 2.5.1.4. Les algorithmes utilisés par l'outil Monet ne sont évidemment pas disponibles, mais nous croyons qu'un algorithme similaire à « Force dirigée » aurait pu être utilisé pour les ordonnancements avec contraintes de temps et un algorithme similaire à l'algorithme « Basé sur une liste » aurait pu être utilisé pour l'ordonnancement avec contraintes de ressources.

Une fois l'ordonnancement fait, un diagramme de Gantt est disponible, comme le montre la figure 4.18, permettant de visualiser les différentes dépendances entre les opérations. Directement à partir du diagramme (d'une manière graphique), il est possible d'ajouter des contraintes d'ordonnancement et d'effectuer un nouvel ordonnancement.

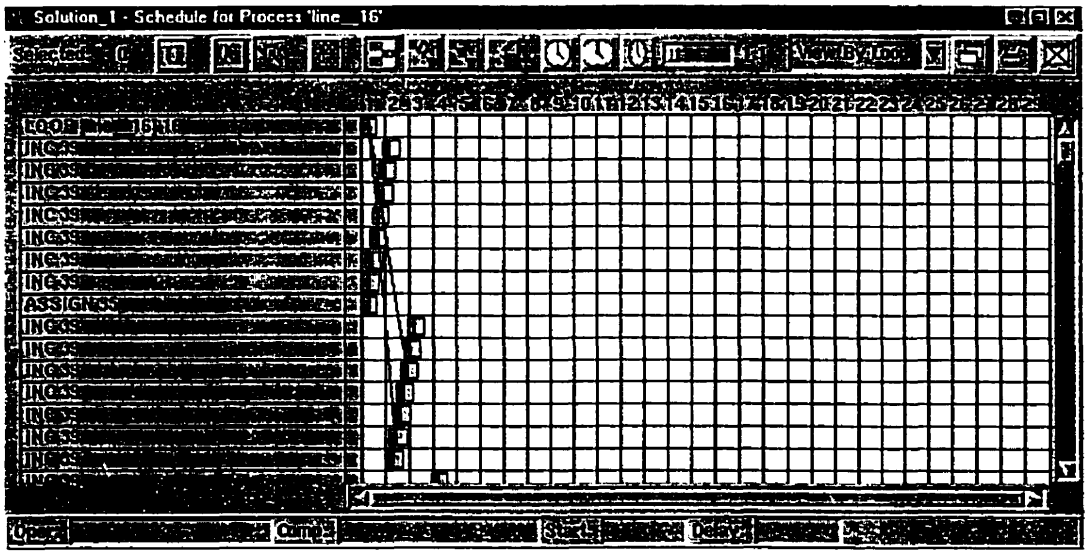


Figure 4.18 Diagramme de Kanttt pour visualiser l'ordonnancement

Les étapes suivantes sur le flot de Monet sont utilisées pour l'extraction de la machine à états (contrôleur) du système et pour l'extraction du chemin de données. Finalement la dernière étape de la méthodologie est le partage des ressources. Un

algorithme similaire à ceux décrit dans la section 2.5.1.5, c'est-à-dire *Force directed, la Programmation en nombres entiers* ou *le Rafinement itératif*, est problablement utilisé pour cette étape. Encore une fois, il est difficile de déterminer avec exactitude l'algorithme utilisé, car ce n'est pas visible pour l'utilisateur.

Tableau 4.6 La surface des blocs matériels

# du bloc	Nom du bloc	Surface	Surface normalisée	# du bloc	Nom du bloc	Surface	Surface normalisée
1	Encodeur RS	2333,06	0,021	7	FFT (128-points)	8408,91	0,074
2	Entrelaceur	560,00	0,005	8	Table d'alloc. des bits	16970,00	0,150
3	Encodeur Treillis	2110,64	0,019	9	Décodeur QAM 128-points	2243,64	0,020
4	Encodeur QAM	1328,00	0,012	10	Décodeur Viterbi	3777,29	0,033
6	IFFT (16-points)	8414,64	0,074	12	Décodeur RS	40921,95	0,361

Pour aider l'outils lors de la synthèse, nous avons parfois dû empêcher l'outil de synthèse de dérouler certaines boucles. Monet déroule par défaut les boucles « if ». Cela avait pour conséquence de diminuer la surface matérielle des blocs, mais en augmentant leurs temps d'exécution. Le temps nécessaire à la synthèse était également beaucoup plus court. Afin de minimiser la surface utilisée pour l'entreposage des variables, des mémoires ont été utilisées. Les mémoires que nous avons utilisés se nomment « *HCMOS5_ramsp* » et se trouvent dans les bibliothèques *ram_singleport_lib* et *ram_mgc.lib* de Mentor Graphics. Lorsque nous avons un grand nombre de variables, il est plus avantageux d'utiliser des mémoires, car des registres seront créés pour chacune des variables. Les mémoires, même si elles sont également composées de registres, sont en général faites d'une manière plus optimale pour le temps d'exécution et la surface qu'un ensemble de registres. Mais cela est surtout avantageux pour le temps nécessaire à la synthèse du bloc.

Nous voulions également être indépendants de la technologie utilisée. Cela signifie que peu importe l'évolution des technologies matérielles, le meilleur partitionnement fait à partir de nos estimations resterait le même. Pour ce faire, nous avons normalisé les valeurs de la surface matérielle, en divisant la surface de chacun des blocs par la surface totale utilisée, avant de l'utiliser dans l'algorithme de partitionnement. Les résultats du tableau 4.6 ont été obtenus en utilisant les bibliothèques *mgc_lca300k_dmag_dc.lib* et *mgc_lca300k_comp_dc.lib* de Mentor Graphics. Le tableau 4.6 donne les résultats obtenus pour l'estimation de la surface matérielle des blocs et le tableau 4.7 donne les résultats obtenus pour l'estimation du temps d'exécution des blocs matériels.

Nous avons eu certains problèmes lors de la synthèse, qui peuvent avoir influencés les résultats. Monet avait parfois de la difficulté à assigner certaines opérations à des opérateurs. Par exemple, si l'on regarde le code VHDL de nos différents blocs, il est possible de voir que le code n'est pas toujours optimisé. Parfois, l'outil de synthèse n'acceptait pas des opérations tels : $a = b * c$, où a, b et c étaient des variables du type entier. Pour contourner ce problème, nous avons dû assigner par exemple la valeur de la variable c à la variable d (du même type), puis utiliser la variable d dans l'équation ($a = b * d$) afin de faire une synthèse. Ce problème s'est produit quelques fois et provenait de l'outil et non du code. Le code a donc dû être changé afin de réussir à synthétiser. Par contre, dans la plupart des cas, le code obtenu au niveau RTL était optimisé et le code redondant que nous avions ajouté était éliminé.

Par contre, cet outil avait pour avantage de nous permettre de faire la synthèse et donc l'estimation de gros circuits. Seul un des blocs, le décodeur Reed-Solomon, a dû être partitionné en vu d'en faire la synthèse, car le bloc entier prenait trop de temps à être synthétisé et l'outil ne le supportait pas. Le décodeur Reed-Solomon a été divisé

en trois parties, qui sont le calcul du syndrôme, l'algorithme de Berlehamp pour trouver le polynôme de location d'erreur et finalement la correction d'erreur (figure 4.19).

Tableau 4.7 Temps d'exécution des blocs matériels					
# du bloc	**Nom du bloc**	**Temps d'exécution (s)**	**# du bloc**	**Nom du bloc**	**Temps d'exécution (s)**
1	Encodeur RS	$1{,}69922 \times 10^{-6}$	7	FFT (128-points)	$4{,}26667 \times 10^{-6}$
2	Entrelaceur	$1{,}03984 \times 10^{-6}$	8	Table d'alloc. des bits	$6{,}82647 \times 10^{-6}$
3	Encodeur Treillis	$0{,}02656 \times 10^{-6}$	9	Décodeur QAM 128-points	$5{,}30320 \times 10^{-5}$
4	Encodeur QAM	$9{,}62500 \times 10^{-7}$	10	Décodeur Viterbi	$4{,}12862 \times 10^{-5}$
6	IFFT (16-points)	$3{,}42491 \times 10^{-6}$	12	Décodeur RS	$3{,}82245 \times 10^{-5}$

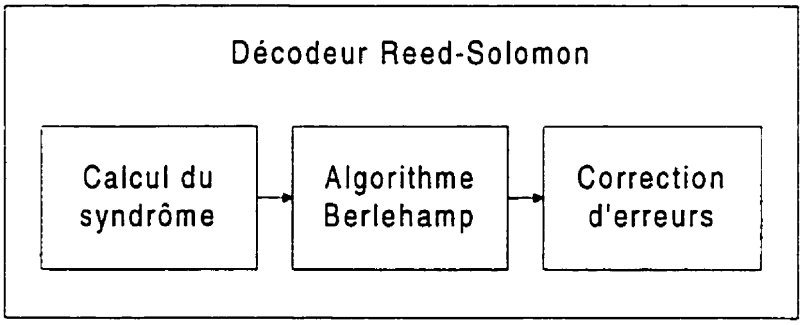


Figure 4.19 Schéma du partitionnement de l'encodeur Reed-Solomon

Il est possible de faire une comparaison entre les temps d'exécution des blocs en matériel et ceux obtenus pour le logiciel. C'est-à-dire que nous comparons les résultats du tableau 4.5 et ceux du tableau 4.7. Le tableau 4.8 contient l'accélération du temps d'exécution des blocs en matériel par rapport à une implantation en logiciel. Par exemple, l'encodeur QAM s'exécute 76 fois plus rapidement en matériel qu'en logiciel.

Tableau 4.8 Accélération des blocs placés en matériel					
# du bloc	**Nom du bloc**	**Accélération entre le logiciel et le matériel**	**# du bloc**	**Nom du bloc**	**Accélération entre le logiciel et le matériel**
1	Encodeur RS	21 113	7	FFT (128-points)	982
2	Entrelaceur	13 753	8	Table d'alloc. des bits	326
3	Encodeur Treillis	32 187	9	Décodeur QAM 128-points	224
4	Encodeur QAM	76	10	Décodeur Viterbi	26 764
6	IFFT (16-points)	379	12	Décodeur RS	42 486

Les blocs 4,5,8 et 9 c'est-à-dire les tables d'allocation des données et l'encodeur et décodeur QAM ont très peu d'accélération comparés aux autres blocs. Cela est dû au fait que les différentes parties de ces blocs ne peuvent être faites en parallèle. Lors de la synthèse, les boucles contenues dans ces blocs n'ont pas été déroulées. Le pragma "don't unroll" a été ajouté à ces boucles. L'accélération est moyenne pour la FFT et la IFFT et elle est grande pour les autres blocs. En examinant le code, nous pouvons donc dire que le code ayant principalement une exécution séquentielle en matériel et en logiciel n'aura pas une grande accélération. Certaines parties de la FFT et de la IFFT peuvent être mises en parallèles. Donc, nous obtenons une accélération moyenne si nous comparons les résultats de l'implantation en matériel, à celle en logiciel. Finalement, les autres blocs possèdent plusieurs boucles imbriquées et l'outil de synthèse a pu les dérouler. Le déroulement de ces boucles a permis d'exécuter plusieurs opérations en parallèle, entraînant par le fait même une plus grande accélération entre les deux implantations.

4.8 Estimation de la dissipation de puissance pour la partie matérielle

Pour l'estimation de la dissipation de puissance pour les blocs matériels, nous avons utilisé un autre outil. Il existe à l'intérieur de l'outil de synthèse de Synopsys

(Design Compiler) la possibilité d'obtenir la puissance utilisée. Comme nous avons au départ une description comportementale en VHDL, nous devons faire une synthèse à l'aide de l'outil de synthèse comportementale de Synopsys (Behavioral Compiler). Une fois la description RTL de nos blocs obtenue, il est possible de faire une nouvelle synthèse et d'obtenir un rapport sur la dissipation de puissance.

Tableau 4.9 Puissance dissipée pour les blocs matériels

# du bloc	Nom du bloc	Puissance dissipée (uW)	Puissance normalisée	# du bloc	Nom du bloc	Puissance dissipée (uW)	Puissance normalisée
1	Encodeur RS	383,8933	0,035	7	FFT (128-points)	1439,8607	0,129
2	Entrelaceur	71,5043	0,006	8	Table d'alloc. des bits	1567,3221	0,141
3	Encodeur Treillis	69,1252	0,006	9	Décodeur QAM 128-points	439,5822	0,040
4	Encodeur QAM	214,5129	0,020	10	Décodeur Viterbi	682,8342	0,061
6	IFFT (16-points)	1066,5635	0,096	12	Décodeur RS	4701,9663	0,422

La synthèse comportementale est faite en utilisant les bibliothèques« *class* ». Pour les opérations et les mémoires, nous avons utilisé des composants de la bibliothèque « Synopsys ». Nous avons utilisés les mémoires *DW03_ram1* pour les variables de la même façon que nous l'avions fait avec l'outil de synthèse Monet. La synthèse au niveau RTL a été faite avec les mêmes bibliothèques. Il est alors possible de faire un rapport sur la puissance. Ce rapport nous donne la puissance dynamique totale, qui est en fait la puissance dissipée interne des cellules et la puissance de la permutation du circuit. Les résultats obtenus sont disponibles dans le tableau 4.9. La dissipation de puissance a été calculée d'une façon statique, car les entrées du circuit ne sont pas utilisées pour déterminer la puissance utilisée.

Nous avons eu certains problèmes qui ont pu influencer les résultats obtenus. Tout d'abord Synopsys ne pouvait faire la synthèse de code VHDL de la taille de nos blocs. Donc, nous avons divisé nos blocs d'une manière plus fine afin de faire une synthèse. Les blocs ont été divisés selon leurs boucles et leurs branchements conditionnels. Nous avons essayé de placer les opérations qui étaient exécutées le même nombre de fois ensemble pour en faire la synthèse. En divisant ainsi les blocs, nous diminuons la possibilité pour certaines opérations de partager des ressources. Donc, nous risquons d'avoir une synthèse qui n'est pas de la même qualité que les précédentes faites avec Monet de Mentor Graphics. De plus, avant d'obtenir des résultats, nous devons faire une première synthèse pour passer d'une description comportementale à RTL, puis faire une synthèse au niveau RTL. Nous faisons donc une estimation sur un design qui pourrait être conçu d'une manière plus optimale, au niveau de la description VHDL et de la synthèse.

4.9 Estimation des communications

Pour faire une bonne estimation des communications, nous devons tenir compte du processeur utilisé. Comme notre système comporte un grand nombre de communications, c'est un élément qui ne peut être négligé. Les données transférées sont très nombreuses et leur temps de transfert est beaucoup plus élevé que le temps nécessaire pour transférer les signaux de contrôle. Nous avons donc décidé de faire une estimation de haut niveau en tenant compte seulement du temps de communications des données. Notre estimation utilise l'équation suivante :

$$cycles\ de\ transfert = \frac{quantité\ de\ données}{largeur\ du\ bus} * nombre\ de\ cycles\ pour\ un\ transfert \quad (4\text{-}4)$$

Comme il est possible de le voir dans les figures 4.20 et 4.21, avec un ARM7 deux cycles sont nécessaires pour faire une écriture ou une lecture. Donc, le processeur ARM7 ayant une vitesse de 2.5 MHz, chaque période de cycle est de 400 ns. Pour donner un exemple de ces estimations, si nous avons 256 mots de 32 bits à transférer, le temps de communications sera de 256*400 ns = 102 400 ns, car le bus de données possède 32 bits de large. Il est évident qu'avant de faire ces transferts, un contrôle ou une synchronisation est nécessaire, ajoutant quelques cycles à notre taux de transfert. Toutefois, il n'est pas pris en considération, le temps de transfert étant beaucoup plus élevé.

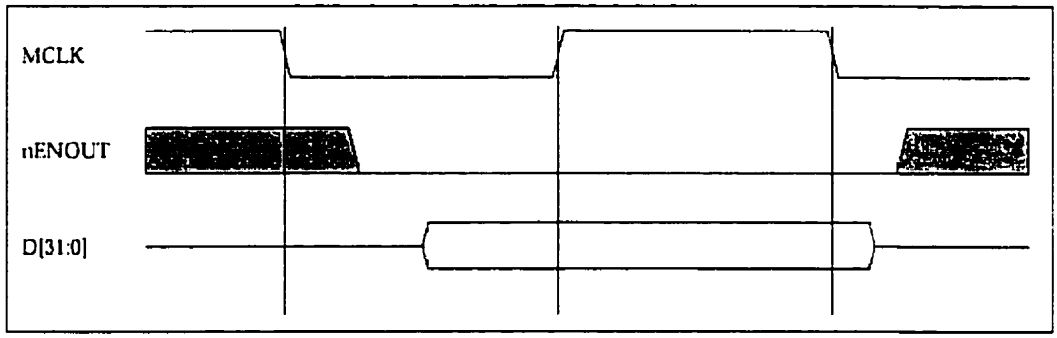


Figure 4.20 Cycle pour écriture sur le bus bidirectionnel

Si nous regardons la formule utilisée pour estimer les communications dans la section 2.6.3, nous pouvons voir que le temps de calcul des pilotes ou des interfaces n'est pas utilisé dans l'équation 4-4. En fait, le temps d'exécution du pilote logiciel et de l'interface matérielle sont inclus dans le temps d'exécution des différents blocs. Les résultats des estimations pour les communications à partir des blocs matériels sont disponibles dans le tableau 4.10, alors que les estimations pour les communications à partir des blocs logiciels sont disponibles dans le tableau 4.11. Ces temps de communication sont pour une seule exécution de chacun des blocs.

Les communications sont seulement données pour le transfert des données à la fin de l'exécution du bloc. Car il est inutile de calculer le temps de communication nécessaire pour transférer les données au bloc suivant et le temps de communication d'un bloc lorsqu'il reçoit les données du bloc précédent. Cela devrait donner le même temps, car ce sont les mêmes données transférées entre les mêmes composants. Seulement le dernier bloc de l'encodeur et du décodeur ont besoins d'un temps de communication après la fin de leur exécution. Nous avons appelés ces blocs 6b et 12b dans les tableaux 4.10 et 4.11. Ces valeurs n'étant pas redondantes, elles doivent être ajoutées au temps d'exécution du système.

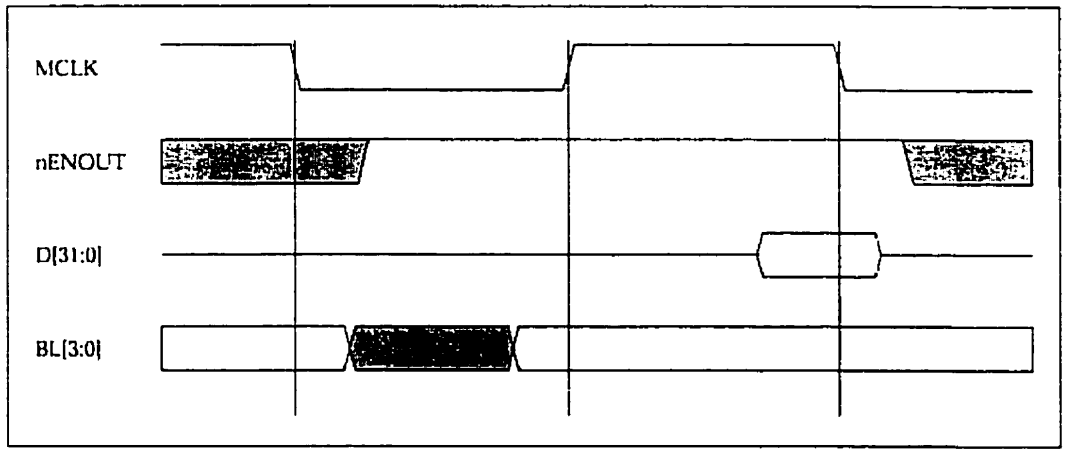


Figure 4.21 Cycle pour lecture sur le bus bidirectionnel

Dans les deux tableaux, les blocs 5 et 8 n'ont pas de valeurs pour les communications. Cela est dû au fait que ces blocs ne reçoivent aucune valeur en entrée, donc aucune valeur pour les communications n'est nécessaire. Ils reçoivent donc la mention "ne s'applique pas". Dans le tableau 4.11, deux autres blocs ont la même mention, c'est-à-dire les blocs 1 et 7 pour les communications provenant du logiciel. Le bloc 1 est le premier bloc de la partie encodeur du modem et le bloc 7 et le premier bloc de la partie décodeur du modem. Si l'on regarde à la figure 2.8 qui explique le modèle

Tableau 4.10 Temps de communication avec un bloc matériel			
# du bloc	**Nom du bloc**	**Temps de com. provenant d'un bloc logiciel (ns)**	**Temps de com. provenant d'un bloc matériel (ns)**
1	Encodeur RS	19 200	2400
2	Entrelaceur	20 800	2600
3	Encodeur Treillis	20 800	2600
4	Encodeur QAM	4400	550
5	Table d'alloc. des bits	Ne s'applique pas	Ne s'applique pas
6	IFFT (16-points)	3200	400
6b	Communication à la fin de l'exécution	3200	400
7	FFT (128-points)	25 600	3200
8	Table d'alloc. des bits	Ne s'applique pas	Ne s'applique pas
9	Décodeur QAM 128-points	25 600	3200
10	Décodeur Viterbi	41 600	5200
12	Décodeur RS	20 800	2600
12b	Communication à la fin de l'exécution	19 200	2400

Tableau 4.11 Temps de communication avec un bloc logiciel			
# du bloc	**Nom du bloc**	**Temps de com. provenant d'un bloc logiciel (ns)**	**Temps de com. provenant d'un bloc matériel (ns)**
1	Encodeur RS	Ne s'applique pas	2400
2	Entrelaceur	0	2600
3	Encodeur Treillis	0	2600
4	Encodeur QAM	0	550
5	Table d'alloc. des bits	Ne s'applique pas	Ne s'applique pas
6	IFFT (16-points)	0	400
6b	Communication à la fin de l'exécution	0	400
7	FFT (128-points)	Ne s'applique pas	3200
8	Table d'alloc. des bits	Ne s'applique pas	Ne s'applique pas
9	Décodeur QAM 128-points	0	3200
10	Décodeur Viterbi	0	5200
12	Décodeur RS	0	2600
12b	Communication à la fin de l'exécution	0	2400

de communication que nous utilisons, nous pouvons voir qu'un bloc est dédié aux entrées et sorties. Ces entrées et sorties proviennent de l'extérieur du système et ne peuvent donc pas être obtenues à partir du logiciel.

4.10 Fonction objectif

Les valeurs qui sont estimées en utilisant les techniques précédentes sont utilisées à l'intérieur d'une fonction objectif pour déterminer la qualité d'un partitionnement. Pour les parties logicielles, les estimations logicielles sont utilisées et pour les parties matérielles, les estimations matérielles sont utilisées. Par exemple, si lors d'un partitionnement, un bloc est placé en matériel, le temps d'exécution du bloc matériel sera utilisé pour calculer le temps d'exécution du bloc.

Nous avons élaboré une fonction objectif sur deux niveaux. Le premier niveau vérifie si le modèle (le partitionnement) élaboré respecte les contraintes de temps. En fait, le temps d'exécution des blocs et les temps de communications sont utilisés pour déterminer le temps d'exécution total du système. Si le temps d'exécution total est supérieur aux contraintes de temps, le modèle est automatiquement rejeté et on passe au deuxième niveau de la fonction objectif.

Le deuxième niveau utilise la dissipation de puissance et la surface utilisée pour le matériel. En fait, une fois la certitude obtenue que le système rencontre les contraintes de temps, nous essayons d'obtenir le système qui coûtera le moins cher, donc de minimiser la surface utilisée, tout en conservant la dissipation de puissance la plus faible possible. Les valeurs sont normalisées en divisant leur valeur par la valeur obtenue par les blocs en entier. Par exemple, si on utilise la surface, on divise la surface d'un bloc, par la surface si tous les blocs étaient placés en matériel. Les valeurs

normalisées de la dissipation de puissance (P) et de la surface (S) sont utilisées dans l'équation suivante pour obtenir un nombre représentant la qualité (Q) du système :

$$Q = S + P \tag{4-5}$$

Donc, en regardant l'équation nous pouvons voir que l'on estime la qualité du système en regardant la dissipation de la puissance et la surface du circuit intégré avec un poids de 50% pour chacun des critères, car il n'y a aucun poids placé devant les termes de l'équation. Les modèles ayant une qualité (Q) inférieure aux autres est le modèle possédant le meilleur partitionnement matériel/logiciel.

4.11 Algorithme de partitionnement

Pour l'algorithme de partitionnement, nous nous sommes basés sur un algorithme précis. C'est-à-dire que nous avons regardé tous les modèles possibles, en faisant une énumération exhaustive. Nous avons en tout 12 blocs, ce qui fait 2^{12} ou 4096 modèles. Ce nombre de modèles n'est pas très grand et il permet de vérifier chacune des possibilités de partitionnement matériel/logiciel. Si nous avions utilisé un nombre plus grand de blocs, une recherche exhaustive aurait pu être impossible et il aurait fallu utiliser un heuristique qui n'aurait peut-être pas trouvé le modèle partitionné d'une manière optimale.

Nous avons tout d'abord créé un tableau de 12 par 4096 pour contenir chacun des blocs de tous les modèles. Puis, des boucles sont utilisées pour déterminer l'endroit où sera situé chacun des blocs, en matériel ou en logiciel selon l'algorithme. Si le bloc est placé en matériel, il aura la valeur « 0 » et s'il est placé en logiciel, il aura la valeur

« 1 ». La fonction objectif de la section 4.7 est ensuite utilisée pour déterminer la qualité des 4096 modèles et seul les 3 meilleurs modèles sont conservés.

4.12 Le meilleur partitionnement pour le Universal ADSL

Pour le Universal ADSL, seulement deux blocs peuvent respecter les contraintes de temps s'ils sont implantés en logiciel sur le ARM7TDMI. Donc, nous n'avons le choix qu'entre trois partitionnements possibles. Le meilleur partitionnement étant celui plaçant *l'encodeur QAM* et la *table de chargement des bits pour l'encodeur* en logiciel. Les blocs de contrôle qui ont été présentés à la figure 2.8, c'est-à-dire notre modèle de communications sont également placés en logiciel. Ces blocs sont *le contrôleur des entrées et sorties*, le *contrôleur du système* et *l'arbitre du bus*. Tous les autres blocs sont placés en matériel.

4.13 Implantation de mécanismes de communications

Nous avons décidé d'utiliser comme mécanisme de communications, les passages de messages. Les données sont transférées au bloc suivant en même temps pour chacun des blocs, de la même façon que dans un pipeline.

Le processeur est le contrôleur du système. Il doit pouvoir communiquer avec tous les co-processeurs matériels. Pour ce faire, il utilise également une communication point à point bloquante.

4.14 Implantation de canaux de communications

Les mécanismes de communications expliqués dans la section 2.6.1 peuvent être intégrés tels quels, en implantant en matériel les signaux de communications nécessaires entre deux blocs placés en matériel. Chacun des blocs ne transmet des données qu'au bloc précédent et au bloc suivant. Deux signaux implantés sur des lignes de communications simples, sont nécessaires pour établir une synchronisation entre les deux blocs et un bus de 32 bits est utilisé pour transférer les données. Comme nous n'utilisons qu'un seul processeur, les blocs exécutés en logiciel n'ont pas besoin de mécanisme de communications entre eux. Les communications se font alors implicitement en utilisant les valeurs en mémoire ou dans les registres. Les communications entre des blocs matériel et logiciel sont un peu plus complexes. Les figures 4.22 à 4.25 montrent certains des mécanismes développés pour une communication entre un bloc matériel et un processeur ARM7TDMI.

Le premier mécanisme, montré à la figure 4.22 utilise une communication avec une architecture mémoire Harvard. C'est-à-dire que notre mémoire possède deux bus de donnée et deux bus d'adresses et que chaque ensemble de bus appartient soit au processeur, soit au bloc matériel. Un comparateur est utilisé pour détecter l'adresse indiquant le départ du bloc matériel et une interruption est utilisée pour en montrer la fin. Les données sont échangées par mémoire partagée et la communication est non-bloquante.

Le mécanisme de la figure 4.23 utilise des entrées et sorties du ARM7 qui sont spécialement faites pour être utilisées avec un co-processeur ou ce que l'on appelle un bloc matériel. Le ARM7 active la sortie nCI qui indique au co-processeur qu'il doit exécuter des instructions. Le bloc matériel active les signaux CPA pour indiquer s'il est présent et CPB pour indiquer s'il est occupé ou non. Lorsqu'il est prêt à démarrer, il

désactive l'accès aux bus du processeur ARM7 en activant les entrées du processeur ABE et DBE. De cette façon, les bus ne peuvent être accédés par le processeur et le bloc matériel en même temps, ainsi que la mémoire.

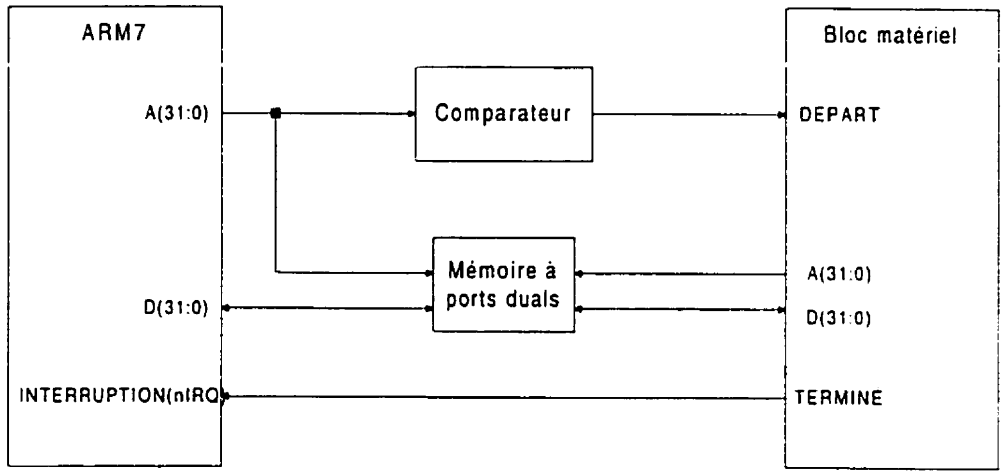


Figure 4.22 Communication avec une architecture mémoire Harvard

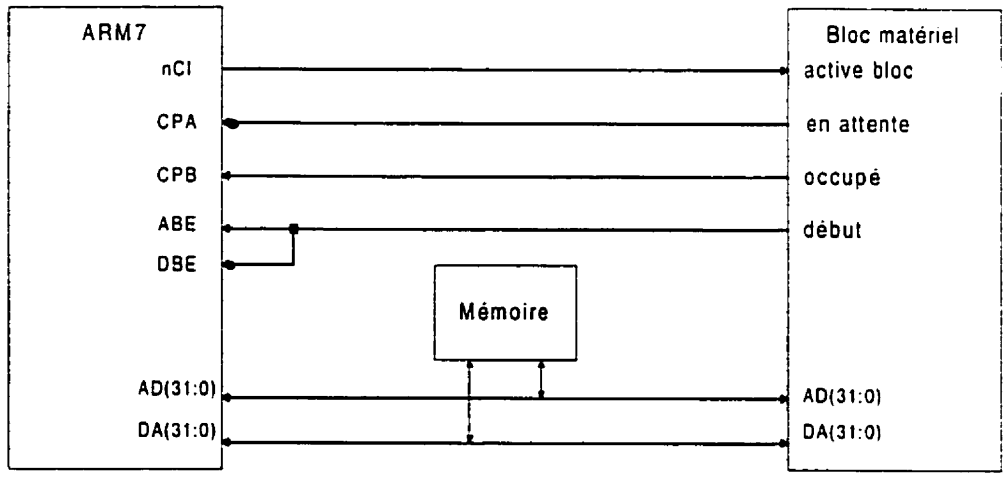


Figure 4.23 Communication avec broches pour co-processeur

La figure 4.24 utilise une communication directe sur le bus de données. Pour indiquer au co-processeur qu'il lui envoie des données ou qu'il veut en recevoir, le ARM7 doit écrire à une adresse spécifique qui est décodée et transformée en un signal

simple pour le bloc matériel. C'est une communication point-à-point qui est bloquante. Avec ce mécanisme, on doit faire attention aux adresses utilisées pour activer les blocs matériels. Si cette adresse est également une adresse mémoire, des conflits peuvent se produire sur le bus.

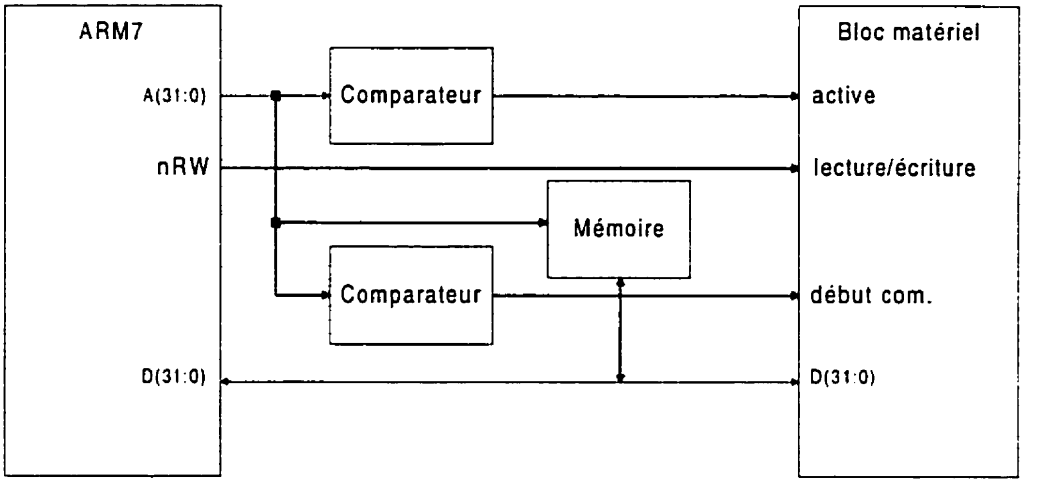


Figure 4.24 Communication directe sur le bus de données

Finalement, le dernier modèle que nous avons examiné est un modèle utilisant les bus spécifiques du ARM7. Le ARM7 possède plusieurs bus de données, dont un bus bi-directionnel, un bus uniquement en entrée et un bus uniquement en sortie. Pour lire le programme en mémoire ou pour écrire ou lire des données en mémoire, un bus bi-directionnel doit être utilisé. Par contre, si certains blocs ne font que recevoir ou envoyer des données au processeur, un bus unidirectionnel peut être utilisé. Dans l'exemple de la figure 4.25, le bloc matériel 1 peut seulement envoyer des données au ARM7, mais il ne peut en recevoir. D'un autre côté, le bloc matériel 2 peut seulement recevoir des données du ARM7, mais il ne peut en envoyer. Des paramètres doivent être fournis au processeur pour lui indiquer sur quel bus envoyer ou recevoir des données. Pour synchroniser le transfert des données, le processeur lit à une adresse particulière, qui active les blocs matériels.

Après une analyse des différents modèles de communications, nous avons décidé d'utiliser le modèle de *Communication directe sur le bus de données*, présenté à la figure 4.24. La raison qui a motivé ce choix, est que nous voulions avoir une communication la plus rapide possible. Les données devaient être envoyées et reçues immédiatement. Pour cette raison, nous avons mis de côté les modèles utilisant une communication par mémoire partagée. Les communications que nous voulons utiliser pour le modem sont bloquantes, donc le fait de lire et d'écrire en mémoire pour chaque mot de données, nous prenait trop de temps. Pour cette raison, les modèles *communications avec une architecture mémoire Harvard et communications avec broches pour co-processeur* n'ont pas été retenus.

De plus, le modèle de communications *Communications utilisant les bus unidirectionels* est très intéressant, car il permet d'utiliser les bus d'entrées et de sorties de manière optimale. Par contre, si nous voulons changer de sorte de processeur, notre modèle de communications ne tient plus. De plus, il est utilisable dans notre cas, car un seul bloc matériel envoie des données au processeur et un seul bloc matériel reçoit des données du processeur. Mais, si un autre bloc non-adjacent aux blocs déjà en logiciel avait été placé en logiciel, ce modèle aurait nécessité de petites modification pour être fonctionnel. Comme les performances en temps sont similaires des modèles *communication directe sur le bus de données* et *communications utilisant les bus unidirectionels*, nous avons donc décidé d'utiliser le modèle de *communication directe sur le bus de donnée* pour implanter notre modem.

En utilisant le modèle de *communication directe sur le bus de donnée*, combiné au modèle de communications de la figure 2.8 pour implanter notre modem, nous obtenons le système de la figure 4.26.

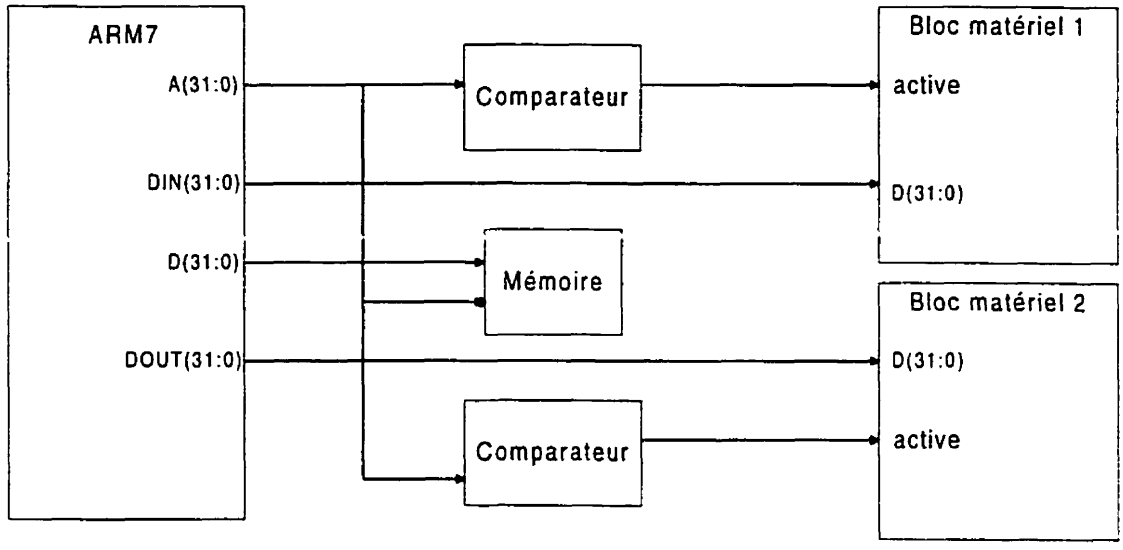


Figure 4.25 Communications utilisant les bus unidirectionels

Pour faciliter la compréhension de notre système, nous allons expliquer deux moyens de communications. Le premier est une communication entre deux blocs matériels et le second est une communication d'un bloc logiciel vers un bloc matériel.

Dix de nos douze blocs sont placés en matériel. Cela entraîne donc beaucoup de communications par paire de bloc matériel entre deux blocs matériels. À titre d'exmple, nous allons expliquer comment se fait la communication entre le déentrelaceur et le décodeur Reed-Solomon. Les autres blocs communiquent de façon similaire. Tout d'abord, lorsque le déentrelaceur a terminé le traitement de ses données et qu'il est prêt à transférer les données au décodeur Reed-Solomon, il active le signal *requête* pour indiquer qu'il est prêt à effectuer le transfert.

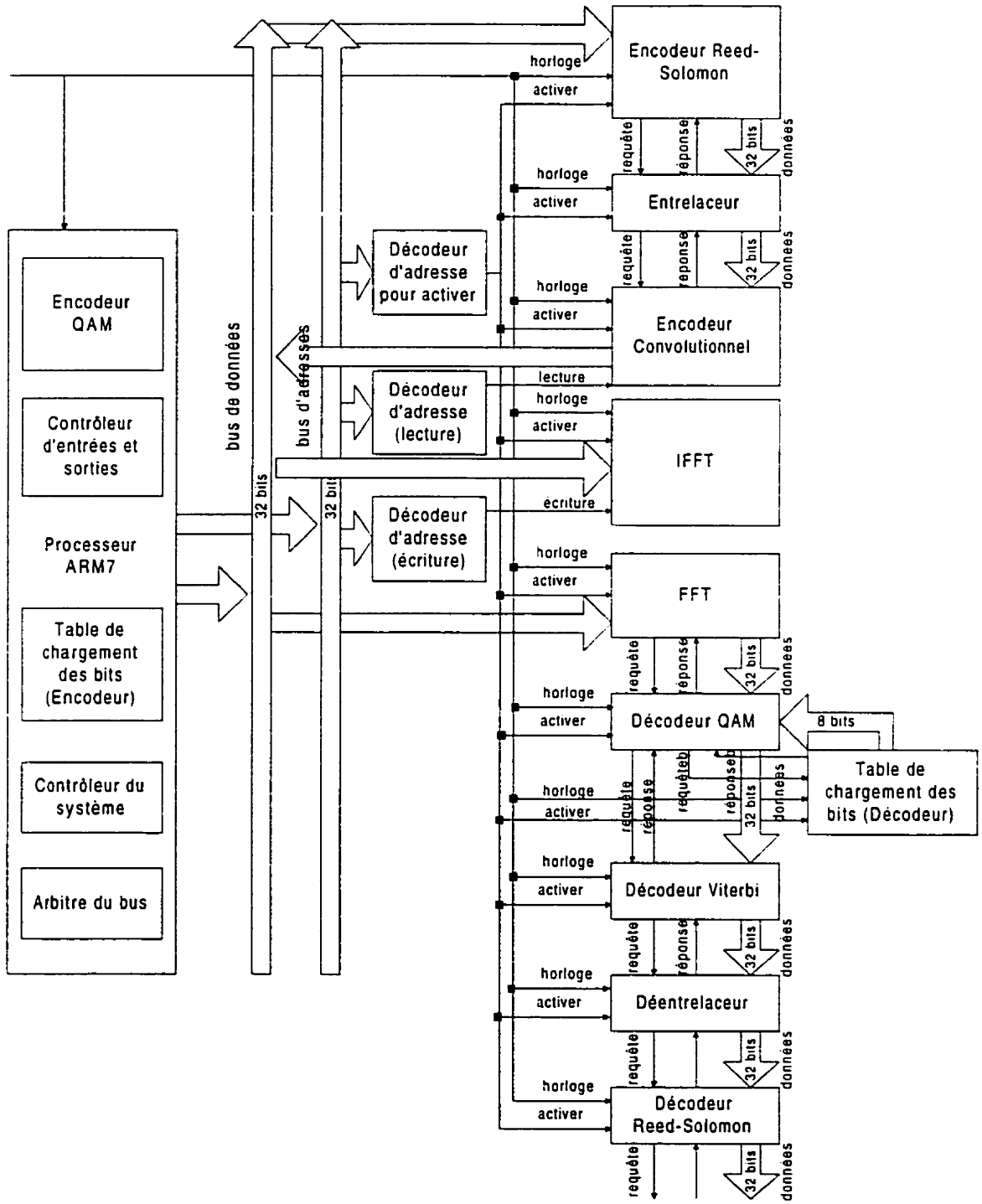


Figure 4.26 Implantation du modem

Ce signal reste activé tant qu'il n'a pas reçu la réponse du décodeur Reed-Solomon à l'aide du signal *réponse*. Le décodeur Reed-Solomon active ce signal lorsqu'il est prêt à recevoir les données. Au cycle suivant, les premières données sont transférées du déentrelaceur jusqu'au décodeur Reed-Solomon par paquets de 32 bits comme il est indiqué à la figure 4.27. Les deux blocs utilisent une horloge commune et la communication est bloquante, car les deux blocs doivent se synchroniser avant de commencer le transfert.

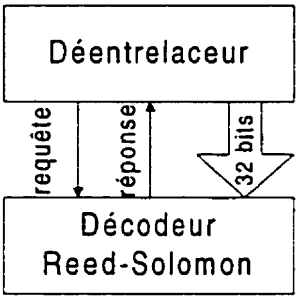


Figure 4.27 Communications entre deux blocs matériels

Pour une communication entre un bloc logiciel et un bloc matériel, nous avons pris l'encodeur QAM et la IFFT (figure 4.28). Le modèle du modem est modélisé par un pipeline. La partie la plus lente étant celle exécutée sur le processeur. Donc, la IFFT n'a pas besoin d'indiquer au processeur qu'il est prêt à recevoir des données. Il est évident qu'il aura toujours terminé de traiter ses données lorque le processeur voudra lui en transférer de nouvelles. Lorsque le processeur veut envoyer des données à la partie matérielle, celui-ci écrit la donner à transférer à une adresse particulière. Cette adresse est lue et décodée par le décodeur d'adresse. Lorsque l'adresse reçue est celle de la IFFT, alors ce bloc se met en mode lecture et prendra la valeur contenue sur le bus de données. À chaque activation du bloc une seule lecture sur 32 bits sera effectuée. Dans ce cas-ci, les communications ne sont pas bloquantes, car il est assumé que la IFFT est en tout temps dans l'attente des données.

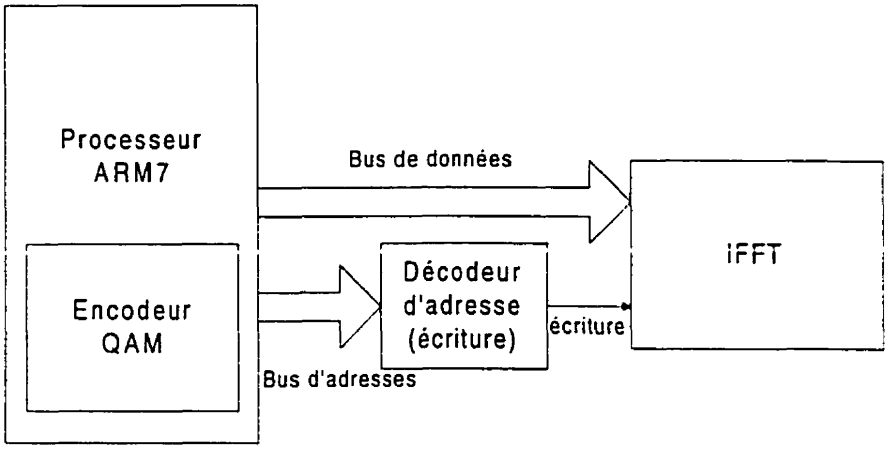


Figure 4.28 Communications entre un bloc logiciel et un bloc matériel

4.15 Co-simulation matérielle/logicielle

La co-simulation se fait à l'aide de Seamless CVE. Cet outil a été décrit au début de ce chapitre, dans la section 4.1.3. Un simlateur de jeu d'instructions est relié à un simulateur logique pour permettre de faire une cosimulation matérielle/logicielle.

Les blocs logiciels sont placés de façon séquentielle dans un programme exécuté sur le modèle VHDL du processeur ARM7 et simulé avec le simulateur logiciel. Toute la fonctionnalité logicielle a été placée dans même programme. Chacun des blocs matériels est décrit en VHDL et est utilisé comme composant dans le fichier VHDL comprenant le système en entier. Ce fichier contient les fils et les bus reliant les divers composants.

Une fois notre système monté, nous l'avons testé en utilisant certaines optimisations. Nous avons tout d'abord vérifié notre système sans aucune optimisation du modèle mémoire. Dans ces conditions, la simulation s'exécute très lentement, mais chaque chargement d'instruction et chaque accès mémoire peuvent être visualisés dans

le simulateur logique. Puis, une fois cette vérification faite, nous avons exécuté la cosimulation en utilisant une optimisation appelée « Fetch » [Seam99]. Cette optimisation permet d'accéder aux instructions à partir du serveur de mémoire. Donc, les instructions ne sont pas lues et envoyées sur le bus de données, mais accédées directement à partir du logiciel. Une deuxième optimisation « Data » permet de faire également la même chose, mais pour les données. Ces deux optimisations réduisent de beaucoup le temps nécessaire pour la cosimulation. Il est alors plus rapide de vérifier le comportement du système et des variables. Nous avons remarqué que ces optimisations étaient pratiques pour tester nos parties matérielles et ses communications avec le processeur. Par contre, ces optimisations peuvent rendre pénible la validation de la partie logicielle, qui doit préférablement être faite sans elles. Une troisième optimisation aurait également pu être utilisée. Cette optimisation appelée « Warp » permet au simulateur du jeu d'instructions de se synchroniser au matériel seulement dans le cas d'un accès à une région non-optimisée de la mémoire, ou lors d'une interruption. Dans ce cas, le simulateur matériel ne respecte pas son temps d'exécution, car il n'avance que lorsque le logiciel l'accède. Nous trouvions cette optimisation inutile dans notre cas, car nous voulions avoir des temps d'exécutions réalistes autant du côté logiciel, que du côté matériel.

Chapitre 5
Conclusion

Vu la complexité toujours croissante des systèmes embarqués, il est de plus en plus difficile d'effectuer manuellement un partitionnement matériel/logiciel. Ceci est la raison pour laquelle nous avons développé une méthodologie de codesign matériel/logiciel. Nous avons développé une méthodologie pour une application de communications à haute vitesse. Pour tester et valider notre méthodologie, trois applications, c'est-à-dire trois modems de la famille xDSL ont été développées. Ces trois modems sont le ADSL (Asymmetric Digital Subscriber Line), le Universal ADSL qui est une version légère du ADSL et finalement le VDSL (Very High Bit-Rate Digital Suscriber Line) qui est une version plus performante du ADSL. Ces trois modems utilisent la même technologie, qui est DMT (Discrete Multitone), mais de manière différente.

Chaque type de système a ses contraintes particulières. Un modem est un système en temps réel. Il a donc des contraintes de temps très strictes à respecter. Par contre, une fois les temps d'exécution atteint, il est n'est pas avantageux de pousser plus loin l'optimisation des performances. Un autre critère a examiner est la dissipation de puissance. Certains des modems développés peuvent être utilisés dans des centrales. Ces endroits contiennent une grande quantité de modems et sont très limités en espace. Il est donc important de minimiser la dissipation de puissance, car il peut être difficile à l'intérieur d'une centrale d'espacer les modems ou d'y placer des ventilateurs. Le troisième critère que nous avons examiner est la surface du matériel. Plus la surface matérielle est grande, plus le coût en sera élevé. Donc, pour minimiser les coûts de fabrication, nous avons placé la plus grande quantité de comportement possible d'une application en logiciel. Car, la partie logicielle reste de la même grosseur quel que soit le nombre de blocs implantés en logiciel. Les trois critères que nous venons d'énumérer

sont intégrés à notre méthodologie de codesign matériel/logiciel afin d'obtenir un modem conçu d'une façon optimale.

Une fois l'algorithme de notre application connu, nous avons partitionné le modem en différents blocs. Nous avons utilisé un gros grain, c'est-à-dire que notre application a été divisée en un petit nombre de blocs, mais contenant chacun une grande quantité de fonctionnalité. Chacun des blocs peut être placé en matériel ou en logiciel. Afin de pouvoir évaluer les deux possibilités, chacun des blocs a été décrit en langage C pour en faire une évaluation logicielle ou en VHDL pour en faire une évaluation matérielle. Il est bon de préciser que ces descriptions ont été faites au niveau comportemental.

La plupart des méthodologies de codesign que nous avons examinés utilisaient des outils pour les estimations conçus complètement par le concepteur de l'environnement de codesign. Notre méthodologie essaie d'utiliser lorsque c'est possible des outils commerciaux existants et utilisés dans les entreprises. Au lieu de créer de nouveaux outils, nous avons préféré utiliser des outils déjà testés et qui utilisent des algorithmes plus élaborés que ce que nous aurions pu développer à l'intérieur d'un projet de recherche de maîtrise. Il s'agissait donc pour nous d'intégrer des outils existants lorsque c'était possible. En fait, seul le calcul de dissipation de puissance pour le logiciel a nécessité l'étude d'une nouvelle méthodogie pour obtenir des estimations.

Deux critères ont été évalués pour les blocs logiciels, c'est-à-dire le temps d'exécution et la dissipation de puissance. Pour le temps d'exécution un profilage a été fait sur différents sous-blocs délimités par les boucles et les branchements conditionnels en utilisant différentes données. Les blocs auraient pu être exécutés directement sur le modèle matériel du processeur. Nous avons à la place utilisé un outil de co-simulation appelé Seamless CVE de Mentor Graphics, qui nous a permis de faire exécuter nos

programmes sur un modèle VHDL du processeur, qui respecte l'ensemble d'instructions, ainsi que les différents délais. Le temps d'exécution le plus grand a été utilisé comme critère. Pour la dissipation de puissance, un modèle d'évaluation basé sur les coûts de base des différentes instructions a été développé. Nous devions au départ utiliser le processeur TMS320C54 de Texas Instrument, mais comme son modèle VHDL n'a pas été disponible pour faire une cosimulation des modems, nous avons dû changer de processeur pour un ARM7. La méthodologie n'a pas été reprise avec le nouveau processeur.

Pour les estimations des blocs matériels, nous avons utilisé une méthodologie de synthèse rapide. Nous avons utilisé Monet de Mentor Graphics qui est un outil de synthèse au niveau comportemental pour estimer les temps d'exécution et la surface matérielle. Pour estimer la dissipation de puissance, nous avons utilisé un autre outil de synthèse, le Design Compiler de Synopsys, qui nous permettait d'obtenir la puissance au niveau RTL (Register Transfert Level). Donc, pour obtenir la dissipation de puissance, nous avons fait une première synthèse à partir de notre description comportementale, puis une deuxième synthèse au niveau RTL afin d'obtenir la dissipation de puissance.

Un quatrième critère a été évalué, le temps de communication entre les différents blocs. Ce critère est utilisé avec le temps d'exécution des blocs pour déterminer le temps d'exécution total du système. Dans une application de communications comme un modem, le transfert des données est un élément important et ne peut être négligé. L'évaluation a été faite en utilisant la largeur des bus, le nombre de données à transférer, ainsi que le temps de transfert pour un groupe de données.

Très peu d'outils de conception de codesign réalisés dans le passé ont utilisé la dissipation comme critère d'évaluation d'un système. Le temps d'exécution et la

surface du matériel sont des métriques courantes, ce qui n'est pas le cas présentement pour la dissipation de puissance. La dissipation de puissance était un élément important dans la conception de nos modems, nous avons donc dû l'intégrer à notre méthodologie. C'est un élément qui était très peu considéré dans le passé, mais qui prend de nos jours de plus en plus d'importance, car les systèmes portables sont de plus en plus populaires.

Une fois les différents critères évalués pour notre modem, nous devons utiliser un algorithme de partitionnement pour évaluer différents modèles. À chaque itération de l'algorithme, un nouveau modèle partitionné entre le matériel et le logiciel est créé. Les outils commerciaux que nous avons utilisé font des estimations beaucoup plus précises que ce que nous aurions pu obtenir en développant nos propres estimateurs. Par contre, le temps nécessaire pour ces estimations est en général beaucoup plus élevé. C'est la raison pour laquelle nous avons dû prendre une approche différente de ce qui avait été fait auparavant. Les outils de conception de codesign matériel/logiciel que nous avions étudiés utilisaient un algorithme de partitionnement pour déterminer un nouveau modèle qui était ensuite évalué à l'aide d'estimateurs. Donc, les estimations étaient faites sur chacun des modèles. Cela aurait été impossible dans notre cas, car calculer la valeur de toutes les métriques pour chacun des modèles aurait été une tâche fastidieuse et aurait pris énormément de temps. Nous avons donc calculé la valeur des différents blocs au départ pour ensuite utiliser ces valeurs dans l'algorithme. Même si nos estimations étaient assez longues à obtenir, nous ne les avons calculées qu'une seule fois. Le temps de calcul de notre algorithme de partitionnement étant ainsi énormément diminué, il nous est possible de tester un plus grand nombre de partitionnement et de trouver le modèle optimal d'une façon plus précise.

Notre modèle ne comptant que douze blocs, nous avons utilisé un algorithme qui trouve la solution d'une manière précise, une énumération exhaustive À chaque itération de l'algorithme, les valeurs pertinentes obtenues lors des évaluations sont

utilisées afin de déterminer le modèle de partitionnement matériel/logiciel étant le plus performant. Pour fusionner ces différentes valeurs en une seule représentant la qualité du modèle, nous utilisons une fonction objectif.

La fonction objectif utilisée possède deux niveaux. Le premier détermine si le modèle rencontre les contraintes de temps en utilisant le temps d'exécution des blocs et les temps de communications. Puis, la qualité du modèle est évaluée en utilisant à 50% chacun la surface matérielle et la dissipation de puissance du système.

Le principal désavantage d'utiliser notre méthodologie, est au niveau du choix des blocs. La taille de nos blocs, ainsi que la fonctionnalité de chacun des blocs sont déterminées au début de l'évaluation et ne peuvent être modifiés par la suite. À chaque fois qu'un bloc est modifié, le processus d'évaluation de toutes les métriques doit être repris à partir du début pour le bloc. Cela enlève donc une certaine flexibilité et peut nous empêcher de trouver un partitionnement qui pourrait être encore plus près d'un système conçu de façon optimale par rapport à celui qui est possible de trouver avec notre méthodologie.

Un autre élément qui nous enlève une certaine flexibilité est l'utilisation d'un seul type de processeur. Les évaluations logicielles doivent être refaites au complet si nous voulons changer de processeur ou si nous voulons utiliser plus d'un processeur à l'intérieur de notre algorithme de partitionnement.

L'application que nous avons utilisée nous permet de valider notre méthodologie, mais comme seulement deux blocs du modem Universal ADSL peuvent être placés en logiciel, il nous est difficile de vraiment explorer toutes les possibilités que pourraient nous offrir notre algorithme de partitionnement. Le processeur que nous devions utiliser au départ et que nous avons dû changer à la dernière minute du projet

aurait pu nous permettre une plus grande flexibilité dans les partitionnements. Par exemple, selon des résultats trouvé sur le site internet de Texas Instrument, il semble être possible de placer la FFT, la IFFT et le décodeur Viterbi sur le TMS320C54 et de respecter les contraintes de temps. Évidemment, les trois blocs n'auraient pu être placés en même temps dans la partie logicielle, mais cela nous aurait permis d'avoir plusieurs partitionnement possibles. Ces performances supérieures à celles du ARM7 sont dus au fait que certaines instructions du C54 permettent d'optimiser ces algorithmes.

De plus, nous avons peut-être été un peu trop ambitieux dans le choix de l'application. Une application moins complexe et moins exigeante en temps de traitement aurait pu nous permettre de mieux démontrer les avantages d'utiliser notre méthodologie. Une autre façon de remédier à ce problème aurait été d'utiliser un partitionnement ayant une granularité plus fine. C'est-à-dire avoir plus de blocs, mais ayant moins de fonctionnalité. Par contre, nous aurions encore eu un problème similaire, c'est-à-dire que même si le nombre de blocs placés en logiciel avait pu être plus élevé, il n'y aurait pas eu plus de fonctionnalité d'implanté en logiciel, mais nous aurions eu un plus grand nombre de partitionnements différents possibles.

Nous croyons qu'avec les critères que nous avons évalués qui sont, le temps d'exécution, le temps de communications, la surface matérielle et la dissipation de puissance, il nous est possible d'évaluer la plupart des types d'applications. Nous croyons également que notre méthodologie pourrait facilement être adaptée à d'autres systèmes en éliminant ou en ajoutant des métriques d'évaluation.

Il reste encore beaucoup de choses à faire dans le codesign matériel/logiciel. Comme mentionné précédemment, il aurait été possible d'utiliser une granualirté différente. Une autre amélioration pourrait être faite au niveau des communications. Dans ce projet, nous n'avons utilisé qu'un type de mécanisme de communication. Il

aurait pu être intéressant d'explorer différents mécanismes et de faire des estimations différentes pour chacun d'eux.

Dans notre application telle que nous l'avons définie, le nombre de blocs était peu élevé. Notre algorithme de partitionnement nous donnait des résultats satisfaisant, mais dans une autre situation où le nombre de bloc est important, il serait intéressant d'examiner plusieurs algorithmes de partitionnement différents. Il serait entre autre possible d'évaluer les performances d'un algorithme comme l'algorithme génétique, comme il a été fait dans [NiMa98]. De plus, avec des applications différentes, il serait possible d'examiner les résultats obtenus avec la fonction objectif, d'en changer les paramètres et d'examiner les variations sur les modèles de partitionnement obtenus.

Annexe A

Champs Galois et encodeur et décodeur Reed-Solomon

Les informations sur les codes Reed-Solomon proviennent de [Barr93][ChSu99][HWSW99][Plan99][Reed99][Rese99].

Les champs Galois

Les champs avec 2^m symboles sont appelés des champs Galois et sont dénotés $GF(2^m)$. Ces concepts mathématiques sont importants dans l'étude de code cyclique, comme les codes Reed-Solomon.

Une arithmétique avec 2^m symboles se dérive de la façon suivante. Premièrement, nous commençons avec une arithmétique ayant deux symboles et un polynôme $P(x)$ de degré m. Ensuite, nous introduisons un nouveau symbole, α, et nous supposons que $p(\alpha)=0$. Alors, il est possible de développer une table des puissances de α. Si nous choisissons le $p(x)$ correctement, les puissance de α de 0 à 2^{m-2} seront toutes différentes les unes des autres et $0, 1, \alpha, \alpha^2, \ldots \alpha^k$, où $k=2^{(m-2)}$ seront les éléments de l'ensembles du champs 2^m. En outre, chaque élément peut être exprimé comme une somme des *éléments* $1, \alpha, \alpha^2, \ldots \alpha^{m-1}$. Par exemple, pour $m=4$, $p(x) = x^4+x+1$ donnera la table de la figure A.1.

0	$\alpha^8 = \alpha^2+1$
1	$\alpha^9 = \alpha^3+\alpha$
α	$\alpha^{10} = \alpha^2+\alpha+1$
α^2	$\alpha^{11} = \alpha^3+\alpha^2+\alpha$
α^3	$\alpha^{12} = \alpha^3+\alpha^2+\alpha+1$
$\alpha^4 = \alpha + 1$	$\alpha^{13} = \alpha^3+\alpha^2+1$
$\alpha^5 = \alpha(\alpha+1) = \alpha^2+\alpha$	$\alpha^{14} = \alpha^3+1$
$\alpha^6 = \alpha(\alpha^2+\alpha) = \alpha^3+\alpha^2$	$\alpha^{15} = 1$
$\alpha^7 = \alpha^3+\alpha+1$	

Figure A.1 Le champs Galois de 2^4 éléments ($GF(2^4)$) $p(\alpha) = \alpha^4+\alpha+1=0$

L'élément α est appelé un élément primitif du champs $GF(2^m)$. En général, tout élément de $GF(2^m)$ dont la puissance génère tous les éléments non nul de $GF(2^m)$ est dit primitif. Par exemple, les puissances de α^4 de $GF(2^4)$ sont donnés dans la figure A.2:

$(\alpha^4)^0 = 1$
$(\alpha^4)^1 = \alpha^4$
$(\alpha^4)^2 = \alpha^8$
$(\alpha^4)^3 = \alpha^{12}$
$(\alpha^4)^4 = \alpha$
$(\alpha^4)^5 = \alpha^5$
$(\alpha^4)^6 = \alpha^9$
$(\alpha^4)^7 = \alpha^{13}$
$(\alpha^4)^8 = \alpha^2$
$(\alpha^4)^9 = \alpha^6$
$(\alpha^4)^{10} = \alpha^{10}$
$(\alpha^4)^{11} = \alpha^{14}$
$(\alpha^4)^{12} = \alpha^3$
$(\alpha^4)^{13} = \alpha^7$
$(\alpha^4)^{14} = \alpha^{11}$

Figure A.2 Les puissance de α^4 de $GF(2^4)$

On observe dans cette figure que tous les 15 éléments sont non-nul dans $GF(2^4)$. Alors α^4 est un élément primitif de $GF(2^4)$. Par contre, si nous faisions la même chose avec α^3 nous verrions que ce n'est pas un élément primitif.

Un polynôme $P(x)$ de degré m qui donne une table complète avec 2^m symboles distincts contenant des 0 et des 1 est appelé primitif. Il a été prouvé que pour chaque nombre entier positif m, il existe au moins un polynôme primitif de degré m. Ce n'est pas facile de reconnaître un polynôme primitif, mais il existe des listes comme celle que nous vous présentons à la figure A.3.

L'encodage des codes Reed-Solomon

Un code du Reed-Solomon est un code correcteur d'erreur à symboles cycliques. Un code RS est une séquence de bloc de champ fini $GF(2^m)$ de 2^m symboles binaires où m est le nombre de bits par symboles. Cette séquence de symboles peut être envisagée comme les coefficients d'un polynôme du code: $C(x) = c_0 + c_1x + c_2x^2 + \ldots + C_{n-1}x^{n-1}$ où l'élément du champs $c_i \in GF(2^m)$.

m	Polynomes primitifs
3	$1+x+x^3$
4	$1+x+x^4$
5	$1+x^2+x^5$
6	$1+x+x^6$
7	$1+x^3+x^7$
8	$1+x^2+x^3+x^4+x^8$
9	$1+x^4+x^9$
10	$1+x^3+x^{10}$
11	$1+x^2+x^{11}$
12	$1+x+x^4+x^6+x^{12}$
13	$1+x+x^3+x^4+x^{13}$
14	$1+x+x^6+x^{10}+x^{14}$
15	$1+x+x^{15}$
16	$1+x+x^3+x^{12}+x^{16}$
17	$1+x^3+x^{17}$
18	$1+x^7+x^{18}$
19	$1+x+x^2+x^5+x^{19}$
20	$1+x^3+x^{20}$
21	$1+x^2+x^{21}$
22	$1+x+x^{22}$
23	$1+x^5+x^{23}$
24	$1+x+x^2+x^7+x^{24}$

Figure A.3 Les polynomes primitifs

Un code RS(n,t) avec des symboles de GF(2m) a les paramètres suivants:

$n = 2m-1$	la longueur du code en symboles;
$k = n-2tf$	nombre de symboles des renseignements;
$n-k = 2t$	nombre de symboles de vérification;
$d_0 = 2t + 1 = d_{min}$	distance minimum;

où t est le nombre de symboles d'erreur pouvant être corrigés.

Considérons le code RS avec les symboles du code de GF(2m) où m est le nombre de bits par symbole.

Prenons $d(x) = c_{n-k}x^{n-k} + c_{n-k+1}x^{n-k+1} + ... + c_{n-1}x^{n-1}$ le polynôme information *et* $p(x) = c_0 + c_1 + ... + c_{n-k-1}x^{n-k-1}$ le polynôme de vérification. Alors, le polynôme RS codé s'exprime de la façon suivante:

$$c(x) = p(x) + d(x) = \sum_{i=0}^{n-1} c_i x^i, \qquad \text{(A-1)}$$

où c_i, $0 \le i \le n-1$ sont des éléments du champs $GF(2^m)$.

Donc un vecteur de symboles n $(c_0, c_1,...,c_{n-1})$ est un mot du code si et seulement si son polynôme correspondant $c(x)$ est un multiple du polynôme générateur $g(x)$. La méthode commune de chiffrer un code cyclique est de trouver le $p(x)$ à partir de $d(x)$ et $g(x)$ qui est accompli en divisant $d(x)$ par $g(x)$, ce qui résulte en $q(x)$, un quotient inutile et un reste important $\gamma(x)$. Cela donne $d(x) = q(x)g(x) + \gamma(x)$, donc $c(x) = p(x) + q(x)g(x) + \gamma(x)$.

Si nous définissons les bits de vérification ou les valeurs négatives du coefficient de $\gamma(x)$, c'est-à-dire $p(x) = -\gamma(x)$, cela entraîne $c(x) = q(x)g(x)$.

Cela assure que le code du polynôme $c(x)$ est un multiple de $g(x)$. Donc, l'encodeur RS exécutera une division pour obtenir le polynôme de vérification $p(x)$.

Prenons α, un élément primitif de $GF(2^m)$. Le polynôme générateur d'un code RS de longueur 2^m-1,corrigeant t-erreurs est défini par:

$$g(x) = (x+\alpha)+(x+\alpha^2)\ldots(x+\alpha^{2t}) = \sum_{i=0}^{2t} g_i x^i, \qquad \text{(A-2)}$$

où les coéfficients g_i, $0 \le i \le 2t$ sont de $GF(2^m)$. Un code $RS(n,t)$ produit par $g(x)$ est un $(n, n-2t)$ code cyclique dont les vecteurs du code sont multiples de $g(x)$.

Prenons par exemple un code RS ayant les paramètres $(15,11)$ avec des symboles provenant de $GF(2^4)$. Ce code doit pouvoir détecter deux erreurs. Donc, nous avons comme paramètres $m=4$, $t=2$ et $k = 15-2t = 11$ symbole information. Prenons α comme étant un polynôme primitif de $p(x)=1+x+x^4$ dans GF(2). Le polynôme générateur de ce code est:

$$\begin{aligned} g(x) &= (x+\alpha)(x+\alpha^2)(x+\alpha^3)(x+\alpha^4) \qquad \text{(A-3)} \\ &= \alpha^{10} + \alpha^3 x + \alpha^6 x^2 + \alpha^{13} x^3 + x^4 \end{aligned}$$

Le symbole information est $d_s = (a_0, a_1, a_2, a_3)$. Le symbole information peut être représenté sous forme de polynôme de la façon suivante:

$$d_s(\alpha) = a_0 + a_1\alpha + a_2\alpha^2 + a_3\alpha^3 \qquad \text{(A-4)}$$

Le résultat de la multiplication du symbole information $d_5(\alpha)$ par les coéfficients de $g(x)$ est le suivant:

$$\begin{aligned}
\alpha^{10}d_s(\alpha) &= (a_0 + a_2 + a_3) + (a_0 + a_1 + a_2)\alpha \\
&\quad + (a_0 + a_1 + a_2 + a_3)\alpha^2 + (a_1 + a_2 + a_3)\alpha^3 \qquad \text{(A-5)} \\
\alpha^{3}d_s(\alpha) &= a_1 + (a_1 + a_2)\alpha + (a_2 + a_3)\alpha^2 + (a_0 + a_3)\alpha^3 \\
\alpha^{6}d_s(\alpha) &= (a_1 + a_2) + (a_1 + a_3)\alpha + (a_0 + a_2)\alpha^2 + (a_0 + a_1 + a_3)\alpha^3 \\
\alpha^{13}d_s(\alpha) &= (a_0 + a_1 + a_2) + a_3 + a_0\alpha^2 + (a_0 + a_1)\alpha^3
\end{aligned}$$

Calcul du syndrome pour les codes RS

Nous définissons $\beta = \alpha^{j_l} \quad i \le l \le n$, comme les nombres d'emplacement d'erreurs quand le modèle d'erreur $e(x)$ contient v erreurs. Les $2t$ des composants du syndrome s_i ne sont pas seulement obtenus en substituant α_i, $i \le l \le 2t$, dans le polynôme reçu $r(x)$; ils peuvent aussi être calculés en divisant le $r(x)$ par $x+\alpha^i$. Les composants sont:

$$s_i = r(\alpha^i) = \sum_{l=1}^{v} e_{j_l}\,\alpha^{(j_l)i}. \qquad \text{(A-6)}$$

$$\text{où } r(x) = g_i(x)(x+\alpha^i) + \gamma_i. \qquad \text{(A-7)}$$

Le résultat de la division est:

$$s_i = r(\alpha^i) = \gamma_i. \qquad \text{(A-8)}$$

Par exemple, si nous avons le code RS (31,25) en utilisant GF(2^5) et détectant 3 erreurs. Nous recevons le polynôme $r(x) = \alpha^8x^2 + \alpha^2x^5 + \alpha x^{10}$ et le code transmis est composé de zéros, $c(x)=0$. Le calcul du syndrôme $(2t=6)$ à partir de $r(x)$ est le suivant:

$$\begin{aligned} s_1 &= r(\alpha) = \alpha^{10} + \alpha^7 + \alpha^{11} = \alpha \\ s_2 &= r(\alpha^2) = \alpha^{12} + \alpha^{12} + \alpha^{21} = \alpha^{21} \\ s_3 &= r(\alpha^3) = \alpha^{14} + \alpha^{17} + \alpha^{31} = \alpha^{23} \\ s_4 &= r(\alpha^4) = \alpha^{16} + \alpha^{22} + \alpha^{20} = \alpha^{15} \\ s_5 &= r(\alpha^5) = \alpha^{18} + \alpha^{27} + \alpha^{20} = \alpha^2 \\ s_6 &= r(\alpha^6) = \alpha^{20} + \alpha + \alpha^{30} = \alpha^{13}. \end{aligned} \tag{A-9}$$

Polynôme d'évaluation d'erreur et valeurs de l'erreur

Considérez le code RS avec les symboles du code de $GF(2^m)$, où m est le nombre de bits par symbole. Soit α un élément primitif dans $GF(2^m)$. Le polynôme générateur d'un code primitif RS correcteur de t-erreurs de longueur 2^m-1 est exprimé de la façon suivante :

$$g(x) = (x+\alpha)+(x+\alpha^2)\ldots(x+\alpha^{2t}) = \sum_{i=0}^{2t} g_i x^i. \tag{A-10}$$

où le g_i du coefficient, $0 \le i \le 2t$ sont de $GF(2^m)$.

Si $c(x)$ est le mot du code transmis et $r(x)$ est le mot correspondant reçu, alors le modèle de l'erreur causé par le bruit du canal devient:

$$e(x) = r(x)+c(x) = \sum_{i=0}^{n-1} (r_i+c_i)x^i = \sum_{i=0}^{n-1} e_i x^i. \tag{A-11}$$

où $e_i = r_i + c_i$, $0 \leq i \leq n-1$, est un symbole de $GF(2^m)$. Si le modèle d'erreur $e(x)$ contient v erreurs aux emplacements α_{jk}, $0 \leq k \leq v$ alors nous avons:

$$e(x) = e_{j1}x^{j1} + e_{j2}x^{j2} + \ldots + e_{jv}x^{jv}. \tag{A-12}$$

De façon à déterminer $e(x)$, nous devons connaître l'emplacement des erreurs x_{jk} ainsi que la valeur de l'erreur e_{jk}, pour $1 \leq k \leq v$.

De l'algorithme de Berkehamp, nous avons:

$$\sigma(x) = \prod_{l=1}^{v} (1 + \alpha^{jl}x) = 1 + \sigma_1 x + \sigma_2 x^2 + \ldots + \sigma_{v-1}x^{v-1} + \sigma_v x^v. \tag{A-13}$$

Prenons le polynôme du syndrome comme étant:

$$s(x) = s_1 x + s_2 x^2 + \ldots + s_v x^v = \sum_{\lambda=1}^{v} s_\lambda x^\lambda \quad , v \leq 2t. \tag{A-14}$$

Le polynôme évaluateur d'erreurs $\Omega(x)$ est défini comme le produit de $\sigma(x)$ et $s(x)$ tel que:

$$\Omega(x) = \sigma(x)s(x) = 1 + (s_1 + \sigma_1)x + (s2 + \sigma_1 s_1 + \sigma_2)x^2 + \ldots + (s_v + \sigma_1 s_{v-1} + \ldots + \sigma_v)x^v. \tag{A-15}$$

Pour les codes RS non-binaires, le polynôme évaluateur d'erreurs $\Omega(x)$ est utilisé pour trouver la valeur de l'erreur (ou la magnitude de l'erreur) pour chaque emplacement d'erreurs.

Supposons que v erreurs se sont produites aux emplacements correspondant aux indices $j_1 < j_2 < \ldots \leq n-1$. Alors, les composants du syndrome peuvent être exprimés comme:

$$s_\lambda = \sum_{k=1}^{v} e_{j_k} (\alpha^{j_k})^\lambda, \quad 1 \leq \lambda \leq 2t \tag{A-16}$$

où α_{jk} pour $k= 1,2,\ldots, v$ sont définis comme étant les nombres d'empalcement d'erreurs pour les positions ayant des indices j_k. considérons le polynôme du syndrome de degré infini tel que

$$s(x) = \sum_{\lambda=0}^{v} s_\lambda x^\lambda. \tag{A-17}$$

Après substitution, cela donne

$$s(x) = \sum_{k=1}^{v} e_{j_k} \sum_{\lambda=0}^{\infty} (\alpha^{j_k})^\lambda x^\lambda. \tag{A-18}$$

Si l'identité suivante est utilisée,

$$\sum_{\lambda=0}^{\infty} \alpha^{\lambda j_k} x^\lambda = 1/(1 + e^{j_k} x), \tag{A-19}$$

nous avons alors:

$$s(x) = \sum_{k=1}^{v} e_{j_k}/(1 + e^{j_k} x). \tag{A-20}$$

Le polynôme évaluateur $\Omega(x)$ de degré inférieur à v peut être écrit de la façon suivante:

$$\Omega(x) = \sum_{k=1}^{v} e_{j_k} \prod_{l=1,\, l \neq k}^{v} (1 + \alpha^{j_l} x). \tag{A-21}$$

Alors, la valeur d'erreur à l'emplacement $x = a^{-jm}$ est facilement obtenue par:

$$e_{j_m} = \Omega(\alpha^{-jm})/(\prod_{l=1,\, l\neq m}^{v} (1+\alpha^{jl}x+\alpha^{-jm})). \qquad \text{(A-22)}$$

Cette magnitude d'erreur est important, car le polynôme d'erreur *e(x)* est composé à partir de celui-ci.

Par exemple, prenons le code correcteur *RS (31,25)* ayant des symboles provenant de GF(2^5) et détectant 3 erreurs. Le polynôme générateur de ce code est:

$$\begin{aligned} g(x) &= (x+\alpha)(x+\alpha^2)(x+\alpha^3)(x+\alpha^4)(x+\alpha^5)(x+\alpha^6) \\ &= \alpha^{21} + \alpha^{24}x + \alpha^{16}x^2 + \alpha^{24}x^3 + \alpha^9x^4 + \alpha^{10}x^5 + x^6. \end{aligned} \qquad \text{(A-23)}$$

Nous recevons le vecteur suivant:

$$r = (0\,0\,\alpha^8\,0\,0\,\alpha^2\,0\,0\,0\,0\,\alpha\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0). \qquad \text{(A-24)}$$

Comme nous l'avons trouvé dans un exemple précédent, le syndrôme est le suivant:

$$s = (\alpha,\ \alpha^{21},\ \alpha^{23},\ \alpha^{15},\ \alpha^2,\ \alpha^{13}). \qquad \text{(A-25)}$$

Le polynôme locateur d'erreur $\sigma(x)$ peut être trouvé en utilisant l'algorithme itératif suivant:

1. $n = 0,\ m = -1$

$$\begin{aligned} \sigma^{(1)}(x) &= \sigma^{(0)}(x) - d_0 d_{-1}^{-1} x \sigma^{(-1)}(x) \\ &= 1 + \alpha x \end{aligned} \qquad \text{(A-26)}$$

$$d_1 = s_2 + s_1\sigma_1^{(1)} = \alpha^{21} + \alpha^2 = \alpha^{13}$$

2. $n = 1, m = 0$

$$\sigma^{(2)}(x) = \sigma^{(1)}(x) - d_1 d_0^{-1} x \sigma^{(0)}(x)$$
$$= 1 + \alpha x + \alpha^{13}\alpha^{-1} x$$
$$= 1 + \alpha^{20} x$$
$$d_2 = s_3 + s_2 \sigma_1^{(2)} = \alpha^{23} + \alpha^{10} = \alpha^{24}$$

3. $n = 2, m = 0$

$$\sigma^{(3)}(x) = \sigma^{(2)}(x) - d_2 d_0^{-1} x^2 \sigma^{(0)}(x)$$
$$= 1 + \alpha^{20} x + \alpha^{24}\alpha^{-1} x^2$$
$$= 1 + \alpha^{20} x + \alpha^{23} x^2$$
$$d_3 = s_4 + s_3 \sigma_1^{(3)} + s_2 \sigma_2^{(3)}$$
$$= \alpha^{15} + \alpha^{12} + \alpha^{13} = \alpha^{8}$$

4. $n = 3, m = 2$

$$\sigma^{(4)}(x) = \sigma^{(3)}(x) - d_3 d_2^{-1} x \sigma^{(2)}(x)$$
$$= 1 + \alpha^{20} x + \alpha^{23} x^2 + \alpha^{15} x + \alpha^{4} x^2$$
$$= 1 + \alpha^{17} x + \alpha^{15} x^2$$
$$d_4 = s_5 + s_4 \sigma_1^{(4)} + s_3 \sigma_2^{(4)}$$
$$= \alpha^{2} + \alpha + \alpha^{7} = \alpha^{30}$$

5. $n = 4, m = 2$

$$\sigma^{(5)}(x) = \sigma^{(4)}(x) - d_4 d_2^{-1} x^2 \sigma^{(2)}(x)$$
$$= 1 + \alpha^{17} x + \alpha^{22} x^2 + \alpha^{26} x^3$$
$$= 1 + \alpha^{17} x + \alpha^{15} x^2$$
$$d_5 = s_6 + s_5 \sigma_1^{(5)} + s_4 \sigma_2^{(5)} + s_3 \sigma_3^{(5)}$$
$$= \alpha^{13} + \alpha^{19} + \alpha^{6} + \alpha^{18} = \alpha^{17}$$

6. $n = 5, m = 4$

$$\sigma^{(6)}(x) = \sigma^{(5)}(x) - d_5 d_4^{-1} x \sigma^{(4)}(x)$$
$$= 1 + \alpha^4 x + \alpha^5 x^2 + \alpha^{17} x^3$$

Puisque $\sigma(x) = \sigma^{(6)}(x)$, le polynôme locateur d'erreurs est:

$$\sigma(x) = 1 + \alpha^4 x + \alpha^5 x^2 + \alpha^{17} x^3$$

où $\sigma_0 = 1$, $\sigma_1 = \alpha^4$, $\sigma_2 = \alpha^5$ et $\sigma_3 = \alpha^{17}$.

Après substitution, nous trouvons que α^{21}, α^{26} et α^{29} sont des racines de $\sigma(x)$. Les réciproques de ces racines sont les nombres locateurs d'erreurs de $e(x)$. Ces nombres sont α^2, α^5 et α^{10}. Il est alors possible de trouver la valeur des erreurs aux positions x^2, x^5 et x^{10} en calculant le polynôme évaluateur d'erreurs $\Omega(x)$. Le polynôme évaluateur d'erreur est le suivant:

$$\Omega(x) = 1 + (\alpha + \alpha^4)x + (\alpha^{21} + \alpha^4\alpha + \alpha^5)x^2 + (\alpha^{23} + \alpha^4\alpha^{21} + \alpha^5\alpha + \alpha^{17})x^3 \qquad \text{(A-27)}$$
$$= 1 + \alpha^{30}x + \alpha^{21}x^2 + \alpha^{23}x^3.$$

Avec les nombres de locations d'erreurs, il est possible de déterminer la valeur des erreurs avec les équations suivantes:

$$e_2 = \Omega(\alpha^{-2})/((1+\alpha^5\alpha^{-5})(1+\alpha^{10}\alpha^{-2})) = \alpha^{26}/\alpha^{18} \qquad \text{(A-28)}$$

$$e_5 = \Omega(\alpha^{-5})/((1+\alpha^2\alpha^{-5})(1+\alpha^{10}\alpha^{-5})) = \alpha^{30}/\alpha^{28} \qquad \text{(A-29)}$$

$$e_{10} = \Omega(\alpha^{-10})/((1+\alpha^2\alpha^{-10})(1+\alpha^5\alpha^{-10})) = \alpha^{10}/\alpha^{9} \qquad \text{(A-30)}$$

De cette façon, le polynôme d'erreur est facile à trouver :

$$e(x) = e_2x^2 + e_5x^5 + e_{10}x^{10} \qquad \text{(A-31)}$$
$$= \alpha^8x^2 + \alpha^2x^5 + \alpha x^{10}$$

Le processus de décodage est terminé en prenant $c(x) = r(x) + e(x)$ ce qui devrait donner le vecteur qui a été transmis.

Solution de s (x) par algorithme Berlehamp

La manière de résoudre le problème de trouver la solution de degré minimale pour le polynôme d'emplacement d'erreurs se comprend plus facilement en suivant les opérations itératives suivantes :

1. Conditions initiales avant de commencer les itérations.

$$\sigma^{(-1)}(x) = 1 \qquad l_{-1} = 0 \qquad d_{-1} = 1$$

$$\sigma^{(0)}(x) = 1 \qquad l_0 = 0 \qquad d_0 = s_1$$

2. Si $d_n=0$, alors $\sigma^{(n+1)}(x) = \sigma^{(n)}(x)$ et $l_{n+1} = l_n$.

3. Si $d_n\neq 0$, trouver $\sigma^{(m)}(x)$ avant $\sigma^{(n)}(x)$ de façon à ce que $d_m \neq 0$, $1 \leq m \leq n$, et le nombre $m - l_m$ a la plus grande valeur. Alors

$$\sigma^{(n+1)}(x) = \sigma^{(n)}(x) - d_n d_m^{-1} x^{n-m} \sigma^{(m)}(x) \text{ et} \qquad \text{(A-19)}$$

$$ln+1 = max [l_n, l_m + n - m]. \qquad \text{(A-20)}$$

4. Pour $d_n=0$ ou $d_n\neq 0$, la prochaine étape est:

$$d_{n+1} = s_{n+2} + s_{n+1}\,\sigma_1^{(n+1)} + \ldots + s_{n+2-ln+1}\,\sigma_{ln+1}^{(n+1)} \qquad (4\text{-}21)$$

où $\sigma_i^{(n+1)}$, $1 \le i \le l_{n+1}$, sont les coéfficients de $\sigma^{(n+1)}(x)$. (4-22)

Annexe B

Un encodeur convolutionnel et un décodeur Treillis utilisant l'algorithme Viterbi

L'information sur l'encodeur convolutionnel et le décodeur Viterbi proviennent de [BDMM99] [CLLC96] [Flem95] [JGIT99] [KsHo94] [LuCa96] [Rese99] [TCL99] [Vita99][Vite99].

Encodeur convolutionnel

Un code convolutionnel est différent d'un code bloc comme le Reed-Solomon. La différence est au niveau de la mémorisation des données précédentes pour le traitement d'un code. C'est-à-dire que les n sorties de l'encodeur ne dépendent pas seulement des k entrées, mais également des m entrées précédentes au bloc. Un code convolutionnel (n,k,m) peut implanter k-entrées, n-sorties avec comme entrée une mémoire de m. Typiquement n et k sont de petits entiers et $k<n$, mais l'ordre de la mémoire m doit être assez grand si l'on veut atteindre une faible probabilité d'erreur. Dans les sections suivantes, nous allons vous présenter deux façons simples d'encoder un code convolutionnel.

L'approche register-stage

Un encodeur convolutionnel avec une contrainte de longueur de n_A consiste à un registre à décalage de m-étages avec n-modulo-2 additionneur et un multiplexeur parallèle-séries pour sérialiser les sorties de l'encodeur en une simple séquence de code. Alors, les données d'entrée de l'encodeur qui sont appelées la séquence information, sont décalées dans le registre k bits à la fois, et les séquences de sortie de l'encodeur sont obtenues en prenant la convolution de la séquence d'information avec les séquences génératrices (la réponse d'impulsion du décodeur) du code. Donc, un encodeur convolutionnel traite les bits d'information d'une manière continuelle de façon sérielle pour la transmission sur un canal. La matrice génératrice G du code est une matrice semi-infinie et possède un nombre infinie de colonnes et de rangées, ce qui

permet aux séquences information et mot-code d'être arbitrairement très larges. Toutefois, pour un traitement pratique, il y a un maximum permis de longueur L pour lequel on définit souvent la $L^{ième}$ troncature d'un code convolutionnel. Donc, la séquence information consiste à kL bits et la séquence code est représentée par $n(m+L)$ bits.

Généralement k et n pour $k<n$, sont de petits entiers. La contrainte de longueur est définie comme $n_A=(m+1)n$ parce que c'est le nombre maximum des sorties de l'encodeur qui peuvent être affectés à un simple bit information. Puisque qu'un encodeur convolutionnel génère n bits encodés pour chaque k bits information, le taux du code est de $R=k/n$. Le taux du code tronqué est donné par :

$$R_T = kL/(n(m+L)) = R(1-(m/(m+L))). \tag{B-1}$$

Puisque $L >> m$ dans la plupart des situations pratiques, le taux R_T sera très près de R. Ceci est la raison pour laquelle R est appelé le taux du code convolutionnel.

Un code convolutionnel ayant un taux $R=k/n$ avec une contrainte de longueur $n_A=(m+1)n$ est décrit comme un ensemble de n séquences génératrices

$$g_i^{(j)} = (g_{i,0}^{(j)}, g_{i,1}^{(j)}, \cdots g_{i,m-1}^{(j)}, g_{i,m}^{(j)}). \tag{B-2}$$

Pour $i=1,2,...,k$ et $j=1,2,...,n$. Les séquences génératrices peuvent être interprétées comme les n réponses d'impulsion de l'encodeur lorsque la séquence information *est* $d=(1\ 0\ 0\ ...)$.

Si la séquence information $d^{(i)} = (d_0^{(i)}, d_1^{(i)}, d_2^{(i)},...)$ entre dans le décodeur un bit à la fois, alors la séquence de sortie du décodeur $c^{(j)} = (c_0^{(j)}, c_1^{(j)}, c_2^{(j)},...)$ peut être

obtenue en combinant la convolution discrète de la séquence information $d^{(i)}$ avec la séquence génératrice $g_i^{(j)}$ de façon à ce que :

$$c_\lambda^{(j)} = \sum_{l=0}^{m} [\sum_{i=1}^{k} d_{\lambda-l}^{(i)} * g_{il}^{(j)}] \qquad pour\ 0 \le l \le \lambda\ \ et\ 1 \le j \le n. \tag{B-3}$$

Ceci est généralement appelé l'équation d'encodage.

Pour donner un exemple simple, le $R=1/2$ encodeur convolutionnel pour un code binaire $(2,1)$ avec $m=2$ est montré à la figure B.1. Puisque m=2 et n=2, cet encodeur consiste en un registre à décalage de 2 étages avec deux additionneurs modulo-2 et un multiplexeur pour sérialiser les sorties de l'encodeur.

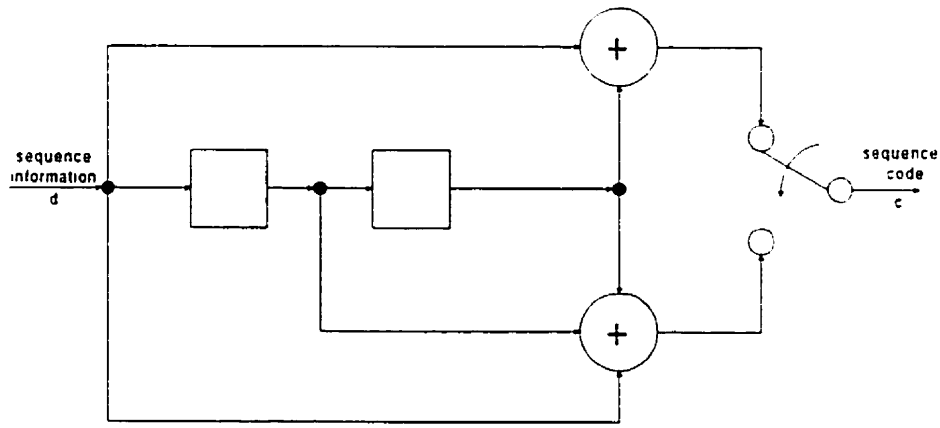


Figure B.1 Encodeur pour un code convolutionnel (2,1) avec m=2

L'approche matricielle

Considérons un code convolutionnel (n,k) avec une mémoire de l'ordre de m. Si une séquence information $d=(d_0,d_1,d_2,...)$ où $d_\lambda = (d_\lambda^{(1)}, d_\lambda^{(2)}, \ldots, d_\lambda^{(k)})$ pour $\lambda=0,1,2,...$ entre dans l'encodeur, alors la séquence mot-code $c= (c_0, c_1, c_2,...)$ dans la quelle $c_\lambda =$

$(c_\lambda^{(1)}, c_\lambda^{(2)}, \ldots, c_\lambda^{(k)})$ pour $\lambda=0,1,2,\ldots$ est donnée par l'équation suivante sous la forme d'une matrice : $c = d*G$

$$c^{(j)} = \sum_{i=1}^{k} d^{(i)} * g_i^{(j)} \qquad j = 1,2,\ldots,n, \tag{B-4}$$

où G est un matrice génératrice semi-infinie dont la structure est généralement donnée par

$$G = \begin{matrix} G_0 & G_1 & G_2 & \ldots & G_m & 0 & 0 \\ 0 & G_0 & G_1 & \ldots & G_{m-1} & G_m & 0 \\ 0 & 0 & G_0 & \ldots & G_{m-2} & G_{m-1} & G_m \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \end{matrix} \tag{B-5}$$

Les autres entrées dans G sont toutes des zéros. Chaque matrice sous-génératrice G_λ, $\lambda=1,2,\ldots, m$ dans G contient k lignes et n colonnes. Toutefois, chaque ensemble de k lignes sont identiques à l'ensemble précédent de k lignes mais sont décalées de n colonnes vers la droite. Chaque $k \times n$ matrice sous-génératrice G_λ dans G est exprimée par :

$$G_\lambda = \begin{matrix} G_{1\lambda}^{(1)} & G_{1\lambda}^{(2)} & \ldots & G_{1\lambda}^{(n)} \\ G_{2\lambda}^{(1)} & G_{2\lambda}^{(2)} & \ldots & G_{2\lambda}^{(n)} \\ \ldots & \ldots & \ldots & \ldots \\ G_{k\lambda}^{(1)} & G_{k\lambda}^{(2)} & \ldots & G_{k\lambda}^{(n)} \end{matrix} \tag{B-6}$$

La structure du code

Un encodeur convolutionnel peut être représenté comme une machine à états. La machine à états correspondant à la figure B-1 est représenté dans la tableau B.1 et la figure B.2. Il y a toujours deux branches quittant chacun des états. Une branche correspond à l'entrée 0, tandis que l'autre branche correspond à l'entre 1. Chaque branche est étiquettée avec un 1 ou un 0, et la sortie comportant deux bits correspond à la transition d'état.

Puisqu'un encodeur peut être vu comme une machine à états finis, sa structure peut être représentée à l'aide d'un diagramme d'états, d'arbre et de treillis. La manière répétitive de représenter les données se prête bien à un diagramme treillis, comme nous pouvons le voir à la figure B.3.

Tableau B.1 La machine à états

État présent	**Entrée**			
	0		**1**	
	Sortie	État de transition	Sortie	État de transition
0 0	0 0	0 0	1 1	1 0
0 1	1 1	0 0	0 0	1 0
1 1	1 0	0 1	0 1	1 1
1 0	0 1	0 1	1 0	1 1

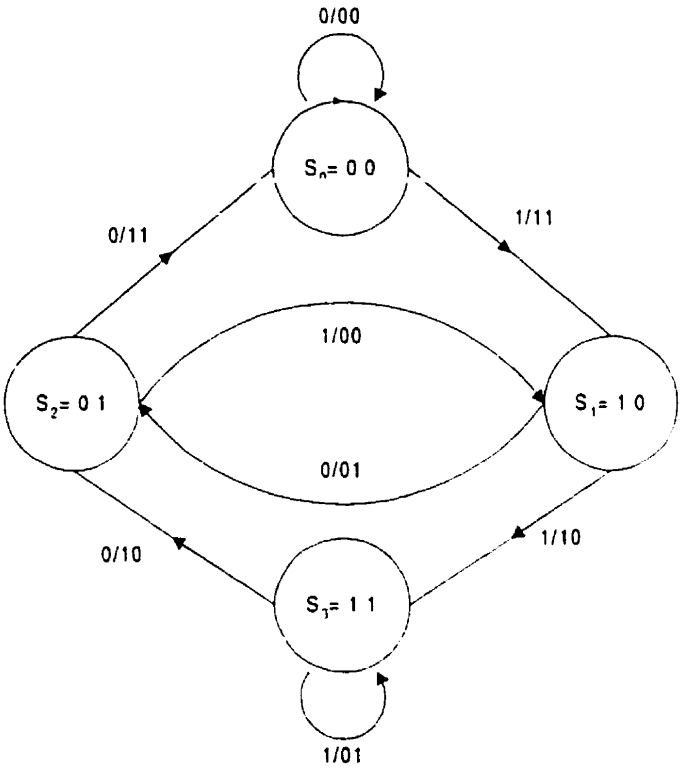


Figure B.2 Encodeur du diagramme d'état du tableau B.1

3.0 Le décodeur Viterbi

En 1967, Viterbi[4] propose un algorithme de décodage de maximum de vraisemblance (maximum likelihhood decoding) qui est relativement facile à implanter avec des codes ayant un ordre de mémoire relativement petit.

Dans l'algorithme Viterbi, lorsque le décodage se fait sur un BSC (Binary Synchronous Communication) ou en français, « communication synchrone binaire », la distance Hamming est utilisée (figure B.4). Le but de l'algorithme est de trouver le chemin ayant la plus petite distance Hamming à travers le treillis en comparant les

distances Hamming de toutes les branches du chemin entrant dans chaque état. Lors du processus de décodage, si à un moment donné, on trouve qu'il sera impossible d'atteindre de cette façon la plus petite distance Hamming, alors ce chemin est éliminé. Donc, le décodeur compare tout les distances de tous les chemins entrant dans un état et garde seulement le survivant, celui qui mène au chemin qui a le plus de probabilité d'être le bon.

Voici les trois étapes utilisées à l'intérieur de l'algorithme Viterbi. La première étape concerne la tête du treillis, la deuxième le corps, tandis que la troisième étape concerne la queue du treillis.

Étape 1 : Jusqu'au niveau $0<j\leq m$, calculer la distance Hamming de r à la branche entrant dans chaque état et sauvegarder l'addition des distances pour chaque état.

Étape 2 : Pour $m<j\leq L$, calculer la distance Hamming en additionnant la distance du survivant au niveau précédent à sa distance de r à la prochaine branche.

Étape 3 : Pour $L<j\leq L+m$, dans la portion de la queue du treillis, il y a peu d'états, parce que l'encodeur retourne dans l'état comprenant des zéros. Répéter l'étape 2 pour chaque état.

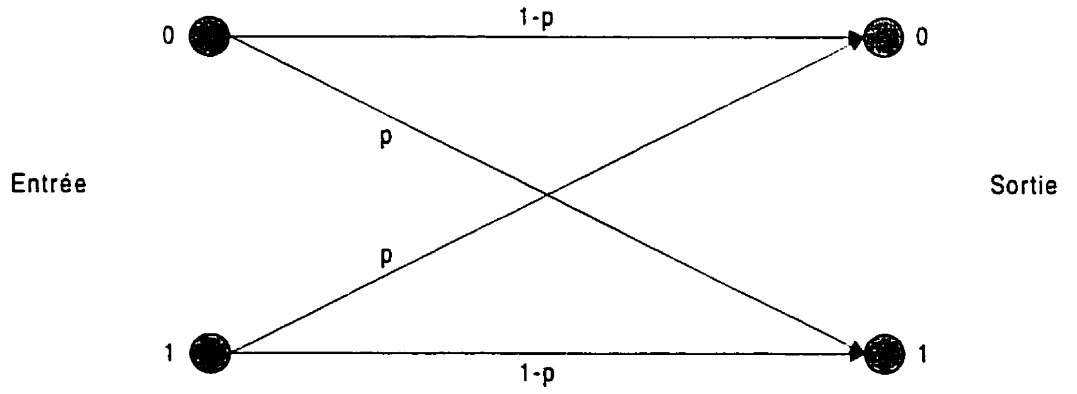


Figure B.3 Diagramme treillis pour un code convolutionnel

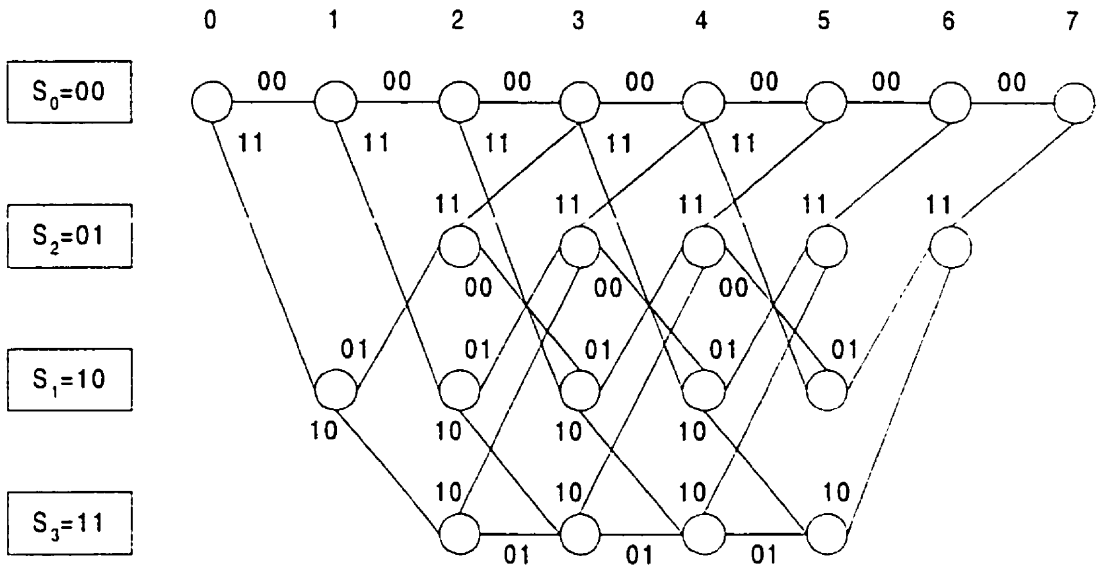


Figure B.4 Représentation BSC

La figure B.5 représente un exemple d'un décodage en utilisant l'algorithme Viterbi. La séquence information d = (1011100) doit être transmise. Elle est donc encodée et la séquence code *(c)* est (1111, 0111, 0000, 1000, 0111, 1000, 1111). La séquence reçue *(r)* diffère à trois endroits de la séquence code envoyée, c'est-à-dire aux positions 2, 5 et 13. Ces erreurs sont représentées par un *x* entre les rangées *c* et *r* dans le haut de l'exemple. Nous partons tout d'abord dans l'état S_0 dans le coin supérieur gauche du treillis. Nous recevons la séquence de bits 1011 et nous avons le choix entre deux branches du treillis. La première est 0000 entraînant une distance Hamming de 3 et la seconde 1111, entraînant une distance Hamming de 1. Donc, le meilleur choix est de prendre le chemin 1111. L'autre chemin est alors éliminé. Puis à partir du nouvel

d	1	0	1	1	1	0	0
c	1111	0111	0000	1000	0111	1000	1111
	x	x		x			
r	1011	1111	0000	0000	0111	1000	1111

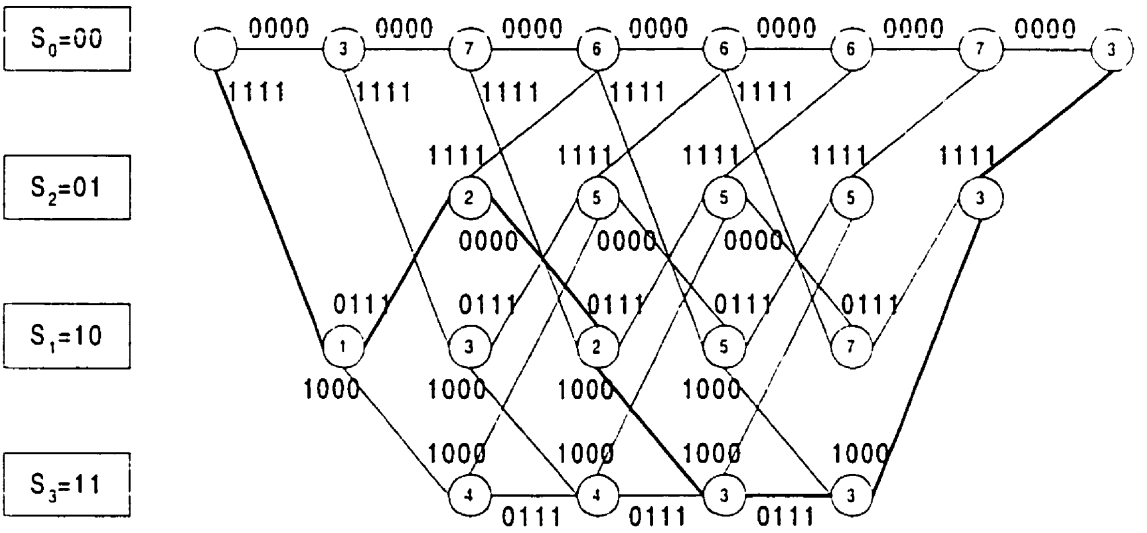


Figure B.5 Décodage Viterbi pour BSC

état qui est équilvalent à S_1. Nous avons le choix entre deux autres chemin du treillis. c'est à dire le chemin 0111 ayant une distance Hamming de 1 et le chemin 1000 ayant une distance Hamming de 3. Et ainsi de suite pour décoder le reste des valeurs. La valeur à l'intérieur de chacun des états du treillis indique la somme des distance Hamming obtenues pour se rendre jusqu'à cet état. Il est possible de voir que dans le dernier état du treillis le chiffre 3 est indiqué. Cela signifie que 3 erreurs ont été détectées et corrigées en utilisant le décodeur Viterbi. Cela correspond bien aux trois erreurs envoyées lors de la réception.

Annexe C

Table d'allocation des bits

Comment déterminer le nombre de bits par porteur?

Dans cette section, nous allons dériver les expressions pour le nombre de bits pouvant être transmis dans chaque bloc DMT. Le signal de sortie de chaque sous-canaux est considéré comme étant un signal QAM. Prenons K comme la fréquence positive utilisée par un sous-canal et assumons un constellation de eux dimensions L_k-QAM pour le $k^{ième}$ sous-canal de l'ensemble. Alors le nombre de bits par bloc DMT est donné par [DSL99][QAM99]:

$$b = \sum_{k \in K} \log_2 L_k \tag{C-1}$$

Le probabilité d'erreur du symbole du $k^{ième}$ sous-canal est approximé par:

$$P_e = 4\Phi(d_{min,k}/(2\sigma_k)). \tag{C-2}$$

où $\Phi(\cdot)$ dénote l'intégral de probabilité, $d_{min,k}$ est la distance minimale entre les constellations des sous-canaux à la sortie du canal, et σ_k^2 est la variance du bruit par dimension sur le $k^{ième}$ sous-canal. Il est assumé que le bruit PSD $S_N(f_k)$ est approximativement plat sur le $k^{ième}$ sous-canal. Dans ce cas, $\sigma_k^2 = S_N(f_k)f_s/N$.

La distance minimale enter le point de constellation du sous-canal à la sortie du canal est donnée par $d_{min,k}^2 = d_k^2/|H_A(f_k)|^2$, où d_k est la distance entre les points de constellation au transmetteur. De façon à maintenir la même probabilité d'erreur du symbole par dimension $(P/2)$ sur chaque sous-canal, nous devons avoir

$$\gamma = \left(\frac{d_{min,k}}{2\sigma_k}\right) \tag{C-3}$$

constant. γest habituellement référé comme étant le SNR requis. Par exemple, si $P_e/2 = 10\text{-}7$, nous avons alors de besoin de $\gamma = 14.5$ dB. Notez qu'une marge Δ_M peut être ajoutée à γ, pour un affaiblissement imprévu du cannal ou pour le gain de codage de tout code appliqué (par exemple, un encodage treillis).

L'énergie moyenne d'un sous-symbole (ou puissance normalisée) P_k dans un constellation L_k-QAM est donné par:

$$P_k = (L_k - 1)/6 \cdot d_k^2. \tag{C-4}$$

La puissance totale transmise:

$$P_T = \sum_{k \in K} P_k/R_L. \tag{C-5}$$

où R_L est la résistance d'arrêt de chargement du xDSL. En substituant dans les équations précédentes, nous obtenons l'expression suivante pour le nombre de points dans une constellation QAM sur $k^{ième}$ sous-canal.

$$L_k = 1 + ((3P_k |H_A(f_k)|^2)/2\sigma_k^2\gamma). \tag{C-6}$$

Définissions un interval SNR, Γ, qui est associé l'exécution du schema DMT considérant la capacité Shannon du cananl, de façon à ce que $3\Gamma = \gamma$. Si nous définisson le SNR du $k^{ième}$ sous-canal comme étant $SNR_k = P_k |H_A(f_k)|^2 / 2\sigma_k^2$, nous pouvons alors démontrer que le nombre maximal de bits que peut supporteur un sous-canal est:

$$P_T = \sum_{k \in K} \log_2(1 + ((P_k |H_A(f_k)|^2)/2\sigma^2\Gamma)). \tag{C-7}$$

où $\Gamma = 9.8 + \Delta_M - \Delta_c\ dB$. Ceci est la capacité du sous-canal avec un facteur de Γ soustrait du SNR qui atteint la capacité Shannon. Le nombre de bits par symbole peut être exprimé par:

$$b_k = \log_2(1 + (SNR_k/\Gamma)). \tag{C-8}$$

Annexe D

Résultats de dissipation pour le TMS320C54

Exemples de routines de test

Comme nous l'avons expliqué au chapitre 4, nous avons utilisé deux modes différents pour tester le courant utilisé par chacune des instructions. Le mode « Repeat » utilise une boucle pour répéter plusieurs fois l'instruction, tandis que le mode « Inline » répète l'instruction le nombre de fois voulu, sans utiliser de boucle. Voici un exemple de routines de test pour chacun des deux modes.

En mode « Repeat »

```
.title  "Loop test"
        .mmregs
        .width  80
        .length  55

        .setsect ".text",0x1800,0
        .sect".text"

       SP  = #0ffah

start:    AR3 = #100h
          a = #0
          REPEAT(#120)
          a = a + *ar3
          goto start
.end
```

En mode « Inline »

```
.title  "Loop test"
        .mmregs
        .width  80
        .length  55
```

```
        .setsect ".text",0x1800,0
        .sect".text"

       SP  = #0ffah

start:  AR3 = #100h
        a = #0
        a = a + *ar3
        a = a + *ar3
        a = a + *ar3
...                          120 instructions
...
        a = a + *ar3
        goto start
.end
```

1.3 Pour les effets inter-instructions

```
.title  "Loop test"
        .mmregs
        .width  80
        .length  55

        .setsect ".text",0x1800,0
        .sect".text"

       SP  = #0ffah

start:  a = #0fffffffcbh
        @brc = #60
        BLOCKREPEAT(end_block-1)
        a = lal
        a = max(a,b)
end_block:
        goto start
.end
```

Résultats des expériences pour le calcul de base des instructions

Voici les résultats obtenus pour le calcul de base des instructions (tableau E.1). Aucune dépendance n'a été considérée. Dans la colonne Instructions, le chiffre entre parenthèse représente le nombre de cycles de l'instruction.

Tableau E.1 Consommation de base des instructions

Instructions	Mode «Repeat» (mA)	Mode « Inline » (mA)	Commentaires
Fonctions sur registres auxiliaires			
Mar (1)	64	83	mar(*ar3+)
Fonctions arithmétiques			
Abs (1)	66	91	a = lal
Cmps (1)	77	102	cmps(a,*ar3)
Add (2)	67	94	a = a + #01h
Add (1)	79	105	a = a + *ar3
Max (1)	65	90	a = max(a,b)
Min (1)	65	90	a = min(a,b)
Neg (1)	78	104	a = -a
Sat (1)	64	90	saturate(a)
Sub (1)	79	105	a = a - *ar3
Sub (1)	89	115	a = a - *ar3+
Sub (2)	67	107	b = b - #012h
Fonctions de multiplication			
Macd (3)	91	107	macd(*ar4-,1234,a)
Macp (2)	82	100	macp(*ar4-,1234,a)
Mul (2)	87	117	a = a + *ar5+ * #1234h
Square (1)	65	90	b = square(hi(a))
Fonctions spéciales de multiplication			
Sqdst (1)	116	141	sqdst(*ar3+,*ar4+)
Firs (2)	97	109	firs(*ar3+,*ar4+,1234)
Lms (1)	114	139	lms(*ar3+,*ar4+)
Poly (1)	104	130	poly(*ar3+%)
Fonctions mémoire données			
Delay (1)	89	100	delay(*AR3) copy data in the next higher adress
Pop (1)	82	102	ar5 = pop()
Push (1)	105	124	push(*ar5+)
Prog (4)	65	90	prog(0FE00h) = @0
Data (2)	79	101	BK = data(300h)
Opérations logiques			
And (1)	78	103	a = a & *ar3
And (1)	66	92	b = b & a
Or (1)	77	103	a = a | *ar3

Tableau E.1 (suite)			
Instructions	**Mode «Repeat» (mA)**	**Mode « Inline » (mA)**	**Commentaires**
Xor (1)	79	104	a = a ^ *ar3
Shift (1)	65	90	a = a \\ CARRY
Test (1)	86	110	TC = bit(*ar3+,15-12)
Fonctions de chargement et stockage			
Load (1)	66	82	b = #248
Load (1)	77	102	b = *ar3
Load (1)	64	90	b = a
Load (1)	65	96	t =#1000
Store (1)	77	102	*ar3 = b
Store (1)	65	91	b = t
Store (1)	84	105	*ar2+ = hi(a)
Store (1)	90	109	*ar2 = *ar3
Store (1)	110	129	*ar2+ = *ar3+
Store (1)	66	85	ar3 = #100

Effets inter-instructions

Le tableau E.2 montre les résultats obtenus pour les effets inter-instructions. Le tableau n'est pas complet, mais c'est quelque chose qui pourrait être éventuellement développé.

Tableau E.2 Les effets inter-instructions				
Première instruction	**Seconde instruction**	**Moyenne des courants pondérée (en mA)**	**Courant réel (en mA)**	**Écart (en %)**
Abs (91)	Cmps (102)	96.5	111	13
	Add (94)	93	102	9
	Add (105)	98	98	0
	Max (90)	90.5	97	7
	Min (90)	90.5	97	7
	Neg (104)	97.5	98	1
	Sat (90)	90.5	95	5
	Sub (115)	103	122	16
	Sub (107)	101.5	107	5
	Macd (107)	103	104	1
	Macp (100)	98	102	4
	Sqdst (116)	103.5	132	22
	Mar (83)	87	101	14
	Square (90)	90.5	94	4
	Lms (139)	115	133	15
	Delay (100)	95.5	118	20
	Poly (130)	110.5	123	10
	And (103)	97	110	12
	And (92)	91.5	98	7
	Or (103)	97	111	13
	Xor (104)	97.5	110	11
	Shift (90)	90.5	97	7
	Test (110)	100.5	112	10
	Load (82)	86.5	103	16
	Load (102)	96.5	114	16
	Load (90)	90.5	98	8
	Load (96)	93.5	100	7
	Store (102)	96.5	110	12
	Store (91)	91	107	15
	Store (109)	100	110	9
	Store (85)	88	98	10
	Prog (90)	90.5	91	1
	Data (101)	97.5	103	5

Tableau E.2 (suite)				
Première instruction	**Seconde instruction**	**Moyenne des courants pondérée (en mA)**	**Courant réel (en mA)**	**Écart (en %)**
Max (90)	Cmps (102)	96	110	13
	Add (94)	92.5	123	25
	Add (105)	97.5	104	6
	Abs (91)	90	97	7
	Min (90)	90	91	1
	Neg (104)	97	110	12
	Sat (64)	90	108	17
	Sub (115)	102.5	127	19
	Sub (107)	101	101	0
	Macd (107)	102.5	105	2
	Macp (100)	97.5	100	2
	Sqdst (116)	103	141	27
	Mar (83)	86.5	102	15
	Square (90)	90	98	8
	Lms (139)	114.5	136	16
	Delay (100)	95	118	19
	Poly (130)	110	127	13
	And (103)	96.5	123	22
	And (92)	91	105	13
	Or (103)	96.5	124	22
	Xor (104)	97	122	20
	Shift (90)	90	99	9
	Test (110)	100	111	10
	Load (82)	86	110	22
	Load (102)	96	120	20
	Load (90)	90	107	16
	Load (96)	93	98	5
	Store (102)	96	110	13
	Store (91)	90.5	110	18
	Store (109)	99.5	110	10
	Store (85)	87.5	88	1
	Prog (90)	90	90	0
	Data (101)	97	104	7

Tableau E.2 (suite)				
Première instruction	**Seconde instruction**	**Moyenne des courants pondérée (en mA)**	**Courant réel (en mA)**	**Écart (en %)**
Macd (107)	Cmps (102)	106	116	9
	Add (94)	102	115	11
	Add (105)	106.5	107	1
	Max (90)	103	104	1
	Min (90)	103	105	1
	Neg (104)	106	107	1
	Sat (90)	103	105	1
	Sub (115)	109	116	6
	Sub (107)	107	107	0
	Abs (91)	103	105	1
	Macp (100)	103.5	127	19
	Sqdst (116)	109	109	0
	Mar (83)	101	107	6
	Square (90)	103	109	6
	Lms (139)	115	128	10
	Delay (100)	105	121	13
	Poly (130)	113	124	9
	And (103)	106	113	6
	And (92)	103	103	0
	Or (103)	106	112	5
	Xor (104)	106	113	5
	Shift (90)	103	104	1
	Test (110)	108	113	4
	Load (82)	101	104	3
	Load (102)	106	112	5
	Load (90)	103	103	0
	Load (96)	104	104	0
	Store (102)	106	114	7
	Store (91)	103	106	3
	Store (109)	107.5	115	6.5
	Store (85)	101.5	107	5
	Prog (90)	97	100	3
	Data (101)	104.5	108	3

La colonne appelée « Moyenne des courants pondérée » indique la valeur calculée avec les coûts de base. La valeur est pondérée selon le nombre de cycle de chacune des instructions. Puis, lacolonne « courant réel » indique le courant utilisé réellement lorsque les deux instructions se suivent. Il est bon de remarquer que si l'ordre des instructions est changé, le courant utilisé n'est pas nécessairement le même.

Annexe E

Programmes de tests pour la dissipation de puissance du C54.

Programme 1

```
.mmregs
      .width  80
      .length  55

      .setsect ".text",0x1800,0
      .sect".text"

     SP  = #0ffah

start:    a = #0fffffffcbh
          @brc = #60
          AR3 = #100h
          BLOCKREPEAT(end_block-1)

          a = lal
          a = max(a,b)
          a = a - *ar3
          macp(*ar3-,1234,a)
          a = -a
          push(*ar3+)
          ar3 = pop()
          *ar3 = b
          b = *ar3
          poly(*ar3+%)

end_block:
          goto start
.end
```

Programme 2

```
      .mmregs
      .width  80
      .length  55

      .setsect ".text",0x1800,0
      .sect".text"

     SP  = #0ffah

start:    a = #0fffffffcbh
          @brc = #60
          AR3 = #100h
          AR2 = #100h
          BLOCKREPEAT(end_block-1)

          poly(*ar3+%)
          a = a & *ar3
```

```
        BK = data(300h)
        macp(*ar3-,1234,a)
        a = -a
        push(*ar3+)
        ar3 = pop()
        *ar3 = b
        *ar2 = *ar3
        poly(*ar3+%)

end_block:
        goto start
.end
```

Programme 3

```
    .mmregs
    .width  80
    .length  55

    .setsect ".text",0x1800,0
    .sect".text"

   SP  = #0ffah

start:  a = #0fffffffcbh
        @brc = #60
        AR3 = #100h
        AR2 = #100h
        BLOCKREPEAT(end_block-1)

        poly(*ar3+%)
        a = a \\ CARRY
        BK = data(300h)
        ar3 = #100
        a = -a
        push(*ar3+)
        ar3 = pop()
        *ar3 = b
        ar3 = #100
        a = a \\ CARRY

end_block:
        goto start
.end
```

Programme 4

```
    .mmregs
    .width  80
    .length  55

    .setsect ".text",0x1800,0
```

```
        .sect".text"

       SP  = #0ffah

start:  a = #0fffffffcbh
        @brc = #60
        AR3 = #100h
        AR4 = #100h
        BLOCKREPEAT(end_block-1)

        lms(*ar3+,*ar4+)
        a = a \\ CARRY
        BK = data(300h)
        ar3 = #100
        a = -a
        firs(*ar3+,*ar4+,1234)
        macp(*ar4-,1234,a)
        *ar3 = b
        firs(*ar3+,*ar4+,1234)
        a = a \\ CARRY

end_block:
        goto start
.end
```

Programme 5

```
        .mmregs
        .width  80
        .length  55

        .setsect ".text",0x1800,0
        .sect".text"

       SP  = #0ffah

start:  a = #0fffffffcbh
        @brc = #60
        AR3 = #100h
        AR4 = #100h
        BLOCKREPEAT(end_block-1)

        a = -a
        a = a \\ CARRY
        BK = data(300h)
        mar(*ar3+)
        a = -a
        firs(*ar3+,*ar4+,1234)
        mar(*ar3+)
        a = a - *ar3+
        firs(*ar3+,*ar4+,1234)
```

```
        a = a \\ CARRY

end_block:
        goto start
.end
```

Programme 6

```
      .mmregs
       width  80
      .length  55

      .setsect ".text",0x1800,0
      .sect".text"

     SP  = #0ffah

start:   a = #0fffffffebh
         @brc = #60
         AR3 = #100h
         AR4 = #100h
         BLOCKREPEAT(end_block-1)

         a = -a
         cmps(a,*ar3)
         BK = data(300h)
         mar(*ar3+)
         a = -a
         firs(*ar3+,*ar4+,1234)
         mar(*ar3+)
         a = a - *ar3+
         a = a - *ar3
         cmps(a,*ar3)

end_block:
         goto start
.end
```

Programme 7

```
      .mmregs
      .width  80
      .length  55

      .setsect ".text",0x1800,0
      .sect".text"

     SP  = #0ffah

start:   a = #0fffffffebh
         @brc = #60
```

```
        AR3 = #100h
        AR4 = #100h
        BLOCKREPEAT(end_block-1)

        a = -a
        cmps(a,*ar3)
        BK = data(300h)
        mar(*ar3+)
        macp(*ar4-,1234,a)
        firs(*ar3+,*ar4+,1234)
        mar(*ar3+)
        cmps(a,*ar3)
        a = a - *ar3
        a = a + *ar3

end_block:
        goto start
.end
```

Programme 8

```
      .mmregs
      .width  80
      .length  55

      .setsect ".text",0x1800,0
      .sect".text"

     SP  = #0ffah

start:  a = #0fffffffcbh
        @brc = #60
        AR3 = #100h
        AR4 = #100h
        BLOCKREPEAT(end_block-1)

        a = a + #01h
        cmps(a,*ar3)
        a = a ^ *ar3
        mar(*ar3+)
        macp(*ar4-,1234,a)
        firs(*ar3+,*ar4+,1234)
        a = a + #01h
        saturate(a)
        mar(*ar3+)
        a = a + *ar3

end_block:
        goto start
.end
```

Programme 9

```
        .mmregs
        .width  80
        .length  55

        .setsect ".text",0x1800,0
        .sect".text"

       SP  = #0ffah

start:  a = #0fffffffcbh
        @brc = #60
        AR3 = #100h
        AR4 = #100h
        BLOCKREPEAT(end_block-1)

        b = *ar3
        cmps(a,*ar3)
        a = a ^ *ar3
        *ar4+ = *ar3+
        *ar3+ = hi(a)
        firs(*ar3+,*ar4+,1234)
        a = a + #01h
        *ar4+ = *ar3+
        mar(*ar3+)
        b = *ar3

end_block:
        goto start
.end
```

Programme 10

```
        .mmregs
        .width  80
        .length  55

        .setsect ".text",0x1800,0
        .sect".text"

       SP  = #0ffah

start:  a = #0fffffffcbh
        @brc = #60
        AR3 = #100h
        AR4 = #100h
        BLOCKREPEAT(end_block-1)

        a = *ar3
        cmps(a,*ar3)
        a = a ^ *ar3
```

```
        a = a | *ar3
        *ar3+ = hi(a)
        *ar3 = a
        TC = bit(*ar3+,15-12)
        a = a \\ CARRY
        mar(*ar3+)
        b = *ar3

end_block:
        goto start
.end
```

Bibliographie

[ABC+98] Allara A., Brandolese C., Fornaciari W., Salice F., Sciuto D. (1998). System-Level Performance Estimation Strategy for Sw and Hw, Proceedings of the 1998 IEEE International Conference on Computer Design, Oct 5-7, 1998, Austin, TX, , USA, 48-53

[AbJe98] Abbas Syed, Jenness Bob, A Proposal for a Reduced Complexity G.lite, T1E1.4/98-123.

[AdTh96] Adams Jay K., Thomas Donald E. (1996). The Design of Mixed Hardware/Software Systems, DAC '96. Proceedings of the 33rd annual conference on Design automation conference, 515-520

[ARM99] http://www.arm.com

[AsPr98] Asawaree Kalavade, Pratyush Moghe. (1998). A Tool for Performance Estimation of Networked Embedded End-Systems, Proceedings of the 35th annual conference on Design automation conference, 257-262.

[AtSi93] Athanos P., Silverman M.F. (1993). Processor Reconfiguration Through Instruction-Set Metamorphosis, Computer, Vol. 26, No. 3, 11-18.

[Axel97] J . Axelsson. (1997). Analysis and Synthesis of Heterogeneous Real-Time Systems, Linkopoing Studies in Science and Technology, Suède, 190 pages.

[Azad98] Azadet Kamran, Nicole Chris. (1998). Low-Power Equalizer Architectures for High-Speed Modems, IEEE Communications Magazine, vol. 36, 88-125.

[BaHo95] Barton Melbourne, Honig Michael L. (1995). Optimization of Discrete Multitone to Maintain Spectrum Compatibility with Other Transmission Systems on Twisted Copper Pairs, IEEE Journal of Selected Areas in Communications, Vol. 13, No. 9, 1558-1563.

[BaLa92] Ball Thomas, Larus James R. (1992). Optimally Profiling and Tracing Programs, Proceedings of the nineteenth annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages, January 19 - 22, 1992, Albequerque, NM, USA, 59-70.

[Barr93] Cipra Barry A. (1993). The Ubiquitous Reed-Solomon Codes, SIAM News, vol. 26, No. 1, http://www.cs.utk.edu/~shuford/terminal/reed_solomon_codes.html

[BDL97] Bolsens Ivo, De Man Hugo J., Lin Bill, Van Rompaey Karl, Vercauteren Steven, Verkest Diederik. (1997). Hardware/Software Co-Design of Digital Telecommunication Systems, Proceedings of the IEEE, Vol. 85, No. 3, 391-417.

[BDMM99] Biglieri Ezio, Divsalar Dariush, McLane Peter J., Simon Mavin K. (1999). Trellis-Coded Modulation in V.32, http://bugs.wpi.edu :8080/EE535/hwk96/hwk4cd96/li/li.html

[Beha99] Behavioral Synthesis for Low Power, http://www.ececs.uc.edu/~ddel/projects/pdss/proposal/node4.html

[Berg99] Bergamashi Reinaldo A. (1999). Behavioral Network Graph Unifying the Domain of High-Level and Logic Synthesis, DAC '99. Proceedings of the 36th ACM/IEEE conference on Design automation, 213-218.

[BGN97] Bharat P. Dave, Ganesh Lakshminarayana, Nijar K. Jha. (1997). COSYN : Hardware-Software Co-Synthesis of Embedded Systems, Proceedings of the 1997 34th Design Automation Conference.

[BLA96] Bennour Imed E., Langevin Michel, Aboulhamid El M. (1996). Performance Analysis for Hardware/Software Co-Synthesis, Conference on Electrical and Computer Engineering, CCECE'96, 162-165

[Blal97] Blalock Garrick. (1997). The BDTImark : A Measure of DSP Execution Speed, Berkeley Design Technology. http://www.bdti.com/articles/wtpaper.htm

[BLMS98] Balarin Felice, Lavagno Luciano, Murthy Praveen, Sangiovanni-Vincentelli Alberto. (1998). Scheduling for Embedded Real-Time Systems, IEEE Design & Test of Computers, January-March, 71-82.

[Boll99] Bollaert, Thomas (1999). La covérification, très utile pour la mise au point d'un pilote de port série, Electronique, Septembre, No. 95, 77-82.

[BrBr87] Brassard G., Bratley P. (1987). Algorithmique, conception et analyse, Les Presses de l'Université de Montréal, 346 pages.

[BrMa94] Brage Jens P., Madsen Jan. (1994). A Codesign Case Study in Computer Graphics, Proceedings of the 3rd International Workshop on Hardware/Software Codesign Sep 22-24 1994, Grenoble, 132-139.

[Brow99] Brown Randy (1999), Splitterless ADSL : Market, Requirements & Opportunities, AG Communication Systems, http://www.agcs.com/TechPapers/splitterless.htm

[ChBo98] Chou Pai, Borriello Gaetano. (1998). An Analysis-Based Approach to Composition of Distributed Embedded Systems, 6th International Workshop on Hardware/Software Codesign, 3-7.

[ChBo94] Chou Pai, Borriello Gaetano. (1994). Software Scheduling in the Co-Synthesis of Reactive Real-Time Systems, Proceedings of the 31st Design Automation Conference, 1-4.

[ChBo95] Chou Pai, Boriello Gaetano. (1995)., Interval Scheduling : Fine-Grained Code Scheduling for Embedded Systems, 32nd Design Automation Conference, 462-467.

[ChSu99] Chang Hyunman, Sunwoo Myung, A Low Complexity Reed-Solomon Architecture using the Euclid's Algorithm, Proceedings of the 1999 ISCAS IEEE International Synposium on Circuits and Systems, 1999.

[CIAD98] Consumer Installable ADSL : An In-Depth Look at G.Lite Technology, DSL by Orckit, December 1998. 10 pages.

[cind99] http://www.ee.princeton.edu/~yauli/cinderella-2.0/

[CLG96] K.-S. Chung, C. L. Liu, and R. K. Gupta. (1996). An Algorithm for Synthesis of System-Level Interface Circuits, Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design, 442-447.

[CLLC96] Chan Ming-Hwa, Lee Wen-Ta, Lin Mao-Chao, Chen Liang-Gee. (1996). IC Design of an Adaptive Viterbi Decoder, IEEE Transactions on Consumer Electronics, Vol. 42, No. 1, 52-61.

[CMC96] Curatelli Francesco, Mangeruca Leonardo, Chirico Marco. (1996). Specification and Management of Timing Constraints in Behavioral VHDL, Proceedings of the conference with EURO-VHDL'96 and exhibition on European Design Automation, 522.

[CMP99] Chunho Lee, Mangione-Smith William, Potkonjak Miodrag. (1999). Power Efficient Mediaprocessors : Design Space Exploration, DAC '99. Proceedings of the 36th ACM/IEEE conference on Design automation conference, 321-326.

[CMR97] Catania Vincenzo, Malgeri Michele, Russo Marco. (1997). Applying Fuzzy Logic to Codesign Partitioning, IEEE Micro, Vol. 17, No. 3, 62-70.

[COB95] P. Chou, R. Ortega, G. Borriello. (1995) Interface Co-Synthesis Techniques for Embedded Systems, IEEE/ACM International Conference on Computer-Aided Design, Santa Clara, CA, 280-287.

[Cole99] Cole Terry L. (1999). Technology Extensions for T1.413/G.dmt/G.lite to Support Consumer Mid-band CO and CP Modems, T1E1.4/97-405

[CWB94] Chou Pai, Walkup Elizabeth, Boriello Gaetano. (1994). Scheduling for Reactive Real-Time Systems, IEEE Micro, Vol. 14, No. 4, 37-47.

[CZG99] Chinosi Mauro, Zafalon Roberto, Guardiani Carlo. (1999). Parallel Mixed-Level Power Simulation Based on Spatio-Temporal Circuit Partitioning, DAC '99. Proceedings of the 36th ACM/IEEE conference on Design automation, 562-567.

[DaPe95] Davies W.S., Peacock M.S. (1995). Simple Test of DMT ADSL interleaved FEC provisions, Electronics Letters, Vol. 31, No. 25, 2152-2155.

[DBL+96] De Man H., Bolsens I., Lin B., Van Rompaey K., Vercauteren S. (1996). Verkest D., Co-Design of DSP Systems, Hardware/Software Co-Design, Kluwer Academic Publisher, Netherlands, 75-194.

[DEA+99] Desmet Dirk, Esvelt Michiel, Avasare Prabhat, Verkest Diederik, De Man Hugo. (1999). Timed Executable System Specificatino of an ADSL Modem using a C++ based Design Environment : A Case Study, Proceedings of the seventh international workshop on Hardware/software codesign, 38-42.

[DeGu97] De Micheli Giovanni, Gupta Rajesh K. (1997). Hardware/Software Co-Design, Proceeding of the IEEE, Vol. 85, No. 3, 349-365.

[DeHa98] Del Castillo Giuseppe, Hardt Wolfram. (1998). Fast Dynamic Analysis of Complex HW/SW Systems Based on Abstract State Machine Models, CODES/CASHE '98. Proceedings of the sixth international workshop on Hardware/software codesign, 77-81

[DeMi94] De Micheli Giovanni. (1994). Synthesis and Optimization of Digital Circuits, McGraw Hill, États-Unis, 579 pages.

[CHB+99] I. Campagna, Y. Henneault, G. Bois, E.M. Aboulhamid, J. Baillargé, P. Yousefpour. (1999). A Hardware/Software Codesign Case Study Using VSIA Recommendations, Micronet Annual Workshop, 26-27 avril 1999.

[DIMJ97] J-M Daveau, T. Ben Ismail, G. Marchioro, A.A. Jerraya. (1997) Protocol Selection and Interface Generation for HW-SW Codesign, IEEE Transactions on Very Large Scale Integration (VLSI) Systems vol. 5 no. 1, 136-144.

[dixi99] http://www.cc.iastate.edu/olc_answers/programming/development/pixie_prof.html

[DJG98] De Veciana G., Jacome M., Guo J.-H. (1998). Hierarchical Algorithms for Assessing Probabilistic Constraints on System Performance, Proceedings of the 35th annual conference on Design automation, 251 - 256.

[DMT99a] Discrete Multitone (DMT) vs. Carrierless Amplitude/Phase (CAP) Line Codes, http://www.aware.com/technology/whitepapers/dmt.html.

[DMT99b] DMT & Cap, Explaining the Technologies, Interoperability, Status and Issues in xDSL, http://nwd2www1.analog.com/tepapers/whitepaper_html.

[DMTC99] DMT & CAP, Explaining the Technologies, http://www.analog.com/publications/whitepapers/whitepaper_html/explaining.html

[DMVJ97] J.M. Daveau, G. Marchioro, C. Valderrama, A. Jerraya. (1997) VHDL Generation from SDL Specification, XIII IFIP Conference on CHDL.

[DRG97] Dasdan Ali, Ramanathan Dinesh, Gupta Rajesh K. (1998). Rate Derivation and Its Applications to Reactive, Real-Time Embedded Systems, DAC '98. Proceedings of the 35th annual conference on Design automation conference, 263-268.

[DSLK99] DSL Knowledge Center, How Does ADSL Work, Orckit Communications, http://www.orckit.com/how_does_ads_works.html.

[EcMe99] Eckhardt Uwe, Merket Renate. (1999). Hierarchical Algorithm Partitioning at System Level for an Improved Utilization of Memory Structures, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 18, No. 1, 14-24.

[EHB93] Ernst R., Henkel J., Benner T. (1993). Hardware-Software Cosynthesis for Microcontrollers, IEEE Design & Test Computer, December 1993, 64-75.

[EHB93] Ernst Rolf, Henkel Jorg, Benner Thomas. (1993). Hardware-Software Cosynthesis for Microcontrollers, IEEE Design & Test of Computers, Vol. 10, No. 4, 64-75.

[Emer99] Emery Mark. (1999). Infrastructure for DSL Solutions, AG Communication Systems, 9 pages, http://www.agcs.com/TechPapers/dsl_inf.htm.

[Erns98] Ernst Rolf. (1998). Codesign of Embedded Systems : Status and Trends, IEEE Design & Test of Computer, Vol. 15, No. 2, 45-54.

[EyBi98] Eyre Jennifer, Bier Jeff. (1998). DSP Processors Hit the Mainstream, Computer, August 1998, 51-59.

[FFM+99] Ferrandi Fabrizio, Fummi Franco, Macii Enrico, Poncino Massimo, Sciuto Donatella, Power Estimation of Behavioral Descriptions.

[FGS+98] Fornaciari William, Gubian Paolo, Sciuto Donatella, Silvano Christina. (1998). Power Estimation of Embedded Systems : A Hardware/Software Codesign Approach, IEEE Transactions on Very Large Scale Integration Systems, Vol. 6, No2, 266-275.

[Flem95] Fleming Chip. (1999). A Tutorial on Convolutional Coding with Viterbi Decoding, Spectrum Application, 25 pages.

[FSS99] Fornaciari William, Sciuto Donatella, Silvano Cristina. (1999). Power Estimation for Architectural Exploration of HW/SW Communication on System-Level Buses, CODES '99. Proceedings of the seventh international workshop on Hardware/software codesign, pages 152-156.

[GaRa94] Gajski Daniel D., Ramachandran Loganath. (1994). Introduction to High-Level Synthesis, IEEE Design & Test of Computers, Vol. 11, No. 4, 44-54.

[GaVa95] Daniel D. Gajski, Frank Vahid. (1995). Specification and design of embedded hardware/software systems, IEEE Design & Test of Computers, vol. 12, no. 1, 53.

[GCM92] R. K. Gupta C. Coelho and G. De Micheli. (1992). Synthesis and Simulation of Digital Systems Containing Interacting Hardware and Software, In Proceedings of the 29th ACM/IEEE conference on Design automation, Anaheim, 225-230.

[Gebo94] Gebotys Catherine. (1994). An Optimization Approach to the Synthesis of Multichip Architectures, IEEE Transactions on Very Large Scale Integration Systems, Vol. 2, No.1, March 1994, 11-19.

[GENV99] General and VDSL Tutorial, http://www.adsl.com/general_tutorial.html

[Glit99] G.Lite White Paper, DSL Knowledge Center, Orckit Communication, http://www.orckit.com/glite.html

[GMM+99] Genini L., Macii A., Macii E., Poncino M., Scarsi R. (1999). Synthesis of Low-Overhead Interfaces for Power-Efficient Communication over Wide Buses, DAC '99. Proceedings of the 36th ACM/IEEE conference on Design automation, 128-133.

[Govi99] Govindarajan Sriram, http://www.ececs.uc.edu/~ddel/projects/dss/hls_paper/

[GuCh97] Gutnik Vadim Chandrakasan Anantha. (1997). Embedded Power Supply for Low-Power DSP, IEEE Transaction on Very Large Scale Integration System, Vol. 5, No. 4, 425.

[GuDe93] Gupta R.K., De Micheli G. (1993). Hardware-Software Cosynthesis for Digital Systems, IEEE Design & Test of Computer, 29-41.

[GuDe93] Gupta Rajesh K., De Micheli Giovanni. (1993). Hardware-Software Cosynthesis for Digital Systems, IEEE Design and Test of Computers, Vol. 10, No. 3, 29-41.

[GuLi97] Gupta Rajesh K., Liao Stan Y., Using a Programming Language for Digital System Design, IEEE Design & Test of Computers, April-June 1997, 72-80.

[GVNG94] Gajski Daniel D., Vahid Frank, Narayan Sanjiv, Gong Jie, Specification and Design of Embedded Systems, PTR Prentice Hall, New Jersey, 1994, 450 pages.

[Harb99] Harbison Samuel P. (1999) System-Level Hardware/Software Trade-Offs, Proceedings of the 1999 36th Annual Design Automation Conference, 258-259.

[Hawl97] Hawley George T. (1997). Systems Considerations for the Use of xDSL Technology for Data Access, IEEE Communications Magazine, vol. 35, No. 3, 56-60.

[HBKG98] Hollstein Thomas, Becker Jurgen, Kirschbaum Andreas, Glesner Manfred. (1998). HiPART : A New Hierarchical Semi-Interactive HW/SW Partitioning Approach with Fast Debugging for Real-Time Embedded Systems, CODES/CASHE '98. Proceedings of the sixth international workshop on Hardware/software codesign, 29-33.

[HeErb95] Henkel Jorg, Ernst Rolf. (1995). High-Level Estimation Techniques for Usage in Hardware/Software Co-Design, ISSS '95. Proceedings of the eighth international symposium on System synthesis, 122-127.

[HeEr97] Henkel Jorg, Ernst Rolf. (1997). A Hardware/Software Partitioner using a Dynamically Determined Granularity, DAC '97. Proceedings of the 34th annual conference on Design automation, 691-696

[HeEra95] Henkel Jorg, Ernst Rolf. (1995). A Path-Based Technique for Estimating Hardware Runtine in HW/SW-Cosynthesis, Proceedings of the eighth international symposium on System synthesis, 116–121.

[HeLi98] Henkel Jorg, Li Yanbing. (1998). Energy-Conscious HW/SW-Partitioning of Embedded Systems : A Case Study on an MPEG-2 Encoder, CODES/CASHE '98. Proceedings of the sixth international workshop on Hardware/software codesign, 23-27.

[HeSa97] Hendry D.C., Sananikone D.S. (1997). Hardware/Software Partitioning of Embedded Systems with Multiple Hardware Processes, IEE Proceedings Computers and Digital Techniques, Vol. 144, No. 5, 285-294.

[HFP99] Harrison Jana, Fife Elizabeth, Pereira Francis, Worthington Richard. (1999). ADSL : Prospects and Possibilities, Center for Telecommunications Management University of Southern California, http://www.adsl.com/mrp_exec_summary.html.

[HHE94] Herrmann D., Henkel J., Ernst R. (1994). An Approach to the Adaptation of Estimated Cost Parameters in the Cosyma System, Proceedings of the 3rd International Workshop on Hardware/Software Codesign, 100-107.

[High99] High-speed Access : xDSL and Other Alternatives for Last Mile Access, ISP Partner Program, Nortel Networks, http://www1.nortelnetworks.com/pen/isp/goals/hispeed.htm

[Hohh97] Ken Hohhof. (1997). Framework for Splitterless ADSL, T1E1.4/97-431

[HsPe98] Hsieh Cheng-Ta, Pedram Massoud. (1998). Microprocessor Power Estimation Using Profile-Driver Program Synthesis, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 17, No 11, 1080-1089.

[HWSW99] Huang Jin-Chuan, Wu Chien-Ming, Shieh Ming-Der, Wu Chien-Hsing. (1999) An Area-Efficient Versatile Reed-Solomon Decoder for ADSL, Proceedings of the 1999 ISCAS IEEE International Synposium on Circuits and Systems.

[JeYo98] Jeong Byung-Jang, Yoo Kyung-Hyun. (1998). Digital RFI Canceller for DMT based VDSL, Electronics Letters, Vol. 34, No. 17, 1640-1641.

[JGIT99] Jia Lihong, Gao Yonghong, Isoaho Jouni, Tenhunen Hannu. (1999) Design of a Super-Pipelined Viterbi Decoder, Proceedings of the 1999 ISCAS IEEE International Synposium on Circuits and Systems.

[JPP92] Jain Rajiv, Parker Alice C., Park Nohbyung. (1992). Predicting System-Level Area and Delay for Pipelined And Nonpipelined Designs, IEEE Transactions on Computer-Aided Design, Vol. 11, No. 8, 955-965.

[Kenn98] Martin Kenneth. (1998). Small Side-Lobe Filter Design for Multitone Data-Communication Applications, IEEE Transactions on Circuits and Systems II : Analog and Digital Signal Processing, Vol. 45, No. 8, 1155-1161.

[KnMa98] Knudsen Peter Voigt Madsen Jan. (1998). Communication Estimation for Hardware/Software Codesign, Proceedings of the 1998 6th International Workshop on Hardware/Software Codesign, 55-59.

[KnPa96] Knieser Michael J., Papachristou Christos A. (1996)., COMET : A Hardware-Software Codesign Methodology, Proceedings of the conference with EURO-VHDL'96 and exhibition on European Design Automation, 178.

[KPL95] Kalavade Asawaree, Pino José Luis, Lee Edward A. (1995). Managing Complexity in Heterogeneous System Specification, Simulation, and Synthesis, Proceedings Special Sessions Proceedings of the 1995 International Conference on Acoustics, Speech, and Signal Processing, 2833-2836.

[KrRa98] Krishna Vamsi, Ranganathan N. (1998). A Methodology for High Level Power Estimation and Exploration, Proceedings of the 1998 8th Great Lakes Symposium on VLSI, 420-425.

[KsHo94] Kshischang Frank, Horn Gavin. (1994). A Heuristic for Ordering a Linear Block Code to Minimize Trellis State Complexity, Proc 32 nd Annual Allerton Conf. on Communications, Control and Computing, 75-84.

[KuDe92] Ku David C., De Micheli Giovanni. (1992). Relative Scheduling Under Timing Constraints : Algorithms for High-Level Synthesis of Digital Circuits. IEEE Transactions on Computer-Aided Design, Vol. 11, No. 6, 696-717.

[Kutt96] Kuttner Carolyn. (1996). Hardware-Software Codesign Using Processor Synthesis. IEEE Design & Test of Computers, Vol. 13, No. 3, 43-53.

[LiMa97] Li Yau-Tsum Steven, Malik Sharad. (1997). Performance Analysis of Embedded Software Using Implicit Path Enumeration, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 16, No. 12. 1477-1487.

[LlGo98] Llopis Rafael Peset, Goossens Kees. (1998). The Petrol Approach to High-Level Power Estimation, ISLPED '98. Proceedings 1998 international symposium on Low power electronics and design, 130-132.

[LRD+99] Lajolo Marcello, Raghunathan Anand, Dey Sujit, Lavagno Luciano, Sangiovanni-Vincentelli Alberto, Efficient Power Estimation Techniques for HW/SW Systems.

[LRK+99] Lakshmirayana Ganesh, Raghunathan Anand, Khouri Kamal, Jha Niraj, Dey Sujit. Common-Case Computation : A High-Level Technique for Power and Performance Optimization, Proceedings of the 36th ACM/IEEE conference on Design automation, 1999, Pages 56 – 61.

[LuCa96] Luthi Eric, Casseau Emmanuel, High Rate Soft Output Viterbi Decoder, Proceedings of European Design and Test Conference Proceedings of the 1996 European Design & Test Conference Mar 11-14, 1996 Paris, p 315-319.

[MaKn95] Martin Raul San, Knight John P. (1995). Power-Profiler : Optimizing ASICs Power Consumption at the Behavioral Level, DAC '95. Proceedings of the 32nd ACM/IEEE conference on Design automation, 42-47.

[MaKn96] Martin Raul San, Knight John. (1996). Optimizing Power in ASIC Behavioral Synthesis, IEEE Design & Test of Computers, 58-70.

[MaPe98] Macii Enrico, Pedram Massoud. (1998). High-Level Power Modeling, Estimation, and Optimization, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 17, No. 11.

[MFT+96] Mecha Hortensia, Fernandez Milagros, Tirado Francisco, Septien Julio, Mozos Daniel, Olcoz Katzalin. (1996). A Method for Area Estimation of Data-Path in High Level Synthesis, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 15, No.2.

[MMP96] Marculescu Diana, Marculescu Radu, Pedram Massoud. (1996). Information Theoretic Measures for Power Analysis, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 15, No. 6.

[ModA99] Modem ADSL, Alcatel, http://www.alcatel.com/telecom/asd/36925/aaaa00064J925del&NS-doc-offset=3&

[MPS98] Macii Enrico, Pedram Massoud, Somenzi Fabio. (1998). High-Level Power Modeling, Estimation and Optimization, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 17, No. 11, p. 1061.

[MYR96] Melsa Peter J.W. Younce Richard C., Rohrs Charles E. (1996). Impulse Responjse Shortening for Discrete Multitone Transceivers, IEEE Transactions on Communications, Vol. 44, No. 12, 1662-1672.

[Najm96] Najm Farid N. (1996). Towards a High-Level Power Estimation Capability, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 15, No. 6, p. 588.

[NeNa96] Nemani Mahadevamurty, Najm Farid N. (1996). Towards a High-Level Power Estimation Capability, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 15, No. 6.

[NeNa99] Nemani Mahadevamurty, Najm Farid N. (1999). High-Level Area and Power Estimation for VLSI Circuits, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 18, No.6, 697-713.

[NiMa97] Niemann Ralf, Marwedel Peter. (1997). Hardware/Software Partitioning using Integer Programming, Design Automation for Embedded Systems v 2 n 2 Mar 1997 Kluwer Academic Publishers Dordrecht Netherlands, 165-193.

[NiMa98] Ralf Niemann, Peter Marwedel. (1998). Hardware/Software co-design for data flow dominated embedded systems, Kluwer Academic Publishers, Boston, 244

[OBE99] Osterling Achim, Benner Thomas, Ernst Rolf, Code Generation and Context Switching for Static Scheduling of Mixed Control and Data Oriented HW/SW Systems.

[ODDe97] O'Donnell Ian, Yee Dennis. (1997). FFT Implementation Exploration, 1997, 24 pages.

[OKDX95] Ohm Seong, Kurdahi Fadi, Dutt Nikil, Xu Min. (1995). A comprehensive Estimation Technique for High-Level Synthesis, ISSS '95. Proceedings of the eighth international symposium on System synthesis, pages 122-127.

[Oksm99] Oksman Vladimir. (1999). On VDSL Scalability and VDSL to ADSL Interoperability, T1E1.4/99-234, 8 pages.

[OrBo98] Ross B. Ortega and Gaetano Borriello. (1998). Communication Synthesis for Distributed Embedded Systems, In Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, p 437-444.

[PDN99] Panda Preeti Ranjan, Dutt Nikil D. Nicolau Alexandru. (1999). Local Memory Exploration and Optimization in Embedded Systems, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 18, No. 1, 3-13.

[PKP97] Potkonjak Miodrag, Kyosum Kim, Kamesh Karri. (1997). Methodology for Behavioral Synthesis-based Algorithm-level Design Space Exploration : DCT Case Study, DAC '97. Proceedings of the 34th annual conference on Design automation, 252-257.

[Plan99] Plank James. (1999). Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-like Systems, Departement of Computer Science, University of Tennessee, http://www.utk.edu/.

[Poit99] Poiteau Gwénaël (1999), Dissipation de puissance dans le TMS320C54, une approche logicielle. Rapport de stage, École Polytechnique de Montréal.

[Poli99] http://www-cad.eecs.berkeley.edu/Respep/Research/hsc/abstract.html

[QiPe99] Qiu Qinru. (1999). Pedram Massoud, Dynamic Power Management Based on Continuous-Time Markov Decision Processes, DAC '99. Proceedings of the 36th ACM/IEEE conference on Design automation, 555-561.

[Qpt99] http://www.cs.wisc.edu/~larus/qpt.html

[RaCa99] Radjassamy Radjakichenin, Carothers Jo Dale. (1999). A Fractal Compaction Algorithm for Efficient Power Estimation , Simulation, Vol. 72, No.5, p. 320.

[REED99] Reed-Solomon Codes, http://tcomwww.epfl.ch/~garcia/publications/PHDbook/nodes49.html

[RESE99] Research on Error-Correcting Codes, http://homepage.arl.mil/~retter/coding.html

[Roka99] Rokach Enud. (1999). QAM : The Choice for VDSL Transmission Line Code, http://www.orckit.com/qam.html

[RuJa98] Russell Jeffry, Jacome Margarida. (1998). Software Power Estimation and Optimization for High Performence, 32-bit Embedded Processors, Proceedings of the 1998 IEEE International Conference on Computer Design, 328-333.

[SaTz95] Sandberg Stuart D., Tzannes Michael A. (1995). Overlapped Discrete Multitone Modulation for High Speed Copper Wire Communications, IEEE Journal on Selected Areas in Communications, Vol. 13, No. 9, 1571-1585.

[Schl98] Schlett Manfred. (1998). Trends in Embedded-Microprocessor Design, Computer, August 1998, 44-49.

[Seam99] Seamless User Manual (1999), www.mentorg.com

[Shan97] Shanbhag Naresh R. (1997). A Mathetical Basis for Power-Reduction in Digital VLSI Systems, IEEE Transactions on Circuits and Systems II Analog and Digital Processing, Vol. 44, No. 11, 935-951.

[ShJa93] Sharma Alok, Jain Rajiv. (1993). Estimating Architectural Resources and Performance for High-Level Synthesis Applications, IEEE Transactions on Very Large Scale Integration Systems, Vol. 1, No. 2, p. 175.

[SoPa99] Song Leilei, Parhi Keshab K. (1999). Low-Energy Software Reed-Solomon Codecs Using Specialized Finite Fiels Datapath and Division-Free Berlekamp-Massey Algorithm, Proceedings of the 1999 ISCAS IEEE International Synposium on Circuits and Systems.

[SrBr91] Srivastona M.B., Brodersen R.W. (1991). Rapide-Prototyping of Hardware and Software in a Unified Framework, International conference on Computer-Aided Design, Los Alanitos, California, 152-155.

[STA98] Sarta Davide, Trifone Dario, Ascia Giuseppe, A Data Dependent Approach to Instruction Level Power Estimation.

[StIs99] Status & Issues in xDSL, Analog Devices, 5 pages, http://www.analog.com/publications/whitepapers/products/xDSL.html

[TCL99] Tsui Chi-Ying, Cheng Roger, Ling Curtis. (1999). Low Power ACS Unit Design for the Viterbi Decoder, Proceedings of the 1999 ISCAS IEEE International Synposium on Circuits and Systems.

[TEIN99] http://www.ti.com

[TMP+95] Tsui Chi-Ying, Monteiro José, Pedram Massoud, Devadas Srinivas, Despain Alvin M., Lin Bill. (1995). Power Estimation Mehods for Sequential Logic Circuits, IEEE Transactions on Very Large Scale Integration Systems, Vol 3, No. 3, 404-415.

[TMW94] Tiwari Vivek, Malik Sharad, Wolfe Andrew. (1994). Power Analysis of Embedded Software : A First Step Toward Software Power Minimization, IEEE Transactions on Very Large Scale Integration Systems, Vol. 2, No. 4, 437-445.

[Tomb97] Tomba L. (1997). Spectral Analysis in DMT-Based Transceivers, Electronics Letters, Vol. 33, No. 12, 1022-1023.

[TTMF95] Tien-Chien Lee Mike, Tiwari Vivek, Malik Sharad, Fujita Masahiro. (1995). Power Analysis and Low-Power Scheduling Techniques for Embedded DSP Software, Fujitsu scientific & technical journal, Vol. 31, No. 2, p. 215.

[TTMF97] Tien-Chien Lee Mike, Tiwari Vivek, Malik Sharad, Fujita Masahiro. (1997). Power Analysis and Minimization Techniques for Embedded DSP Software, IEEE Transactions on Very Large Scale Integration Systems, Vol. 5, No 1, 123-135.

[Univ99] Universal ADSL Working Group Questions and Answers, http://www.uawg.com/q_and_a.html

[user99] http://www.it.dtu.dk.lycos/users_guide

[VaGa95a] Vahid Frank, Gajski Daniel D., Closeness Metrics for System-Level Functional Partitioning, Proceedings of European design automation conference with EURO-VHDL '95 on EURO-DAC '95 with EURO-VHDL '95, 328-333.

[VaGa95b] Vahid Frank, Gajski Daniel D. (1995). Clustering for Improved System-Level Functional Partitioning, ISSS '95. Proceedings of the eighth international symposium on System synthesis, 28-35.

[VaGa95c] Vahid Frank, Gajski Daniel D. (1995). Incremental Hardware Estimation During Hardware/Software Functional Partitioning, IEEE Transactions on Very Large Scale Integration Systems, Vol. 3, No. 3, 459-464.

[VGG94] Vahid Frank, Gong Jie, Gajski Daniel D. (1994). A Binary-Constraint Search Algorithm for Minimizing Hardware during Hardware/Software Partitioning, EURO-DAC '94. Proceedings of the conference on European design automation, 214-219.

[VITA99] Viterbi Algorithme, http://www-ee.eng.hawaii.edu/~msmith/Viterbi/HTML/ch3/ch3.htm

[VITE99] The Viterbi Algorithm, http://lcmwww.epfl.ch/Viterbi/decoding.html

[YiYu98] Yin changchuan, Yue Guangxin. (1998). Optimal Impulse response shortening for discrete multitone transceivers, Electronics Letters, Vol. 34, No. 1, 35-36.

[YMS+98] Young James Shin, MacDonald Josh, Shilman Michael, Tabbara Abdallah, Hilfinger Paul, Newton A. Richard. (1998) Design and Specification of Embedded Systems in Java Using Successive, Formal Refinement, DAC '98. Proceedings of the 35th annual conference on Design automation, 70-75.

[YoKi98] Yoo Sungjoo Kiyoung Choi. (1998). Optimistic Distributed Timed Cosimulation Based on Thread Simulation Model, CODES/CASHE '98. Proceedings of the sixth international workshop on Hardware/software codesign, 71-75

[ZAC95] Zogakis Nicholas, Aslanis James T., Cioffi John M. (1995). A Coded and Shaped Discrete Multitone System, IEEE Transactions on Communications Vol. 43, No. 12, 2941-2949.

[ZhGa99] Zhu Jianwen, Gajski Daniel D. (1995). Soft Scheduling in High Level Synthesis, DAC '99. Proceedings of the 36th ACM/IEEE conference on Design automation, 219-224.

[ZoCi96] Zogakis Nicholas, Cioffi John. (1996). The Effect of Timing Jitter on the Performance of a Discrete Multitone System, IEEE Transactions on Communications, vol. 44, No. 7, 799-808.